OTRS Developer Manual

发布 ***7.0***

OTRS AG

2020 年 **05** 月 **24** 日

目录

CHAPTER 1

入门

OTRS 是一个多平台的 Web 应用程序框架，最初是为故障单系统开发的 。它支持不同的 Web 服务器和数据库 。

本手册介绍了如何基于 OTRS 样式指南开发自己的 OTRS 模块和应用程序 。

1.1 开发环境

为了便于编写 OTRS 扩展模块，必须创建开发环境 。可以在 GitHub 上找到 OTRS 和其它公共模块的源代码 。

1.1.1 检出框架

首先，必须创建一个可以存储模块的目录 。然后使用命令行切换到新目录，并使用以下命令检出它们：

```
# for git master
shell> git clone git@github.com:OTRS/otrs.git -b master
# for a specific branch like OTRS 3.3
shell> git clone git@github.com:OTRS/otrs.git -b rel-3_3
```

对于您的开发环境，也可以检出 `module-tools` 模块（ 来自 GitHub ）。它包含许多有用的工具：

```
shell> git clone git@github.com:OTRS/module-tools.git
```

Please configure the OTRS system according to the installation instructions.

1.1.2 有用的工具

强烈建议 OTRS 开发使用这两个模块：

- OTRS 代码策略

- Fred

OTRSCodePolicy 是一个代码质量检查器，可以为 OTRS 开发团队强制使用通用编码标准。如果您计划做贡献，强烈建议您使用它。您可以将它用作独立的测试脚本，甚至可以将其注册为每次创建提交时运行的 git commit hook。有关详细信息，请参阅 模块文档。

Fred 是一个小型开发助手模块，您可以将其实际安装或链接（如下所述）到您的开发系统中。它具有几个可以激活的有用模块，例如 SQL 记录器或 STDERR 控制台。您可以在其 模块文档 中找到更多详细信息。

顺便说一句，这些工具也是开源的，我们将为您可以贡献的任何改进感到高兴。

1.1.3 链接扩展模块

OTRS 和模块之间的明确分离对于正确开发是必要的。特别是当使用 git 克隆时，明确的分离是至关重要的。为了便于 OTRS 访问文件，必须创建链接。这是通过模块工具存储库目录中的脚本完成的。

示例：链接 *Calendar* 日历模块：

```
shell> ~/src/module-tools/link.pl ~/src/Calendar/ ~/src/otrs/
```

每当添加新文件时，必须如上所述链接它们。

链接完成后，必须重建系统配置以在 OTRS 中注册模块。还必须执行模块中的其它 SQL 或 Perl 代码。

例如：

```
shell> ~/src/otrs/bin/otrs.Console.pl Maint::Config::Rebuild
shell> ~/src/module-tools/DatabaseInstall.pl -m Calendar.sopm -a install
shell> ~/src/module-tools/CodeInstall.pl -m Calendar.sopm -a install
```

要从 OTRS 中删除链接，请输入以下命令：

```
shell> ~/src/module-tools/remove_links.pl ~/src/otrs/
```

1.2 架构概述

OTRS 框架是模块化的。下图显示了 OTRS 的基本层架构。

OTRS 通用接口继续 OTRS 模块化。下图显示了通用接口的基本层架构。

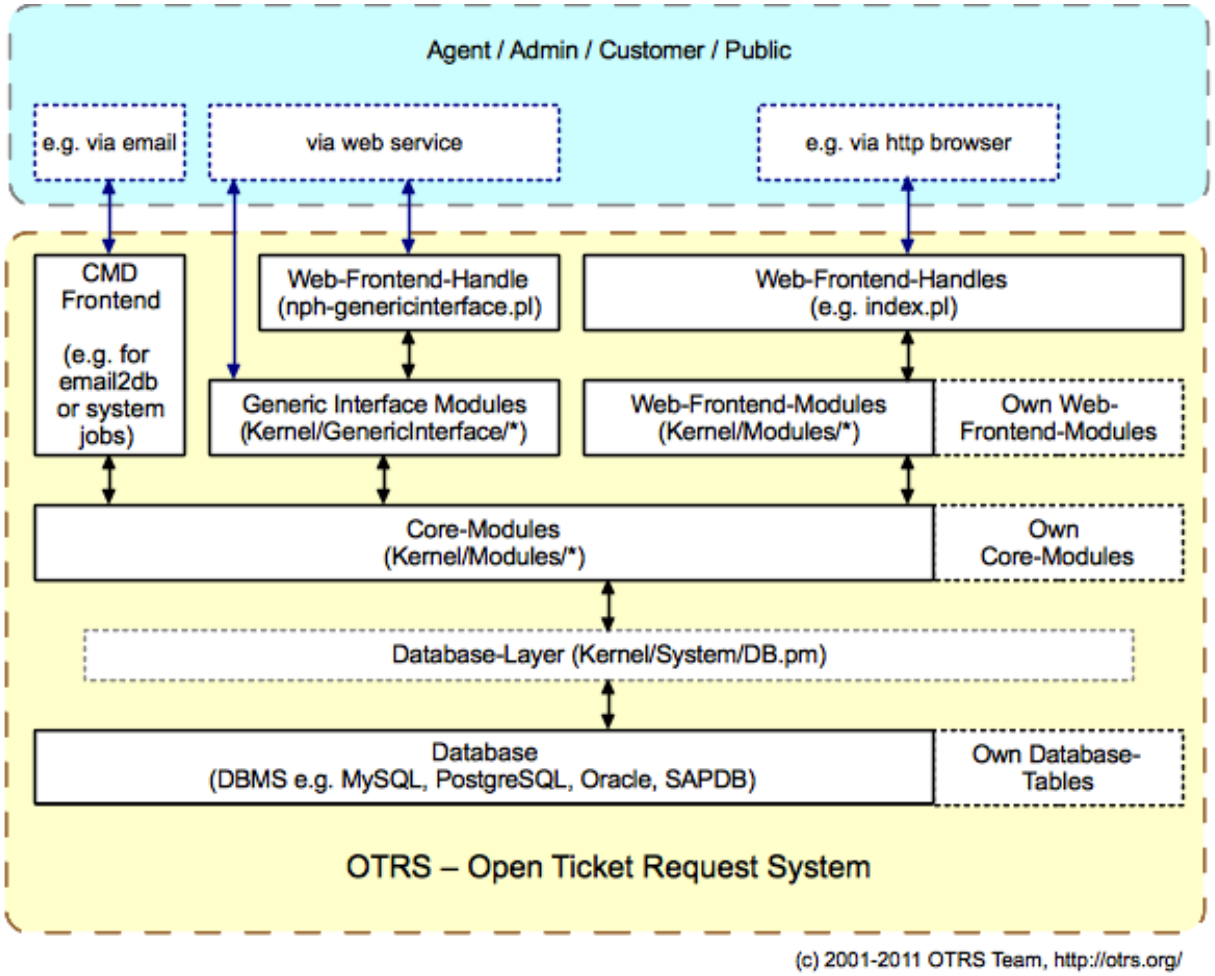
Agent / Admin / Customer / Public
e.g. via email
via web service
e.g. via http browser
CMD Frontend
(e.g. for email2db or system jobs)
Web-Frontend-Handle (nph-genericinterface.pl)
Web-Frontend-Handles (e.g. index.pl)
Generic Interface Modules (Kernel/GenericInterface/*)
Web-Frontend-Modules (Kernel/Modules/*)
Own Web-Frontend-Modules
Core-Modules (Kernel/Modules/*)
Own Core-Modules
Database-Layer (Kernel/System/DB.pm)
Database (DBMS e.g. MySQL, PostgreSQL, Oracle, SAPDB)
Own Database-Tables
OTRS – Open Ticket Request System
(c) 2001-2011 OTRS Team, http://otrs.org/

图 1: OTRS 架构

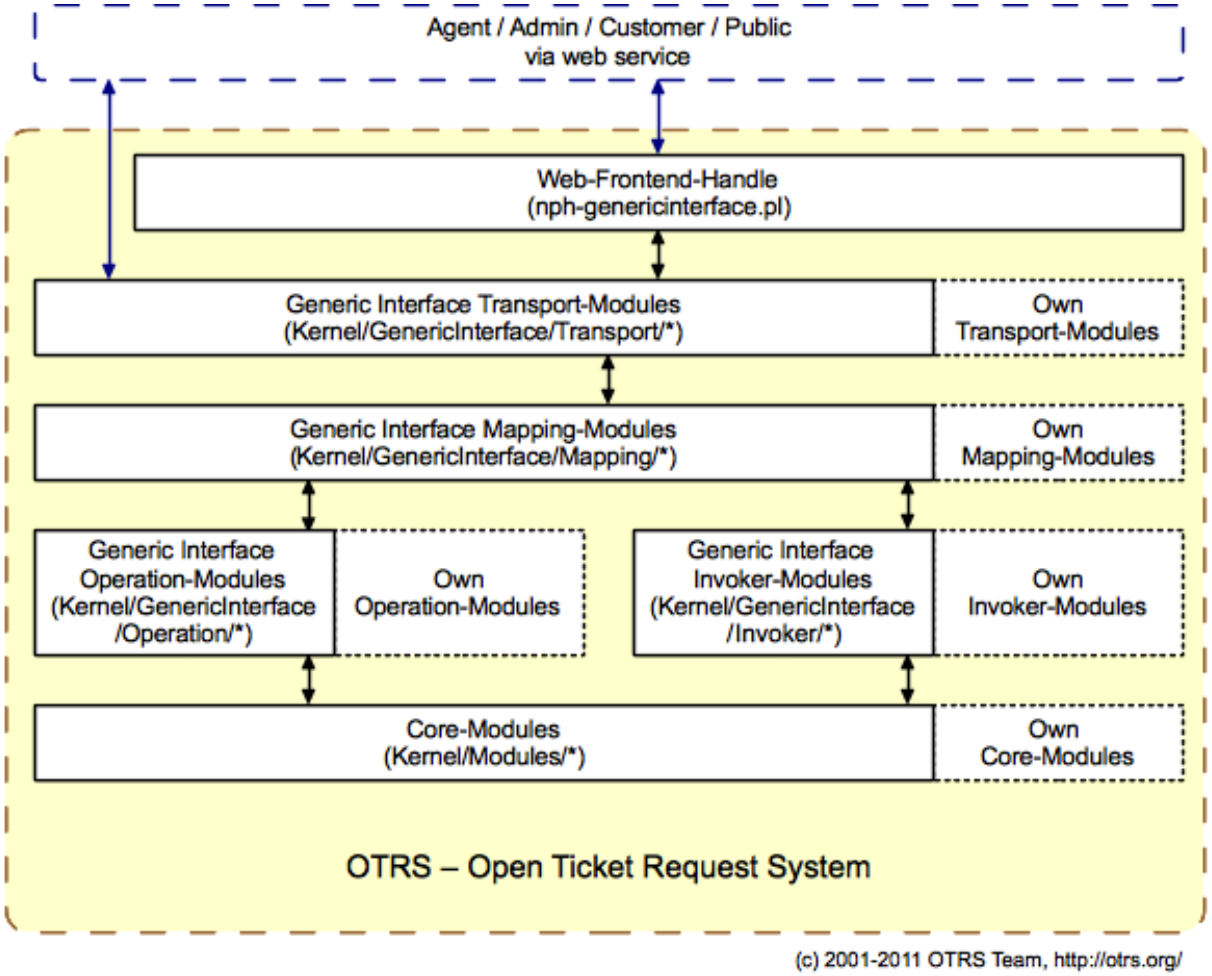
Agent / Admin / Customer / Public
via web service
Web-Frontend-Handle
(nph-genericinterface.pl)
Generic Interface Transport-Modules
(Kernel/GenericInterface/Transport/*)
Own
Transport-Modules
Generic Interface Mapping-Modules
(Kernel/GenericInterface/Mapping/*)
Own
Mapping-Modules
Generic Interface
Operation-Modules
(Kernel/GenericInterface
/Operation/*)
Own
Operation-Modules
Generic Interface
Invoker-Modules
(Kernel/GenericInterface
/Invoker/*)
Own
Invoker-Modules
Core-Modules
(Kernel/Modules/*)
Own
Core-Modules
OTRS – Open Ticket Request System
(c) 2001-2011 OTRS Team, http://otrs.org/

图 2: 通用接口架构

1.2.1 目录

目录	描述
bin/	命令行工具
bin/cgi-bin/	web handle
bin/fcgi-bin/	快速的 CGI Web 句柄
Kernel	应用程序代码库
Kernel/Config/	配置文件
Kernel/Config/Files	配置文件
Kernel/GenericInterface/	通用接口 API
Kernel/GenericInterface/Invoker/	通用接口的调用程序模块
Kernel/GenericInterface/Mapping/	用于通用接口的映射模块，例如 Simple
Kernel/GenericInterface/Operation /	通用接口的操作模块
Kernel/GenericInterface/Transport /	通用接口的传输模块，例如“HTTP SOAP”
Kernel/Language	语言翻译文件
Kernel/Scheduler/	调度程序文件
Kernel/Scheduler/TaskHandler	调度程序任务的处理程序模块，例如 GenericInterface
Kernel/System/	核心模块，例如日志 、工单
Kernel/Modules/	前端模块，例如 QueueView
Kernel/Output/HTML/	HTML 模板
var/	可变的数据
var/log	日志文件
var/cron/	cron 文件
var/httpd/htdocs/	带有 index.html 的 htdocs 目录
var/httpd/htdocs/skins/Agent/	服务人员界面的可用皮肤
var/httpd/htdocs/skins/Customer/	客户界面的可用皮肤
var/httpd/htdocs/js/	JavaScript 文件
scripts/	杂项文件
scripts/test/	单元测试文件
scripts/test/sample/	单元测试样本数据文件

1.2.2 文件

- .pl = Perl
- .pm = Perl 模块
- .tt = Template::Toolkit 模板文件
- .dist = 文件的默认模板
- .yaml or .yml = YAML 文件，用于 Web 服务配置

1.2.3 核心模块

核心模块位于 `$OTRS_HOME/Kernel/System/*` 下 。该层用于逻辑工作 。核心模块用于处理系统例程，如锁定工单和 创建工单 。一些主要的核心模块是：

- `Kernel::System::Config` 用于访问配置选项 。
- `Kernel::System::Log` 用于记录到 OTRS 日志后端 。
- `Kernel::System::DB` 用于访问数据库后端 。

- `Kernel::System::Auth` 用于检查用户认证 。
- `Kernel::System::User` 用于管理用户 。
- `Kernel::System::Group` 用于管理组 。
- `Kernel::System::Email` 用于发送电子邮件 。

有关更多信息，请访问 文档门户网站 。

1.2.4 前端句柄

浏览器 、Web 服务器和前端模块之间的接口 。可以通过 HTTP 链接使用前端模块 。

```
http://localhost/otrs/index.pl?Action=Module
```

1.2.5 前端模块

前端模块位于 `$OTRS_HOME/Kernel/Modules/*.pm` 下 。那里有两个公共函数 - `new()` 和 `run()`- 可以从前端句柄访问(例如 `index.pl`)。

`new()` 用于创建前端模块对象 。前端句柄为使用过的前端模块提供基本框架对象 。这些对象是以下示例：

- `ParamObject` 用于获取 WEB 表单参数 。
- `DBObject` 用于使用现有的数据库连接 。
- `LayoutObject` 用于使用模板和其它的 HTML 布局函数 。
- `ConfigObject` 用于访问配置设置 。
- `LogObject` 用于使用框架日志系统 。
- `UserObject` 用于获取当前用户的用户函数 。
- `GroupObject` 用于获取组函数 。

有关更多信息，请访问 文档门户网站 。

1.2.6 CMD 前端

CMD(命令行)前端就像 Web 前端句柄和 Web 前端模块一样(只是没有 `LayoutObject`)，使用核心模块进行系统中的某些操作 。

1.2.7 通用接口模块

通用接口模块位于 `$OTRS_HOME/Kernel/GenericInterface/*` 下 。通用接口模块用于处理系统上 Web 服务执行的每个部分 。通用接口的主要模块是：

- `Kernel::GenericInterface::Transport` 用于与远程系统交互 。
- `Kernel::GenericInterface::Mapping` 用于将数据转换为所需格式 。
- `Kernel::GenericInterface::Requester` 用于将 OTRS 作为 Web 服务的客户端 。
- `Kernel::GenericInterface::Provider` 用于将 OTRS 作为 Web 服务的服务器端 。
- `Kernel::GenericInterface::Operation` 用于执行提供者操作 。

- Kernel::GenericInterface::Invoker 用于执行请求者操作 。
- Kernel::GenericInterface::Debugger 使用日志条目跟踪 Web 服务通信 。

有关更多信息，请访问 文档门户网站 。

通用接口调用程序模块

通用接口调用程序模块位于 $OTRS_HOME/Kernel/GenericInterface/Invoker/* 下 。每个调用程序都包含在一个名为 Controller 的文件夹中 。此方法不仅有助于为内部类和方法定义名称空间，而且还有助于为文件名定义名称空间 。例如：$OTRS_HOME/Kernel/GenericInterface/Invoker/Test/ 是所有测试类型调用程序的控制器 。

通用接口调用程序模块用作后端，以创建远程系统执行操作的请求 。

有关更多信息，请访问 文档门户网站 。

通用接口映射模块

通用接口映射模块位于 $OTRS_HOME/Kernel/GenericInterface/Mapping/* 下 。这些模块用于将数据（ 键和值 ）从一种格式转换为另一种格式 。

有关更多信息，请访问 文档门户网站 。

通用接口操作模块

通用接口操作模块位于 $OTRS_HOME/Kernel/GenericInterface/Operation/* 下 。每个操作都包含在一个名为 Controller 的文件夹中 。此方法不仅可以为内部类和方法定义名称空间，还可以为文件名定义名称空间 。例如：$OTRS_HOME/Kernel/GenericInterface/Operation/Ticket/ 是所有工单类型操作的控制器 。

通用接口操作模块用作后端以执行远程系统请求的动作 。

有关更多信息，请访问 文档门户网站 。

通用接口传输模块

通用接口网络传输模块位于 $OTRS_HOME/Kernel/GenericInterface/Transport/* 下 。应将每个传输模块放在名为所用网络协议的目录中 。例如：HTTP SOAP 传输模块，位于 $OTRS_HOME/Kernel/GenericInterface/Transport/HTTP/SOAP.pm 中 。

通用接口传输模块用于向远程系统发送数据和从远程系统接收数据 。

有关更多信息，请访问 文档门户网站 。

1.2.8 调度程序任务处理程序模块

调度程序任务处理程序模块位于 $OTRS_HOME/Kernel/Scheduler/TaskHandler/* 下 。这些模块用于执行异步任务 。例如，GenericInterface 任务处理程序对 Apache 进程外的远程系统执行通用接口请求 。这有助于系统提高响应速度，防止出现可能的性能问题 。

有关更多信息，请访问 文档门户网站 。

1.2.9 数据库

数据库接口支持不同的数据库 。

对于 OTRS 数据模型，请参阅 /doc 目录中的文件 。或者，您可以在 GitHub 上查看数据模型 。

CHAPTER 2

OTRS 内部 - 如何工作

2.1 配置机制

OTRS 带有专用机制，可通过图形界面(系统配置)管理配置选项 。本节介绍内部如何工作以及如何提供新配置选项或更改现有默认值 。

2.1.1 Defaults.pm: OTRS 默认配置

OTRS 的默认配置文件是 Kernel/Config/Defaults.pm 。在没有部署的 XML 配置的情况下运行新安装的系统需要此文件，并且应该保持不变，因为它在框架更新时自动更新 。

2.1.2 自动生成的配置文件

在 Kernel/Config/Files 中，您可以找到一些自动生成的配置文件：

ZZZAAuto.pm XML 配置的当前值的 Perl 缓存(默认或由用户修改)。

ZZZACL.pm 来自数据库的 ACL 配置的 Perl 缓存 。

ZZZProcessManagement.pm 来自数据库的流程管理配置的 Perl 缓存 。

这些文件是当前系统配置的 Perl 表示法 。永远不要手动更改它们，因为它们由 OTRS 覆盖 。

2.1.3 XML 配置文件

在 OTRS 中，管理员可以通过系统配置配置的配置选项通过具有特殊格式的 XML 文件提供 。要转换旧 XML，您可以使用以下命令：

```
otrs> /opt/otrs/bin/otrs.Console.pl Dev::Tools::Migrate::ConfigXMLStructure
```

Kernel/Config/Files/ZZZAAuto.pm 文件是 XML 的缓存 Perl 版本，包含所有具有当前值的设置 。它可以这样（ 重新 ）生成：

```
otrs> /opt/otrs/bin/otrs.Console.pl Maint::Config::Rebuild
```

每个 XML 配置文件都具有以下布局：

```
<?xml version="1.0" encoding="utf-8" ?>
<otrs_config version="2.0" init="Changes">

    <!--  settings will be here -->

</otrs_config>
```

init 全局 “init“属性描述了应该加载配置选项的位置 。有不同级别可用，将按以下顺序加载/重载：

- Framework (用于框架设置，例如会话选项)
- Application（ 用于应用程序设置，例如工单选项 ）
- Config（ 用于扩展现有应用程序，例如 ITSM 选项 ）
- Changes（ 用于自定义开发，例如覆盖框架或工单选项 ）。

配置项被写为 Setting 元素，有一个 Description 、一个 Navigation 组（ 用于 GUI 中的基于树的导航 ）和它所代表的 Value 。这是一个例子：

```
<Setting Name="Ticket::Hook" Required="1" Valid="1">
    <Description Translatable="1">The identifier for a ticket, e.g. Ticket#,␣
↪Call#, MyTicket#. The default is Ticket#.</Description>
    <Navigation>Core::Ticket</Navigation>
    <Value>
        <Item ValueType="String" ValueRegex="">Ticket#</Item>
    </Value>
</Setting>
```

Required 如果设置为 1，则无法禁用配置设置 。

Valid 确定默认情况下配置设置是激活（ 1 ）还是非激活（ 0 ）。

ConfigLevel 如果设置了可选属性 ConfigLevel，则管理员可能无法编辑配置变量，具体取决于他自己的配置级别 。配置变量 ConfigLevel 设置管理员的技术经验水平 。它可以是 *100*（ 专家 ），*200*（ 高级 ）或 *300*（ 初学者 ）。作为应该为选项提供配置级别的指导原则，建议所有与外部交互配置相关的选项（ 如 Sendmail 、LDAP 、SOAP 等 ）的配置级别至少应为 *200*（ 高级 ）。

Invisible 如果设置为 1，则 GUI 中不显示配置设置 。

Readonly 如果设置为 1，则无法在 GUI 中更改配置设置 。

UserModificationPossible 如果 UserModificationPossible 设置为 1 ，管理员可以启用用户修改此设置（ 在用户首选项中 ）。

UserModificationActive 如果 UserModificationActive 设置为 1，则启用用户对此设置的修改（ 在用户首选项中 ）。您应该将此属性与 UserModificationPossible 一起使用 。

UserPreferencesGroup 使用 UserPreferencesGroup 属性来定义哪个组配置变量属于哪个（ 在 UserPreferences 用户首选项屏幕中 ）。您应该将此属性与 UserModificationPossible 一起使用 。

在右侧导航节点中放置设置的准则：

- 只在必要时才创建新节点 。如果可能，请避免仅使用极少数设置的节点 。

- 在第一个树级别，不应添加新节点。
- 不要直接在 Core 中放置新设置。它保留用于一些重要的全局设置。
- Core::* 可以采用包含相同主题设置的新组（如 Core::Email）或与同一实体相关（如 Core::Queue）。
- 所有事件处理程序注册都转到 Core::Event。
- 不要在 Frontend 中添加新的直接子节点。全局前端设置转到 Frontend::Base，只影响系统一部分的设置转到相应的 Admin、Agent、Customer 或 Public 子节点。
- 只影响一个屏幕的前端设置应该转到相关的屏幕（View）节点（如果需要，创建一个），例如 AgentTicketZoom 相关的设置转到 Frontend::Agent::View::TicketZoom。如果一个屏幕中有模块层与相关设置组，他们也会到这里的子组（例如 Frontend::Agent::View::TicketZoom::MenuModule 用于所有工单详情菜单模块注册）。
- 所有全局加载器设置都转到 Frontend::Base::Loader，屏幕特定的加载器设置为 Frontend::*::ModuleRegistration::Loader。

Value 元素的结构

Value 元素保存实际的配置数据有效负载。它们可以包含单个值、哈希或数组。

Item

一个 Item 元素包含一个数据。可选的 ValueType 属性确定哪种数据以及如何在 GUI 中向用户呈现它。如果没有指定 ValueType，则默认为 String。

请参阅值类型，了解不同值类型的定义。

```
<Setting Name="Ticket::Hook" Required="1" Valid="1">
    <Description Translatable="1">The identifier for a ticket, e.g. Ticket#,␣
↪Call#, MyTicket#. The default is Ticket#.</Description>
    <Navigation>Core::Ticket</Navigation>
    <Value>
        <Item ValueType="String" ValueRegex="">Ticket#</Item>
    </Value>
</Setting>
```

Array

使用此配置元素可以显示数组。

```
<Setting Name="SettingName">
    ...
    <Value>
        <Array>
            <Item Translatable="1">Value 1</Item>
            <Item Translatable="1">Value 2</Item>
            ...
        </Array>
    </Value>
</Setting>
```

Hash

使用此配置元素可以显示哈希值 。

```
<Setting Name="SettingName">
    ...
    <Value>
        <Hash>
            <Item Key="One" Translatable="1">First</Item>
            <Item Key="Two" Translatable="1">Second</Item>
            ...
        </Hash>
    </Value>
</Setting>
```

可以使用嵌套的数组/哈希元素(如数组哈希 、哈希数组 、数组哈希数组等)。下面是哈希数组的示例 。

```
<Setting Name="ExampleAoH">
    ...
    <Value>
        <Array>
            <DefaultItem>
                <Hash>
                    <Item></Item>
                </Hash>
            </DefaultItem>
            <Item>
                <Hash>
                    <Item Key="One">1</Item>
                    <Item Key="Two">2</Item>
                </Hash>
            </Item>
            <Item>
                <Hash>
                    <Item Key="Three">3</Item>
                    <Item Key="Four">4</Item>
                </Hash>
            </Item>
        </Array>
    </Value>
</Setting>
```

值类型

XML 配置设置支持各种类型的配置变量 。

String

```
<Setting Name="SettingName">
    ...
    <Value>
```

(下页继续)

(续上页)

```
        <Item ValueType="String" ValueRegex=""></Item>
    </Value>
</Setting>
```

用于数字和单行字符串的配置元素 。可以使用正则表达式检查有效性(可选)。这是默认的 ValueType 。

```
<Setting Name="SettingName">
    ...
    <Value>
        <Item ValueType="String" ValueRegex="" Translatable="1">Value</Item>
    </Value>
</Setting>
```

可选的 Translatable 属性将此设置标记为可翻译，这将使其包含在 OTRS 翻译文件中 。此属性可以放在任何标记上(另请参见下文)。

Password

用于密码的配置元素 。它仍然以纯文本形式存储在 XML 中，但它在 GUI 中被屏蔽 。

```
<Setting Name="SettingName">
    ...
    <Value>
        <Item ValueType="Password">Secret</Item>
    </Value>
</Setting>
```

PerlModule

用于 Perl 模块的配置元素 。它有一个 ValueFilter 属性，用于过滤可能的选择值 。在下面的示例中，用户可以选择 Perl 模块 Kernel::System::Log::SysLog 或 Kernel::System::Log::File 。

```
<Setting Name="SettingName">
    ...
    <Value>
        <Item ValueType="PerlModule" ValueFilter="Kernel/System/Log/*.pm">
↪Kernel::System::Log::SysLog</Item>
    </Value>
</Setting>
```

Textarea

用于多行文本的配置元素 。

```
<Setting Name="SettingName">
    ...
    <Value>
        <Item ValueType="Textarea"></Item>
```

(下页继续)

(续上页)

```
    </Value>
</Setting>
```

Select

此配置元素提供预设值作为下拉菜单。SelectedID 或 SelectedValue 属性可以预先选择一个默认值。

```
<Setting Name="SettingName">
    ...
    <Value>
        <Item ValueType="Select" SelectedID="Queue">
            <Item ValueType="Option" Value="Queue" Translatable="1">Queue</
↪Item>
            <Item ValueType="Option" Value="SystemAddress" Translatable="1">
↪System address</Item>
        </Item>
    </Value>
</Setting>
```

Checkbox

此配置元素复选框有两种状态：0 或 1。

```
<Setting Name="SettingName">
    ...
    <Value>
        <Item ValueType="Checkbox">0</Item>
    </Value>
</Setting>
```

Date

此配置元素包含日期值。

```
<Setting Name="SettingName">
    ...
    <Value>
        <Item ValueType="Date">2016-02-02</Item>
    </Value>
</Setting>
```

DateTime

此配置元素包含日期时间值。

```
<Setting Name="SettingName">
    ...
    <Value>
        <Item ValueType="DateTime">2016-12-08 01:02:03</Item>
    </Value>
</Setting>
```

Directory

此配置元素包含一个目录 。

```
<Setting Name="SettingName">
    ...
    <Value>
        <Item ValueType="Directory">/etc</Item>
    </Value>
</Setting>
```

File

此配置元素包含一个文件路径 。

```
<Setting Name="SettingName">
    ...
    <Value>
        <Item ValueType="File">/etc/hosts</Item>
    </Value>
</Setting>
```

Entity

此配置元素包含特定实体的值 。ValueEntityType 属性定义实体类型 。支持的实体：DynamicField 、Queue 、Priority 、State 和 Type 。一致性检查将确保只能配置有效实体，并且配置中使用的实体不能设置为无效 。此外，重命名实体时，将更新所有引用配置设置 。

```
<Setting Name="SettingName">
    ...
    <Value>
        <Item ValueType="Entity" ValueEntityType="Queue">Junk</Item>
    </Value>
</Setting>
```

TimeZone

此配置元素包含时区值 。

```
<Setting Name="SettingName">
    ...
    <Value>
        <Item ValueType="TimeZone">UTC</Item>
    </Value>
</Setting>
```

VacationDays

此配置元素包含每年重复的休假日的定义。以下属性是必填的：ValueMonth、ValueDay。

```
<Setting Name="SettingName">
    ...
    <Value>
        <Item ValueType="VacationDays">
            <DefaultItem ValueType="VacationDays"></DefaultItem>
            <Item ValueMonth="1" ValueDay="1" Translatable="1">New Year's Day
↪</Item>
            <Item ValueMonth="5" ValueDay="1" Translatable="1">International␣
↪Workers' Day</Item>
            <Item ValueMonth="12" ValueDay="24" Translatable="1">Christmas Eve
↪</Item>
        </Item>
    </Value>
</Setting>
```

VacationDaysOneTime

此配置元素包含仅发生一次的休假日的定义。以下属性是必填的：ValueMonth、ValueDay 和 ValueYear。

```
<Setting Name="SettingName">
    ...
    <Value>
        <Item ValueType="VacationDaysOneTime">
            <Item ValueYear="2004" ValueMonth="1" ValueDay="1">test</Item>
        </Item>
    </Value>
</Setting>
```

WorkingHours

此配置元素包含工作时间的定义。

```
<Setting Name="SettingName">
    ...
    <Value>
        <Item ValueType="WorkingHours">
            <Item ValueType="Day" ValueName="Mon">
```

(下页继续)

(续上页)

```
                <Item ValueType="Hour">8</Item>
                <Item ValueType="Hour">9</Item>
            </Item>
            <Item ValueType="Day" ValueName="Tue">
                <Item ValueType="Hour">8</Item>
                <Item ValueType="Hour">9</Item>
            </Item>
        </Item>
    </Value>
</Setting>
```

前端注册

服务人员界面的模块注册：

```
<Setting Name="SettiFrontend::Module###AgentModuleName">
    ...
    <Value>
        <Item ValueType="FrontendRegistration">
            <Hash>
                <Item Key="Group">
                    <Array>
                    </Array>
                </Item>
                <Item Key="GroupRo">
                    <Array>
                    </Array>
                </Item>
                <Item Key="Description" Translatable="1">Phone Call.</Item>
                <Item Key="Title" Translatable="1">Phone-Ticket</Item>
                <Item Key="NavBarName">Ticket</Item>
            </Hash>
        </Item>
    </Value>
</Setting>
```

数组和哈希中的默认项

新的 XML 结构允许我们创建复杂的结构。因此，我们需要 `DefaultItem` 条目来描述数组/哈希的结构。如果未提供，系统会认为您需要具有标量值的简单数组/散列。`DefaultItem` 在添加新元素时用作模板，因此它可以包含其它属性如 `ValueType`，并且可以定义默认值。

这里有些例子：

带有 **Select** 项的嵌套数组

```
<Array>
    <DefaultItem>
```

(下页继续)

(续上页)

```
        <Array>
            <DefaultItem ValueType="Select" SelectedID='option-2'>
                <Item ValueType="Option" Value="option-1">Option 1</Item>
                <Item ValueType="Option" Value="option-2">Option 2</Item>
            </DefaultItem>
        </Array>
    </DefaultItem>
    ...
</Array>
```

包含 **"Date"** 项的嵌套哈希

```
<Hash>
    <DefaultItem>
        <Hash>
            <DefaultItem ValueType="Date">2017-01-01</DefaultItem>
        </Hash>
    </DefaultItem>
    ...
</Hash>
```

2.1.4 在运行时访问配置选项

您可以通过核心模块 `Kernel::Config` 读取和写入(一次请求)配置选项 。

如果要读取配置选项：

```
my $ConfigOption = $Kernel::OM->Get('Kernel::Config')->Get('Prefix::Option');
```

如果要在运行时更改配置选项，只需一个请求/进程：

```
$Kernel::OM->Get('Kernel::Config')->Set(
    Key => 'Prefix::Option'
    Value => 'SomeNewValue',
);
```

2.2 数据库机制

OTRS 附带了一个支持不同数据库的数据库层 。

数据库层 `Kernel::System::DB` 有两个输入选项：*SQL* 和 *XML* 。

2.2.1 SQL

SQL 接口应该用于正常的数据库操作(`SELECT` 、`INSERT` 、`UPDATE` 等)。它可以像普通的 Perl DBI 接口一样使用 。

INSERT/UPDATE/DELETE 语句

```
$Kernel::OM->Get('Kernel::System::DB')->Do(
    SQL=> "INSERT INTO table (name, id) VALUES ('SomeName', 123)",
);

$Kernel::OM->Get('Kernel::System::DB')->Do(
    SQL=> "UPDATE table SET name = 'SomeName', id = 123",
);

$Kernel::OM->Get('Kernel::System::DB')->Do(
    SQL=> "DELETE FROM table WHERE id = 123",
);
```

SELECT 语句

```
my $SQL = "SELECT id FROM table WHERE tn = '123'";

$Kernel::OM->Get('Kernel::System::DB')->Prepare(SQL => $SQL, Limit => 15);

while (my @Row = $Kernel::OM->Get('Kernel::System::DB')->FetchrowArray()) {
    $Id = $Row[0];
}
return $Id;
```

注解：注意使用 `Limit` 作为参数而不是 SQL 字符串，因为并非所有数据库都支持 SQL 字符串中的 `LIMIT` 。

```
my $SQL = "SELECT id FROM table WHERE tn = ? AND group = ?";

$Kernel::OM->Get('Kernel::System::DB')->Prepare(
    SQL   => $SQL,
    Limit => 15,
    Bind  => [ $Tn, $Group ],
);

while (my @Row = $Kernel::OM->Get('Kernel::System::DB')->FetchrowArray()) {
    $Id = $Row[0];
}
return $Id;
```

注解：尽可能使用 `Bind` 属性，特别是对于长语句。如果使用 `Bind`，则不需要 `Quote()` 函数。

QUOTE

String（字符串）：

```
my $QuotedString = $Kernel::OM->Get('Kernel::System::DB')->Quote("It's a␣
↪problem!");
```

Integer（ 整数 ）:

```
my $QuotedInteger = $Kernel::OM->Get('Kernel::System::DB')->Quote('123',
↪'Integer');
```

Number（ 数字 ）:

```
my $QuotedNumber = $Kernel::OM->Get('Kernel::System::DB')->Quote('21.35',
↪'Number');
```

注解: 请尽可能使用 Bind 属性而不是 Quote() 。

2.2.2 XML

XML 接口应该用于 INSERT 、CREATE TABLE 、DROP TABLE 和 ALTER TABLE 。由于此语法在数据库与数据库之间不同，因此使用它可确保编写可在所有这些数据库中使用的应用程序 。

INSERT

```
<Insert Table="some_table">
    <Data Key="id">1</Data>
    <Data Key="description" Type="Quote">exploit</Data>
</Insert>
```

CREATE TABLE

可能的数据类型有：BIGINT 、SMALLINT 、INTEGER 、VARCHAR （ 长度为 1-1000000 ），DATE （ 格式：yyyy-mm-dd hh:mm:ss ）和 LONGBLOB 。

```
<TableCreate Name="calendar_event">
    <Column Name="id" Required="true" PrimaryKey="true" AutoIncrement="true"␣
↪Type="BIGINT"/>
    <Column Name="title" Required="true" Size="250" Type="VARCHAR"/>
    <Column Name="content" Required="false" Size="250" Type="VARCHAR"/>
    <Column Name="start_time" Required="true" Type="DATE"/>
    <Column Name="end_time" Required="true" Type="DATE"/>
    <Column Name="owner_id" Required="true" Type="INTEGER"/>
    <Column Name="event_status" Required="true" Size="50" Type="VARCHAR"/>
    <Index Name="calendar_event_title">
        <IndexColumn Name="title"/>
    </Index>
    <Unique Name="calendar_event_title">
        <UniqueColumn Name="title"/>
    </Unique>
    <ForeignKey ForeignTable="users">
        <Reference Local="owner_id" Foreign="id"/>
    </ForeignKey>
</TableCreate>
```

LONGBLOB 列需要特殊处理。如果数据库驱动程序不支持 DirectBlob 功能，则它们的内容需要进行 base64 转码。请参阅以下示例：

```
my $Content = $StorableContent;
if ( !$DBObject->GetDatabaseFunction('DirectBlob') ) {
    $Content = MIME::Base64::encode_base64($StorableContent);
}
```

类似地，当从这样的列读取时，通过将 Encode => 0 标志传递给 Prepare()，内容不能自动解码为 UTF-8：

```
return if !$DBObject->Prepare(
    SQL => '
        SELECT content_type, content, content_id, content_alternative,␣
↪disposition, filename
        FROM article_data_mime_attachment
        WHERE id = ?',
    Bind   => [ \$AttachmentID ],
    Encode => [ 1, 0, 0, 0, 1, 1 ],
);

while ( my @Row = $DBObject->FetchrowArray() ) {

    $Data{ContentType} = $Row[0];

    # Decode attachment if it's e. g. a postgresql backend.
    if ( !$DBObject->GetDatabaseFunction('DirectBlob') ) {
        $Data{Content} = decode_base64( $Row[1] );
    }
    else {
        $Data{Content} = $Row[1];
    }
    $Data{ContentID}          = $Row[2] || '';
    $Data{ContentAlternative} = $Row[3] || '';
    $Data{Disposition}        = $Row[4];
    $Data{Filename}           = $Row[5];
}
```

DROP TABLE

```
<TableDrop Name="calendar_event"/>
```

ALTER TABLE

以下显示了添加、更改和删除列的示例。

```
<TableAlter Name="calendar_event">
    <ColumnAdd Name="test_name" Type="varchar" Size="20" Required="true"/>

    <ColumnChange NameOld="test_name" NameNew="test_title" Type="varchar"␣
↪Size="30" Required="true"/>
```

(下页继续)

(续上页)

```
    <ColumnChange NameOld="test_title" NameNew="test_title" Type="varchar"␣
↪Size="100" Required="false"/>

    <ColumnDrop Name="test_title"/>

    <IndexCreate Name="index_test3">
        <IndexColumn Name="test3"/>
    </IndexCreate>

    <IndexDrop Name="index_test3"/>

    <UniqueCreate Name="uniq_test3">
        <UniqueColumn Name="test3"/>
    </UniqueCreate>

    <UniqueDrop Name="uniq_test3"/>
</TableAlter>
```

下一个示例显示了如何重命名一个表 。

```
<TableAlter NameOld="calendar_event" NameNew="calendar_event_new"/>
```

处理 **XML** 的代码

```
my @XMLARRAY = @{$Self->ParseXML(String => $XML)};

my @SQL = $Kernel::OM->Get('Kernel::System::DB')->SQLProcessor(
    Database => \@XMLARRAY,
);
push(@SQL, $Kernel::OM->Get('Kernel::System::DB')->SQLProcessorPost());

for (@SQL) {
    $Kernel::OM->Get('Kernel::System::DB')->Do(SQL => $_);
}
```

2.2.3 数据库驱动程序

数据库驱动程序位于 `$OTRS_HOME/Kernel/System/DB/*.pm` 下 。

2.2.4 支持的数据库

- MySQL
- PostgreSQL
- Oracle
- Microsoft SQL Server (仅用于外部数据库连接，而不是 OTRS 数据库)

2.3 日志机制

2.3.1 系统日志

OTRS 带有一个系统日志后端，可用于应用程序日志记录和调试 。

可以通过对象管理器访问和使用 Log 对象，如下所示：

```
$Kernel::OM->Get('Kernel::System::Log')->Log(
    Priority => 'error',
    Message  => 'Need something!',
);
```

根据系统配置中 MinimumLogLevel 选项配置的日志级别，记录的消息将根据其 Priority(优先级) 标志保存或不保存 。

如果设置了 error，则只记录错误 。使用 debug，您可以获得所有日志消息 。日志级别的顺序是：

- debug
- info
- notice
- error

系统日志的输出可以定向到 syslog 守护进程或日志文件，具体取决于系统配置中配置的 LogModule 选项 。

2.3.2 通信日志

除系统日志外，OTRS 还为任何与通信相关的日志记录提供专门的日志记录后端 。该系统配有专用的表和前端，用于跟踪和显示通信日志，以便于调试和状态概览 。

要利用新系统，首先要创建一个非单例的通信日志对象实例：

```
my $CommunicationLogObject = $Kernel::OM->Create(
    'Kernel::System::CommunicationLog',
    ObjectParams => {
        Transport   => 'Email',      # Transport log module
        Direction   => 'Incoming',   # Incoming|Outgoing
        AccountType => 'POP3',       # Mail account type
        AccountID   => 1,            # Mail account ID
    },
);
```

拥有通信日志对象实例时，可以启动对象日志以记录单个消息 。目前有两个对象日志：Connection 和 Message 。

Connection 对象日志应该用于记录任何与连接相关的消息(例如：在服务器上进行身份验证或检索传入消息)。

简单地说，通过声明其类型来启动对象日志，您可以立即使用它：

```
$CommunicationLogObject->ObjectLogStart(
    ObjectLogType => 'Connection',
);
```

(下页继续)

(续上页)

```
$CommunicationLogObject->ObjectLog(
    ObjectLogType => 'Connection',
    Priority      => 'Debug',                                # Trace, Debug,␣
↪Info, Notice, Warning or Error
    Key           => 'Kernel::System::MailAccount::POP3',
    Value         => "Open connection to 'host.example.com' (user-1).",
);
```

通信日志对象实例处理当前启动的对象日志，因此您无需记住并随处携带它们，但这也意味着您只能为每种类型启动一个对象 。

如果遇到不可恢复的错误，可以选择关闭对象日志并将其标记为失败：

```
$CommunicationLogObject->ObjectLog(
    ObjectLogType => 'Connection',
    Priority      => 'Error',
    Key           => 'Kernel::System::MailAccount::POP3',
    Value         => 'Something went wrong!',
);

$CommunicationLogObject->ObjectLogStop(
    ObjectLogType => 'Connection',
    Status        => 'Failed',
);
```

接下来，同样将通信日志标记为失败：

```
$CommunicationLogObject->CommunicationStop(
    Status => 'Failed',
);
```

否则，请停止对象日志，然后将通信日志视为成功：

```
$CommunicationLogObject->ObjectLog(
    ObjectLogType => 'Connection',
    Priority      => 'Debug',
    Key           => 'Kernel::System::MailAccount::POP3',
    Value         => "Connection to 'host.example.com' closed.",
);

$CommunicationLogObject->ObjectLogStop(
    ObjectLogType => 'Connection',
    Status        => 'Successful',
);

$CommunicationLogObject->CommunicationStop(
    Status => 'Successful',
);
```

`Message` 对象日志应该用于任何有关特定消息及其处理的日志条目 。它以类似的方式使用，只需确保在使用它之前启动它：

```
$CommunicationLogObject->ObjectLogStart(
    ObjectLogType => 'Message',
);

$CommunicationLogObject->ObjectLog(
    ObjectLogType => 'Message',
    Priority      => 'Error',
    Key           => 'Kernel::System::MailAccount::POP3',
    Value         => "Could not process message. Raw mail saved (report it on␣
↪http://bugs.otrs.org/)!",
);

$CommunicationLogObject->ObjectLogStop(
    ObjectLogType => 'Message',
    Status        => 'Failed',
);

$CommunicationLogObject->CommunicationStop(
    Status => 'Failed',
);
```

您还可以链接日志对象，然后查找特定对象类型和 ID 的通信：

```
$CommunicationLogObject->ObjectLookupSet(
    ObjectLogType    => 'Message',
    TargetObjectType => 'Article',
    TargetObjectID   => 2,
);

my $LookupInfo = $CommunicationLogObject->ObjectLookupGet(
    TargetObjectType => 'Article',
    TargetObjectID   => 2,
);
```

如果任何日志对象都失败，您应该确保始终停止通信并将其标记为失败。这将允许管理员在概览中查看失败的通信，并在需要时执行任何操作。

在单个进程的持续时间内保留通信日志非常重要。如果您的工作跨越多个模块，并且其中任何一个都可以从日志记录中受益，请确保传递现有的通信日志实例，以便所有方法都可以使用相同的模块。使用此方法，您将确保为同一进程生成的任何日志条目都包含在单个通信中。

如果无法传递通信日志实例（异步任务！），您还可以选择以与上一步相同的状态重新创建通信日志对象。只需获取通信 ID 并将其传递给新代码，然后使用提供的此参数创建实例：

```
# Get communication ID in parent code.
my $CommunicationID = $CommunicationLogObject->CommunicationIDGet();

# Somehow pass communication ID to child code.
# ...

# Recreate the instance in child code by using same communication ID.
my $CommunicationLogObject = $Kernel::OM->Create(
    'Kernel::System::CommunicationLog',
    ObjectParams => {
```

(下页继续)

(续上页)

```
        CommunicationID => $CommunicationID,
    },
);
```

然后，您可以继续使用此实例，如前所述，根据需要启动任何对象日志，最后添加条目和设置状态 。

如果您需要检索通信日志数据或使用它做其它事情，请同时查看 Kernel::System::CommunicationLog::DB.pm 。

2.4 日期和时间

OTRS 自带软件包来处理日期和时间，确保正确计算和存储日期和时间 。

2.4.1 介绍

日期和时间由一个 Kernel::System::DateTime 的对象表示 。每个 DateTime 对象都有自己的日期、时间和时区信息 。与现在已弃用的 Kernel::System::Time 包相比，这意味着你可以而且应该为你想要使用的每个日期/时间创建一个 DateTime 对象 。

2.4.2 创建一个 DateTime 对象

OTRS 的对象管理器已经通过 Create 方法进行了扩展，以支持可以为其创建多个实例的软件包：

```
my $DateTimeObject = $Kernel::OM->Create(
    'Kernel::System::DateTime',
    ObjectParams => {
        TimeZone => 'Europe/Berlin'
    },
);
```

上面的例子将为 *Europe/Berlin* 时区的当前日期和时间创建一个 DateTime 对象 。有更多选项来创建 DateTime 对象（ 时间组件 、字符串 、时间戳 、克隆 ），请参阅 Kernel::System::DateTime 的 POD（ Perl 在线文档 ）。

注解： 如果您尝试通过 $Kernel::OM->Get('Kernel::System::DateTime') 取回一个 DateTime 对象，将会出错 。

2.4.3 时区

以小时（ + 2，-10 等 ）为单位的时间偏移已被时区（ Europe/Berlin 、America/New_York 等 ）取代 。时区之间的转换完全封装在 DateTime 对象中 。如果要转换为其它时区，只需使用以下代码：

```
$DateTimeObject->ToTimeZone( TimeZone => 'Europe/Berlin' );
```

有一个系统配置选项 OTRSTimeZone 。此设置定义 OTRS 内部用于在数据库中存储日期和时间的时区 。

注解: 你必须确保将 DateTime 对象转换为 OTRS 时区，然后才能将它存储到数据库中(有一种方便的方法：ToOTRSTimeZone())。例外情况可能是您明确希望数据库列在特定时区中保存日期/时间 。但请注意，数据库本身在检索时不会自行提供时区信息 。

注解: Kernel::System::DateTime 的 TimeZoneList() 提供了一个可用的时区列表 。

2.4.4 方法摘要

Kernel::System::DateTime 包提供了以下方法(这只是一个选择，详见源代码)。

对象创建方法

可以通过对象管理器的 Create() 方法创建 DateTime 对象，或者用 Clone() 方法克隆另一个 DateTime 对象 。

Get 方法

使用 Get()，DateTime 对象的所有数据都将作为哈希返回(日期和时间组件包括日期名称等，以及时区)。

Set 方法

使用 Set()，您可以更改 DateTime 对象的某些组件(年 、月 、日 、小时 、分钟 、秒)，也可以根据给定的字符串设置日期和时间(*2016-05-24 23:04:12*)。请注意，您无法使用此方法更改时区 。

时区方法

要更改 DateTime 对象的时区，请使用 ToOTRSTimeZone() 方法或作为转换为 OTRS 时区 ToOTRSTimeZone() 的快捷方式 。

要检索配置的 OTRS 时区或用户默认时区，请始终使用方法 OTRSTimeZoneGet() 或 UserDefaultTimeZoneGet() 。永远不要通过 Kernel::Config 手动检索它们 。

比较操作符和方法

Kernel::System::DateTime 使用运算符重载进行比较 。所以你可以简单地将两个 DateTime 对象与 < 、<= 、== 、! = 、> = 和 > 进行比较 。Compare() 在 Perl 的排序上下文中可用，因为它返回-1 、0 或 1 。

2.4.5 弃用的包 Kernel::System::Time

现已弃用的包 Kernel::System::Time 已经扩展到完全支持时区而不是时间偏移 。这样做是为了确保现有代码在没有(更大)更改的情况下工作 。

但是，有一种情况需要更改现有代码 。如果您有使用旧时偏移量来计算新日期/时间或差异的代码，则必须迁移此代码以使用新的 DateTime 对象 。

示例(旧代码):

```
my $TimeObject      = $Kernel::OM->Get('Kernel::System::Time'); # Assume a␣
↪time offset of 0 for this time object
my $SystemTime      = $TimeObject->TimeStamp2SystemTime( String => '2004-08-14␣
↪22:45:00' );
my $UserTimeZone    = '+2'; # normally retrieved via config or param
my $UserSystemTime = $SystemTime + $UserTimeZone * 3600;
my $UserTimeStamp   = $TimeObject->SystemTime2TimeStamp( SystemTime =>
↪$UserSystemTime );
```

示例（ 新代码 ）：

```
my $DateTimeObject = $Kernel::OM->Create('Kernel::System::DateTime'); # This␣
↪implicitly sets the configured OTRS time zone
my $UserTimeZone    = 'Europe/Berlin'; # normally retrieved via config or param
$DateTimeObject->ToTimeZone( TimeZone => $UserTimeZone );
my $SystemTime     = $DateTimeObject->ToEpoch(); # note that the epoch is␣
↪independent from the time zone, it's always calculated for UTC
my $UserTimeStamp = $DateTimeObject->ToString();
```

2.5 皮肤

OTRS 的视觉外观由 皮肤控制 。

一个皮肤是一组 CSS 和图像文件，它们共同控制 GUI 呈现给用户的方式 。皮肤不会更改由 OTRS 生成的 HTML 内容（ 这是 主题所做的 ），但它们控制着它的显示方式 。在现代 CSS 标准的帮助下，可以彻底改变显示（ 例如，重新定位对话框的各部分 、隐藏元素等 ）。

2.5.1 皮肤基础

所有皮肤都在 `$OTRS_HOME/var/httpd/htdocs/skins/Agent/$SKIN_NAME` 中 。

每个服务人员可以单独选择 穿着哪一款已安装的服务人员皮肤 。

注解： 使用新的外部人员界面删除了客户界面的皮肤支持 。要为外部人员界面创建自定义布局，请使用管理员界面的 排版布局模块 。

2.5.2 如何加载皮肤

重要的是要注意，`default` 皮肤将 始终 首先加载 。如果服务人员选择了默认皮肤之外的另一个皮肤，则它将仅在默认皮肤 之后加载 。在这里的加载，我们的意思是 OTRS 会将标记放入 HTML 内容中，这会导致浏览器加载 CSS 文件 。让我们看一个这样的例子：

```
<link rel="stylesheet" href="/otrs-web/skins/Agent/default/css-cache/
↪CommonCSS_179376764084443c181048401ae0e2ad.css" />
<link rel="stylesheet" href="/otrs-web/skins/Agent/ivory/css-cache/CommonCSS_
↪e0783e0c2445ad9cc59c35d6e4629684.css" />
```

在这里可以清楚地看到，首先加载 `default` 皮肤，然后加载服务人员指定的自定义皮肤 。在这个例子中，我们看到激活的加载器的结果（ `Loader::Enabled` 设置为 1 ），它收集所有 CSS 文件，连接并缩小它们

并将它们作为浏览器的一个块提供。这样可以节省带宽并减少 HTTP 请求的数量。让我们看一下关闭加载器的相同示例：

```
<link rel="stylesheet" href="/otrs-web/skins/Agent/default/css/Core.Reset.css
↪" />
<link rel="stylesheet" href="/otrs-web/skins/Agent/default/css/Core.Default.
↪css" />
<link rel="stylesheet" href="/otrs-web/skins/Agent/default/css/Core.Header.css
↪" />
<link rel="stylesheet" href="/otrs-web/skins/Agent/default/css/Core.
↪OverviewControl.css" />
<link rel="stylesheet" href="/otrs-web/skins/Agent/default/css/Core.
↪OverviewSmall.css" />
<link rel="stylesheet" href="/otrs-web/skins/Agent/default/css/Core.
↪OverviewMedium.css" />
<link rel="stylesheet" href="/otrs-web/skins/Agent/default/css/Core.
↪OverviewLarge.css" />
<link rel="stylesheet" href="/otrs-web/skins/Agent/default/css/Core.Footer.css
↪" />
<link rel="stylesheet" href="/otrs-web/skins/Agent/default/css/Core.Grid.css"␣
↪/>
<link rel="stylesheet" href="/otrs-web/skins/Agent/default/css/Core.Form.css"␣
↪/>
<link rel="stylesheet" href="/otrs-web/skins/Agent/default/css/Core.Table.css
↪" />
<link rel="stylesheet" href="/otrs-web/skins/Agent/default/css/Core.Widget.css
↪" />
<link rel="stylesheet" href="/otrs-web/skins/Agent/default/css/Core.
↪WidgetMenu.css" />
<link rel="stylesheet" href="/otrs-web/skins/Agent/default/css/Core.
↪TicketDetail.css" />
<link rel="stylesheet" href="/otrs-web/skins/Agent/default/css/Core.Tooltip.
↪css" />
<link rel="stylesheet" href="/otrs-web/skins/Agent/default/css/Core.Dialog.css
↪" />
<link rel="stylesheet" href="/otrs-web/skins/Agent/default/css/Core.Print.css
↪" />
<link rel="stylesheet" href="/otrs-web/skins/Agent/default/css/Core.Agent.
↪CustomerUser.GoogleMaps.css" />
<link rel="stylesheet" href="/otrs-web/skins/Agent/default/css/Core.Agent.
↪CustomerUser.OpenTicket.css" />
<link rel="stylesheet" href="/otrs-web/skins/Agent/ivory/css/Core.Dialog.css"␣
↪/>
```

在这里，我们可以更好地查看来自皮肤的各个文件。

有不同类型的 CSS 文件：必须始终加载的公用文件，以及仅为 OTRS 框架内的特殊模块加载的特定于模块的文件。

此外，可以指定只能在 IE7 或 IE8 上加载的 CSS 文件（在客户界面的情况下，还有 IE6）。这是不幸的，但是在这些浏览器上开发现代 GUI 没有特殊的 CSS 是不可能的。

有关 CSS 文件类型的详细信息，请参阅*CSS 和 JavaScript 加载器* 部分。

对于每个 HTML 页面生成，加载器将首先从 `default` 皮肤中获取所有已配置的 CSS 文件，然后为每个文件查看它是否也在自定义皮肤中可用（如果选择了自定义皮肤）并在默认文件之后加载它们。

这意味着 a ）自定义皮肤中的 CSS 文件需要与默认皮肤中的名称相同，并且 b ）自定义皮肤不需要具有默认皮肤的所有文件 。这是首先加载默认皮肤的一大优势：自定义皮肤具有所有默认的 CSS 规则，只需要更改应该导致不同显示的那些 。这通常可以在单个文件中完成，如上例所示 。

这样做的另一个影响是，您需要小心覆盖要更改的自定义外观中的所有默认 CSS 规则 。我们来看一个例子：

```
.Header h1 {
    font-weight: bold;
    color: #000;
}
```

这将 .Header 元素内部的特殊标题定义为粗体黑色文本 。现在，如果你想将皮肤中的颜色改为另一种颜色和普通文本，那么写这个是不够的：

```
.Header h1 {
    color: #F00;
}
```

因为 font-weight 的原始规则仍然适用 。您需要显式覆盖它：

```
.Header h1 {
    font-weight: normal;
    color: #F00;
}
```

2.5.3 创建一个新皮肤

在本节中，我们将创建一个新的服务人员皮肤，它使用自定义公司颜色（ 浅灰色 ）替换默认 OTRS 背景颜色（ 白色 ），并使用自定义 Logo 替换默认 Logo（ 徽标 ）。此外，我们将配置该皮肤为默认情况下所有服务人员将看到的皮肤 。

为实现这一目标，我们只需要采取三个简单的步骤：

- 创建皮肤文件
- 配置新的 Logo
- 让 OTRS 系统知道该皮肤

让我们从创建新皮肤所需的文件开始 。首先，我们需要为这个皮肤创建一个新文件夹（ 我们称之为 custom ）。该文件夹将是 $OTRS_HOME/var/httpd/htdocs/skins/Agent/custom 。

在那里，我们需要将新的 CSS 文件放在一个新目录 css 中，它定义了新皮肤的外观 。我们称之为 Core.Default.css 。请记住，它必须与默认外观中的一个文件具有相同的名称 。这是 CSS 文件所需的代码：

```
body {
    background-color: #c0c0c0; /* not very beautiful but it meets our purpose␣
↪*/
}
```

接下来是第二步，添加一个新 Logo 并使 OTRS 系统知道该新皮肤 。为此，我们首先需要将我们的自定义 logo（ 例如 logo.png ）放在我们的皮肤目录中的新目录（ 例如 img ）中 。然后我们需要创建一个新的配置文件 $OTRS_HOME/Kernel/Config/Files/XML/CustomSkin.xml，它将包含所需的设置，如下所示：

```
<?xml version="1.0" encoding="utf-8" ?>
<otrs_config version="1.0" init="Changes">
```

(下页继续)

(续上页)

```
    <ConfigItem Name="AgentLogo" Required="0" Valid="1">
        <Description Translatable="1">The logo shown in the header of the␣
↪agent interface. The URL to the image must be a relative URL to the skin␣
↪image directory.</Description>
        <Group>Framework</Group>
        <SubGroup>Frontend::Agent</SubGroup>
        <Setting>
            <Hash>
                <Item Key="URL">skins/Agent/custom/img/logo.png</Item>
                <Item Key="StyleTop">13px</Item>
                <Item Key="StyleRight">75px</Item>
                <Item Key="StyleHeight">67px</Item>
                <Item Key="StyleWidth">244px</Item>
            </Hash>
        </Setting>
    </ConfigItem>
    <ConfigItem Name="Loader::Agent::Skin###100-custom" Required="0" Valid="1">
        <Description Translatable="1">Custom skin for the development manual.
↪</Description>
        <Group>Framework</Group>
        <SubGroup>Frontend::Agent</SubGroup>
        <Setting>
            <Hash>
                <Item Key="InternalName">custom</Item>
                <Item Key="VisibleName">Custom</Item>
                <Item Key="Description">Custom skin for the development␣
↪manual.</Item>
                <Item Key="HomePage">www.yourcompany.com</Item>
            </Hash>
        </Setting>
    </ConfigItem>
</otrs_config>
```

要使此配置处于活动状态，我们需要导航到 OTRS 系统管理区域中的系统配置模块 。或者，您可以运行脚本：

```
$OTRS_HOME/bin/otrs.Console.pl Maint::Config::Rebuild
```

这将重新生成 XML 配置文件的 Perl 缓存，以便我们的新皮肤为系统所知并可在系统中选择。要使它成为服务人员在进行自己的皮肤选择之前看到的默认皮肤，请编辑系统配置设置 `Loader::Agent::DefaultSelectedSkin` 并将其设置为 *custom* 。

总之：要在 OTRS 中创建一个新的皮肤，我们必须放置新的徽标文件，并创建一个 CSS 和一个 XML 文件，从而产生三个新文件：

```
$OTRS_HOME/Kernel/Config/Files/XML/CustomSkin.xml
$OTRS_HOME/var/httpd/htdocs/skins/Agent/custom/img/custom-logo.png
$OTRS_HOME/var/httpd/htdocs/skins/Agent/custom/css/Core.Header.css
```

2.6 CSS 和 JavaScript 加载器

OTRS 中的 CSS 和 JavaScript 代码增长很多。必须解决能够满足开发问题（大量单独文件的良好可维护性）和性能问题（少量 HTTP 请求和提供缩小内容而没有不必要的空白和文档）的问题。为了实现这些目标，发明了装载机。

2.6.1 它如何运作

简单来说，加载器：

- 为每个请求确定当前应用程序模块在客户端需要哪些 CSS 和 JavaScript 文件
- 收集所有相关数据
- 删除不必要的空格和文档来缩小数据
- 仅在少数 HTTP 请求中为客户端提供服务而不是许多单独的 HTTP 请求，允许客户端将这些片段缓存在浏览器缓存中
- 利用 OTRS 的缓存机制，以高性能的方式执行这些任务

当然，还有一些更详细的内容，但这应该足以作为首次概览。

2.6.2 基本操作

使用配置设置 `Loader::Enabled::CSS` 和 `Loader::Enabled::JavaScript`，可以分别为 CSS 和 JavaScript 打开和关闭加载器（默认情况下它处于启用状态）。

> **警告：** 由于 Internet Explorer 中的渲染问题，无法为此客户端浏览器的 CSS 文件关闭加载器（将覆盖配置设置）。Internet Explorer 8 之前均无法处理超过 32 个 CSS 文件的页面。

要了解加载器的工作原理，请使用上述配置设置在 OTRS 安装中将其关闭。现在查看您当前在此 OTRS 系统中使用的应用程序模块的源代码（当然，在重新加载之后）。您将看到页面的 `<head>` 部分中加载了许多 CSS 文件，页面底部有许多 JavaScript 文件，就在闭合 `</body>` 元素之前。

在具有可读格式的许多单个文件中具有这样的内容使得开发更容易，甚至可能。但是，这具有大量 HTTP 请求（网络延迟具有很大影响）和需要传输到客户端的不必要内容（空白和说明文档）的缺点。

加载器通过执行上面简短描述中概述的步骤来解决此问题。请再次打开 Loader 并立即重新加载您的页面。现在您可以看到 HTML 代码中只有非常少的 CSS 和 JavaScript 标记，如下所示：

```
<script type="text/javascript" src="/otrs30-dev-web/js/js-cache/CommonJS_
↪d16010491cbd4faaaeb740136a8ecbfd.js"></script>

<script type="text/javascript" src="/otrs30-dev-web/js/js-cache/ModuleJS_
↪b54ba9c085577ac48745f6849978907c.js"></script>
```

刚刚发生了什么？在为该页面生成 HTML 代码的原始请求期间，加载器生成了这两个文件（或从缓存中获取它们）并在页面上显示链接到这些文件的 `<script>` 标签，而不是到所有相关的 JavaScript 文件的链接（正如您关闭加载器状态时看到的那样）。

CSS 部分看起来有点复杂：

```
    <link rel="stylesheet" type="text/css" href="/otrs30-dev-web/skins/Agent/
↪default/css-cache/CommonCSS_00753c78c9be7a634c70e914486bfbad.css" />

<!--[if IE 7]>
    <link rel="stylesheet" type="text/css" href="/otrs30-dev-web/skins/Agent/
↪default/css-cache/CommonCSS_IE7_59394a0516ce2e7359c255a06835d31f.css" />
<![endif]-->

<!--[if IE 8]>
    <link rel="stylesheet" type="text/css" href="/otrs30-dev-web/skins/Agent/
↪default/css-cache/CommonCSS_IE8_ff58bd010ef0169703062b6001b13ca9.css" />
<![endif]-->
```

原因是 Internet Explorer 7 和 8 除了缺省 CSS 之外还需要特殊处理，因为它们缺乏对 Web 标准技术的支持。所以我们在所有浏览器中加载了一些普通的 CSS，以及一些特殊的 CSS，这些 CSS 在所谓的 条件注释中，这使得它 只能由 Internet Explorer 7/8 加载。所有其它浏览器都会忽略它。

现在我们已经概述了加载器的工作原理。让我们看一下如何通过向加载器添加配置数据，告诉它加载额外的或替代的 CSS 或 JavaScript 内容，在自己的 OTRS 扩展中利用它。

2.6.3 配置加载器：JavaScript

为了能够正确操作，加载器需要知道必须为特定 OTRS 应用程序模块加载哪些内容。首先，它将查找 始终必须加载的 JavaScript 文件，然后查找仅与当前应用程序模块相关的特殊文件。

共用 Javascript

要加载的 JavaScript 文件列表在配置设置 `Loader::Agent::CommonJS` (用于服务人员界面）和 `Loader::Customer::CommonJS`（ 用于客户界面)中配置。

这些设置被设计为哈希值，因此 OTRS 扩展可以为要加载的其它内容添加自己的哈希键。我们来看一个例子：

```
<Setting Name="Loader::Agent::CommonJS###000-Framework" Required="1" Valid="1
↪">
    <Description Translatable="1">List of JS files to always be loaded for the␣
↪agent interface.</Description>
    <Navigation>Frontend::Base::Loader</Navigation>
    <Value>
        <Array>
            <Item>thirdparty/jquery-3.2.1/jquery.js</Item>
            <Item>thirdparty/jquery-browser-detection/jquery-browser-
↪detection.js</Item>

            ...

            <Item>Core.Agent.Header.js</Item>
            <Item>Core.UI.Notification.js</Item>
            <Item>Core.Agent.Responsive.js</Item>
        </Array>
    </Value>
</Setting>
```

这是始终需要为 OTRS 的服务人员界面加载的 JavaScript 文件列表 。

要添加应该始终在服务人员界面中加载的新内容，只需添加带有另一个哈希条目的 XML 配置文件：

```
<Setting Name="Loader::Agent::CommonJS###000-Framework" Required="1" Valid="1
↪">
    <Description Translatable="1">List of JS files to always be loaded for the␣
↪agent interface.</Description>
    <Navigation>Frontend::Base::Loader</Navigation>
    <Value>
        <Array>
            <Item>thirdparty/jquery-3.2.1/jquery.js</Item>
        </Array>
    </Value>
</Setting>
```

很简单，不是吗？

特定于模块的 JavaScript

并非所有 JavaScript 都可用于 OTRS 的所有应用程序模块 。因此，可以指定特定于模块的 JavaScript 文件 。每当使用某个模块（ 例如 `AgentDashboard` ）时，也会加载该模块的模块特定 JavaScript 。配置在 XML 配置中的前端模块注册中完成 。再举一个例子：

```
<Setting Name="Loader::Module::AgentDashboard###001-Framework" Required="0"␣
↪Valid="1">
    <Description Translatable="1">Loader module registration for the agent␣
↪interface.</Description>
    <Navigation>Frontend::Agent::ModuleRegistration::Loader</Navigation>
    <Value>
        <Hash>
            <Item Key="CSS">
                <Array>
                    <Item>Core.Agent.Dashboard.css</Item>

                    ...

                </Array>
            </Item>
            <Item Key="JavaScript">
                <Array>
                    <Item>thirdparty/momentjs-2.18.1/moment.min.js</Item>
                    <Item>thirdparty/fullcalendar-3.4.0/fullcalendar.min.js</
↪Item>
                    <Item>thirdparty/d3-3.5.6/d3.min.js</Item>
                    <Item>thirdparty/nvd3-1.7.1/nvd3.min.js</Item>
                    <Item>thirdparty/nvd3-1.7.1/models/OTRSLineChart.js</Item>
                    <Item>thirdparty/nvd3-1.7.1/models/OTRSMultiBarChart.js</
↪Item>
                    <Item>thirdparty/nvd3-1.7.1/models/OTRSStackedAreaChart.js
↪</Item>
                    <Item>thirdparty/canvg-1.4/rgbcolor.js</Item>
                </Array>
```

(下页继续)

(续上页)

```
            </Item>
        </Hash>
    </Value>
</Setting>
```

可以在前端模块注册中放置一个 <Item Key="JavaScript"> 标签，包含 <Array> 和为此应用程序模块加载的每个 JavaScript 文件一个 <Item> 标签 。

现在，您拥有配置加载器处理 JavaScript 代码的方式所需的所有信息 。

2.6.4 配置加载器：CSS

加载器处理与 JavaScript 文件非常相似的 CSS 文件，如上一节所述，扩展设置也以相同的方式工作 。

共用 CSS

处理共用 CSS 的加载方式非常类似于共用 *Javascript* 。

2.7 模板机制

OTRS 在内部使用模板机制动态生成其 HTML 页面（ 和其它内容 ），同时保持程序逻辑（ Perl ）和呈现（ HTML ）分离 。通常，前端模块将使用自己的模板文件，将一些数据传递给它并将渲染的结果返回给用户 。

模板文件位于 $OTRS_HOME/Kernel/Output/HTML/Standard/*.tt 。

OTRS 依赖于 Template::Toolkit 渲染引擎 。可以在 OTRS 模板中使用完整的 Template::Toolkit 语法 。本节介绍一些示例用例和 Template::Toolkit 语法的 OTRS 扩展 。

2.7.1 模板命令

插入动态数据

在模板中，必须插入动态数据 、引用等 。本节列出了执行此操作的相关命令 。

```
[% Data.Name %]
```

如果应用程序模块将数据参数提供给模板，则这些数据可以输出到模板 。[%data.name%] 是最简单但也是最危险的 。它将把名为 Name 的数据参数按原样插入到模板中，而无需进一步处理 。

> 警告：由于缺少 HTML 引用，这可能导致安全问题 。永远不要输出用户输入的数据而不在 HTML 上下文中引用 。例如，用户可以只插入一个 <script> 标签，它将在 OTRS 生成的 HTML 页面上输出 。
>
> 尽可能使用 [%Data.Name | html%]（ 在 HTML 中 ）或 [%Data.Name | uri%]（ 在链接中 ）。

示例：每当我们在应用程序中生成 HTML 时，我们需要将其输出到模板而不使用 HTML 引用，例如 <select> 元素，这些元素由 OTRS 中的 Layout::BuildSelection() 函数生成 。

```
<label for="Dropdown">Example Dropdown</label>
[% Data.DropdownString"]
```

如果您的数据条目包含包含特殊字符的复杂名称，则不能使用点(.)表示法来访问此数据。使用 item() 代替：[% Data.item('Complex-name') %]。

[% Data.Name | html %]

此命令与前一个命令具有相同的功能，但它在插入模板时对数据执行 HTML 引用。

```
The name of the author is [% Data.Name | html %].
```

也可以指定值的最大长度。例如，如果您只想显示变量的 8 个字符(结果将是 *SomeName[…]*)，请使用以下命令：

```
The first 20 characters of the author's name: [% Data.Name | truncate(20) |␣
↪html %].
```

[% Data.Name | uri %]

此命令在插入模板时对数据执行 URL 编码。这应该用于输出单个参数名称或 URL 值，以防止出现安全问题。它不能用于完整的 URL，因为它也会掩盖 = 。

```
<a href="[% Env("Baselink") %];Location=[% Data.File | uri %]">[% Data.File |␣
↪truncate(110) | html %]</a>
```

[% Data.Name | JSON %]

此命令将字符串或其它数据结构输出为 JavaScript JSON 字符串。

```
var Text = [% Data.Text | JSON %];
```

请注意，过滤符号仅适用于简单字符串。要将复杂数据输出为 JSON，请将其用作函数：

```
var TreeData = [% JSON(Data.TreeData) %];
```

[% Env() %]

插入由 LayoutObject 提供的环境变量。一些例子：

```
The current user name is: [% Env("UserFullname") %]
```

其它一些常见的预定义变量是：

- [% Env("Action") %]: 当前操作
- [% Env("Baselink") %]: 基链接，如 index.pl?SessionID=...
- [% Env("CGIHandle") %]: 当前 CGI 句柄，如 index.pl
- [% Env("SessionID") %]: 当前会话 ID

- `[% Env("Time") %]`: 当前时间，如 *Thu Dec 27 16:00:55 2001*
- `[% Env("UserFullname") %]`: 用户全名，如 Dirk Seiffert
- `[% Env("UserIsGroup[admin]") %]`: 用户是 admin 组成员
- `[% Env("UserIsGroup[users]") %]`: 用户是 users 组成员（ 对自己的链接有用 ）
- `[% Env("UserLogin") %]`: 用户登录名，如 mgg@x11.org

> 警告: 由于缺少 HTML 引用，这可能导致安全问题 。永远不要输出用户输入的数据而不在 HTML 上下文中引用 。例如，用户可以只插入一个 `<script>` 标签，它将在 OTRS 生成的 HTML 页面上输出 。
>
> 别忘了添加适当的 `| html` 过滤器 。

`[% Config() %]`

将配置变量插入模板 。让我们看一个示例 `Kernel/Config.pm`:

```
[Kernel/Config.pm]
    # FQDN
    # (Full qualified domain name of your system.)
    $Self->{FQDN} = 'otrs.example.com';
    # AdminEmail
    # (Email of the system admin.)
    $Self->{AdminEmail} = 'admin@example.com';
[...]
```

要在模板中输出它的值，请使用:

```
The hostname is '$Config{"FQDN"}'
The admin email address is '[% Config("AdminEmail") %]'
```

> 警告: 由于缺少 HTML 引用，这可能导致安全问题 。
>
> 别忘了添加适当的 `| html` 过滤器 。

本地化命令

`[% Translate() %]`

将字符串转换为当前用户的选定语言 。如果未找到翻译，将使用原始字符串 。

```
Translate this text: [% Translate("Help") | html %]
```

您还可以使用 `Translate` 作为过滤器来翻译动态数据:

```
Translate data from the application: [% Data.Type | Translate | html %]
```

您还可以在字符串内指定一个或多个参数(`%s`)，这些参数应替换为动态数据:

```
Translate this text and insert the given data: [% Translate("Change %s␣
↪settings", Data.Type) | html %]
```

JavaScript 中的字符串可以使用 JSON 过滤器进行翻译和处理 。

```
var Text = [% Translate("Change %s settings", Data.Type) | JSON %];
```

[% Localize() %]

根据当前语言/区域设置输出数据 。

在不同的文化区域中，使用不同的日期和时间格式约定 。例如，在德国是 01.02.2010，而在美国是 02/01/2010 。`[% Localize() %]` 从模板中抽象出来 。我们来看一个例子：

```
[% Data.CreateTime  Localize("TimeLong") %]
# Result for US English locale:
06/09/2010 15:45:41
```

首先，使用 `Data` 从应用程序模块插入数据 。这里始终必须将 ISO UTC 时间戳（ 2010-06-09 15:45:41 ）作为数据传递给 `[% Localize() %]` 。然后根据当前语言环境的日期/时间定义输出它 。

传递给 `[% Localize() %]` 的数据必须是 UTC 。如果为当前服务人员指定了时区偏移量，则会在生成输出之前将其应用于 UTC 时间戳 。

有不同的日期/时间输出格式：`TimeLong`（ 完整日期/时间 ）、`TimeShort`（ 没有秒 ）和 “Date“（ 没有时间 ）。

```
[% Data.CreateTime  Localize("TimeLong") %]
# Result for US English locale:
06/09/2010 15:45:41

[% Data.CreateTime  Localize("TimeShort") %]
# Result for US English locale:
06/09/2010 15:45

[% Data.CreateTime  Localize("Date") %]
# Result for US English locale:
06/09/2010
```

此外，人类可读文件大小的输出可用 “Localize(‘Filesize’)“（ 只传递原始文件大小，以字节为单位 ）选项设置 。

```
[% Data.Filesize  Localize("Filesize") %]
# Result for US English locale:
23 MB
```

[% ReplacePlaceholders() %]

用传递的参数替换字符串中的占位符（ `%s` ）。

在某些情况下，您可能希望在翻译中插入 HTML 代码，而不是占位符 。另一方面，您还需要处理清理，因为翻译的字符串不应该被原样信任 。为此，您可以先转换字符串，将其传递给 HTML 过滤器，最后用静态（ 安全 ）HTML 代码替换占位符 。

```
[% Translate("This is %s.") | html | ReplacePlaceholders('<strong>bold text</
↪strong>') %]
```

ReplacePlaceholders() 过滤器的参数数量应与原始字符串中的占位符数相匹配 。

您也可以在函数格式中使用 [%ReplacePlaceholders()%]，以防您没有翻译任何内容 。在这种情况下，第一个参数是目标字符串，其中任何找到的占位符都替换为后续参数 。

```
[% ReplacePlaceholders("This string has both %s and %s.", '<strong>bold text</
↪strong>, '<em>italic text</em>') %]
```

模板处理命令

注释

以 # 开头的行不会显示在 html 输出中 。这既可用于注释模板代码，也可用于禁用部分模板代码 。

```
# this section is temporarily disabled
# <div class="AsBlock">
#     <a href="...">link</a>
# </div>
```

[% InsertTemplate("Copyright.tt") %]

警告: 请注意，添加了 InsertTemplate 命令以提供与旧 DTL 系统更好的向后兼容性 。它可能会在未来的 OTRS 版本中弃用，并在以后删除 。如果你不在包含的模板中使用块命令，则不需要 InsertTemplate 并且可以使用标准的 Template::Toolkit 语法，例如 INCLUDE/PROCESS 。

将另一个模板文件包含到当前模板中 。包含的文件也可能包含模板命令 。

```
# include Copyright.tt
[% InsertTemplate("Copyright") %]
```

请注意，这与 Template::Toolkit 的 [% INCLUDE %] 命令不同，后者只处理引用的模板 。[% InsertTemplate() %] 实际上将引用模板的内容添加到当前模板中，以便可以一起处理它 。这使得嵌入式模板可以访问与主模板相同的环境/数据 。

[% RenderBlockStart %] / [% RenderBlockEnd %]

警告: 请注意，添加了块命令以提供与旧 DTL 系统更好的向后兼容性 。它们可能在将来的 OTRS 版本中被弃用，稍后会被删除 。我们建议您在不使用 blocks 命令的情况下开发任何新代码 。您可以使用标准的 Template::Toolkit 语法，如 IF/ELSE 、LOOP 以及条件模板输出的其它有用的东西 。

使用此命令，可以将模板文件的某些部分指定为块 。需要使用来自应用程序的函数调用显式填充此块，以使其出现在生成的输出中 。应用程序可以调用块 0 次(它不会出现在输出中)、1 次或多次(每次都可能有一组不同的数据参数传递给模板)。

一个常见的用例是使用动态数据填充表：

```
<table class="DataTable">
    <thead>
        <tr>
            <th>[% Translate("Name") | html %]</th>
            <th>[% Translate("Type") | html %]</th>
            <th>[% Translate("Comment") | html %]</th>
            <th>[% Translate("Validity") | html %]</th>
            <th>[% Translate("Changed") | html %]</th>
            <th>[% Translate("Created") | html %]</th>
        </tr>
    </thead>
    <tbody>
[% RenderBlockStart("NoDataFoundMsg") %]
        <tr>
            <td colspan="6">
                [% Translate("No data found.") | html %]
            </td>
        </tr>
[% RenderBlockEnd("NoDataFoundMsg") %]
[% RenderBlockStart("OverviewResultRow") %]
        <tr>
            <td><a class="AsBlock" href="[% Env("Baselink") %]Action=[% Env(
↪"Action") %];Subaction=Change;ID=[% Data.ID | uri %]">[% Data.Name | html %]
↪</a></td>
            <td>[% Translate(Data.TypeName) | html %]</td>
            <td title="[% Data.Comment | html %]">[% Data.Comment |␣
↪truncate(20) | html %]</td>
            <td>[% Translate(Data.Valid) | html %]</td>
            <td>[% Data.ChangeTime | Localize("TimeShort") %]</td>
            <td>[% Data.CreateTime | Localize("TimeShort") %]</td>
        </tr>
[% RenderBlockEnd("OverviewResultRow") %]
    </tbody>
</table>
```

始终生成带有标题的完整表。如果没有找到数据，则会调用块 `NoDataFoundMsg` 一次，从而产生一个包含一个数据行的表，显示消息 没有找到数据 。

如果找到数据，则对于每一行，都会对 `OverViewResultRow` 块进行一次函数调用(每次传入此特定行的数据)，从而得到一个表，包含与找到结果一样多的数据行 。

让我们看看如何从应用程序模块调用块：

```
my %List = $Kernel::OM->Get('Kernel::System::State)->StateList(
    UserID => 1,
    Valid  => 0,
);

# if there are any states, they are shown
if (%List) {

    # get valid list
```

(下页继续)

(续上页)

```
    my %ValidList = $Kernel::OM->Get('Kernel::System::Valid')->ValidList();
    for my $ListKey ( sort { $List{$a} cmp $List{$b} } keys %List ) {

        my %Data = $Kernel::OM->Get('Kernel::System::State)->StateGet( ID =>
↪$ListKey );
        $Kernel::OM->Get('Kernel::Output::HTML::Layout')->Block(
            Name => 'OverviewResultRow',
            Data => {
                Valid => $ValidList{ $Data{ValidID} },
                %Data,
            },
        );
    }
}

# otherwise a no data found msg is displayed
else {
    $Kernel::OM->Get('Kernel::Output::HTML::Layout')->Block(
        Name => 'NoDataFoundMsg',
        Data => {},
    );
}
```

注意块如何将它们的名称和一组可选数据作为单独的参数传递给块函数调用。块内的数据插入命令总是需要提供给该块的块函数调用的数据，而不是一般的模板渲染调用。

有关更多信息，请访问 文档门户网站。

[% WRAPPER JSOnDocumentComplete %]...[% END %]

标记在加载所有 CSS、JavaScript 和其它外部内容并完成基本 JavaScript 初始化之后应执行的 JavaScript 代码。让我们再看一个例子：

```
<form action="[% Env("CGIHandle") %]" method="post" enctype="multipart/form-
↪data" name="MoveTicketToQueue" class="Validate PreventMultipleSubmits" id=
↪"MoveTicketToQueue">
    <input type="hidden" name="Action"       value="[% Env("Action") %]"/>
    <input type="hidden" name="Subaction"    value="MoveTicket"/>

    ...

    <div class="Content">
        <fieldset class="TableLike FixedLabel">
            <label class="Mandatory" for="DestQueueID"><span class="Marker">*
↪</span> [% Translate("New Queue") | html %]:</label>
            <div class="Field">
                [% Data.MoveQueuesStrg %]
                <div id="DestQueueIDError" class="TooltipErrorMessage" ><p>[%␣
↪Translate("This field is required.") | html %]</p></div>
                <div id="DestQueueIDServerError" class="TooltipErrorMessage">
↪<p>[% Translate("This field is required.") | html %]</p></div>
[% WRAPPER JSOnDocumentComplete %]
```

(下页继续)

(续上页)

```
<script type="text/javascript">
    $('#DestQueueID').bind('change', function (Event) {
        $('#NoSubmit').val('1');
        Core.AJAX.FormUpdate($('#MoveTicketToQueue'), 'AJAXUpdate',
↪'DestQueueID', ['NewUserID', 'OldUserID', 'NewStateID', 'NewPriorityID' [%␣
↪Data.DynamicFieldNamesStrg %]]);
    });
</script>
[% END %]
                    </div>
                    <div class="Clear"></div>
```

这个片段创建一个小型表单，并在 `<select>` 元素上放置一个 `onchange` 处理程序，触发基于 AJAX 的表单更新 。

为什么有必要将 JavaScript 代码包含在 `[% WRAPPER JSOnDocumentComplete %]...[% END %]` 中？出于性能原因，JavaScript 加载已移至页面的页脚部分 。这意味着在页面的 `<body>` 中，还没有加载任何 JavaScript 库 。使用 `[% WRAPPER JSOnDocumentComplete %]...[% END %]`，您可以确保将此 JavaScript 移动到最终 HTML 文档的一部分，只有在整个外部 JavaScript 和 CSS 内容已成功加载和初始化之后才能执行它 。

在 `[% WRAPPER JSOnDocumentComplete %]...[% END %]` 块中，您可以使用 `<script>` 标签来封装您的 JavaScript 代码，但您不必这样做 。这可能是有益的，因为它将在支持它的 IDE 中启用正确的语法突出显示 。

2.7.2 使用一个模板文件

好了，那如何实际处理模板文件并生成结果呢？这很简单：

```
# render AdminState.tt
$Output .= $Kernel::OM->Get('Kernel::Output::HTML::Layout')->Output(
    TemplateFile => 'AdminState',
    Data         => \%Param,
);
```

在前端模块中，调用 `Kernel::Output::HTML::Layout` 的 `Output()` 函数（ 在该模板中调用了所有需要的块之后 ），以生成最终输出 。对于不在块内的所有数据插入命令，将向模板传递一组可选的数据参数 。

2.8 创建你自己的主题

您可以创建自己的主题，以便使用您喜欢的 OTRS Web 前端布局 。要创建自定义主题，应根据需要自定义输出模板 。有关输出模板语法和结构的详细信息，请参见模板机制 。

例如，执行以下步骤创建名为 *Company* 的新主题：

1. 创建一个名为 `Kernel/Output/HTML/Templates/Company` 的目录，并将您想要更改的所有文件从 `Kernel/Output/HTML/Templates/Standard` 复制到新文件夹中 。

   注解： 仅复制您计划更改的文件 。OTRS 将自动从标准主题中获取缺失的文件 。这将使后期升级变得更加容易 。

2. 定制 Kernel/Output/HTML/Templates/Company 目录中的文件，并根据需要更改布局 。
3. 要激活新主题，请在系统配置 Frontend::Themes 中添加它们 。

现在应该可用新主题了 。您可以通过个人首选项选择它 。

> 警告：不要直接修改 OTRS 自带的主题文件，因为这些更改在 OTRS 升级后会丢失 。只需按上述步骤创建你自己的主题就行了 。

2.9 本地化/翻译机制

翻译/本地化软件需要四个步骤：在源文件中标记可本地化的字符串 、生成翻译数据库/文件 、翻译过程本身以及在代码中使用翻译的数据 。

2.9.1 在源文件中标记可翻译字符串

在 Perl 代码中，所有要翻译的文字字符串都会自动标记为翻译：

```
$LanguageObject->Translate('My string %s', $Data)
```

这将标记为 *My string %s* 进行翻译 。如果你需要标记字符串，但尚未在代码中翻译它们，你可以使用 NOOP 方法 Kernel::Langauge::Translatable() 。

```
package MyPackage;

use strict;
use warnings;

use Kernel::Language (qw(Translatable));

...

my $UntranslatedString = Translatable('My string %s');
```

在模板文件中，所有包含在 Translate() 标签中的文字字符串都会自动标记以便提取：[% Translate('My string %s', Data.Data)%] 。

在系统配置和数据库 XML 文件中，您可以使用 Translatable="1" 属性标记要提取的字符串 。

```
# Database XML
<Insert Table="groups">
    <Data Key="id" Type="AutoIncrement">1</Data>
    ...
    <Data Key="comments" Type="Quote" Translatable="1">Group for default␣
↪access.</Data>
    ...
</Insert>

# SysConfig XML
<Setting>
    <Option SelectedID="0">
```

(下页继续)

(续上页)

```
        <Item Key="0" Translatable="1">No</Item>
        <Item Key="1" Translatable="1">Yes</Item>
    </Option>
</Setting>
```

2.9.2 将可翻译字符串收集到翻译数据库中

控制台命令 otrs.Console.pl Dev::Tools::TranslationsUpdate 用于从源文件中提取所有可翻译的字符串。这些将被收集并写入翻译文件。

对于 OTRS 框架和所有扩展模块，都会编写 .pot 和 .po 文件。这些文件由翻译人员用于本地化软件。

但出于速度原因，OTRS 要求翻译出现在 Perl 文件中。这些文件也将由 otrs.Console.pl Dev::Tools::TranslationsUpdate 生成。有两种不同的转换缓存文件类型，按以下顺序使用。如果在翻译文件中重新定义了单词/句子，则将使用最后一个定义的。

1. 默认的框架翻译文件：Kernel/Language/$Language.pm
2. 自定义的翻译文件：Kernel/Language/$Language_Custom.pm

默认的框架翻译文件

默认的框架翻译文件包括基本的翻译。下面是一个默认的框架翻译文件的一个例子。

```
package Kernel::Language::de;

use strict;
use warnings;

use vars qw(@ISA $VERSION);

sub Data {
    my $Self = shift;

    # $$START$$

    # possible charsets
    $Self->{Charset} = ['iso-8859-1', 'iso-8859-15', ];
    # date formats (%A=WeekDay;%B=LongMonth;%T=Time;%D=Day;%M=Month;%Y=Jear;)
    $Self->{DateFormat} = '%D.%M.%Y %T';
    $Self->{DateFormatLong} = '%A %D %B %T %Y';
    $Self->{DateFormatShort} = '%D.%M.%Y';
    $Self->{DateInputFormat} = '%D.%M.%Y';
    $Self->{DateInputFormatLong} = '%D.%M.%Y - %T';

    $Self->{Translation} = {
    # Template: AAABase
    'Yes' => 'Ja',
    'No' => 'Nein',
    'yes' => 'ja',
    'no' => 'kein',
    'Off' => 'Aus',
```

(下页继续)

(续上页)

```
    'off' => 'aus',
    };
    # $$STOP$$
    return 1;
}

1;
```

自定义的翻译文件

自定义翻译文件最后读出并使用其翻译。如果要在安装中添加自己的措辞，请为您的语言创建此文件。

```
package Kernel::Language::xx_Custom;

use strict;
use warnings;

use vars qw(@ISA $VERSION);

sub Data {
    my $Self = shift;

    # $$START$$

    # own translations
    $Self->{Translation}->{'Lock'} = 'Lala';
    $Self->{Translation}->{'Unlock'} = 'Lulu';

    # $$STOP$$
    return 1;
}

1;
```

注解：新接口的语言文件现在是构建应用程序（静态 JSON）的一部分。将自定义语言文件添加到文件系统时，需要重新生成应用程序以使更改生效。要触发重建，请使用 `--deploy-assets` 选项重新启动服务器：

```
otrs> /opt/otrs/bin/otrs.WebServer.pl --deploy-assets
```

在构建过程中，语言文件将被刷新，任何 `*_Custom.pm` 都会被处理。

2.9.3 翻译过程本身

OTRS 使用 Weblate 来管理翻译过程。有关详细信息，请参阅翻译 部分。

2.9.4 使用代码中的翻译数据

您可以使用 `$LanguageObject->Translate()` 方法在运行时从 Perl 代码翻译字符串，在模板机制 中使用 `Translate()` 标记 。

CHAPTER 3

如何扩展 OTRS

3.1 编写新的 OTRS 前端模块

在本节中，基于一个简单的小程序来说明新 OTRS 模块的编写 。必要的先决条件是在开发环境章节中指定的 OTRS 开发环境 。

3.1.1 要编写的内容

我们想要编写一个小的 OTRS 模块，在调用时显示文本“Hello World”。首先，我们必须为开发人员目录中的模块构建 `/Hello World` 目录 。在此目录中，可以创建 OTRS 中存在的所有目录 。每个模块至少应包含以下目录：

```
Kernel
Kernel/System
Kernel/Modules
Kernel/Output/HTML/Templates/Standard
Kernel/Config
Kernel/Config/Files
Kernel/Config/Files/XML/
Kernel/Language
```

3.1.2 默认配置文件

创建模块注册有助于在 OTRS 中显示新模块 。因此我们创建一个文件 `/Kernel/Config/Files/XML/HelloWorld.xml` 。在这个文件中，我们创建了一个新的配置元素 。各种设置的影响在配置机制 章节中描述 。

```
<?xml version="1.0" encoding="UTF-8" ?>
<otrs_config version="2.0" init="Application">
```

(下页继续)

(续上页)

```
    <Setting Name="Frontend::Module###AgentHelloWorld" Required="1" Valid="1">
        <Description Translatable="1">FrontendModuleRegistration for␣
↪HelloWorld module.</Description>
        <Navigation>Frontend::Agent::ModuleRegistration</Navigation>
        <Value>
            <Item ValueType="FrontendRegistration">
                <Hash>
                    <Item Key="Group">
                        <Array>
                            <Item>users</Item>
                        </Array>
                    </Item>
                    <Item Key="GroupRo">
                        <Array>
                        </Array>
                    </Item>
                    <Item Key="Description" Translatable="1">HelloWorld.</
↪Item>
                    <Item Key="Title" Translatable="1">HelloWorld</Item>
                    <Item Key="NavBarName">HelloWorld</Item>
                </Hash>
            </Item>
        </Value>
    </Setting>
    <Setting Name="Loader::Module::AgentHelloWorld###002-Filename" Required="0
↪" Valid="1">
        <Description Translatable="1">Loader module registration for the␣
↪agent interface.</Description>
        <Navigation>Frontend::Agent::ModuleRegistration::Loader</Navigation>
        <Value>
            <Hash>
                <Item Key="CSS">
                    <Array>
                    </Array>
                </Item>
                <Item Key="JavaScript">
                    <Array>
                    </Array>
                </Item>
            </Hash>
        </Value>
    </Setting>
    <Setting Name="Frontend::Navigation###AgentHelloWorld###002-Filename"␣
↪Required="0" Valid="1">
        <Description Translatable="1">Main menu item registration.</
↪Description>
        <Navigation>Frontend::Agent::ModuleRegistration::MainMenu</Navigation>
        <Value>
            <Array>
                <DefaultItem ValueType="FrontendNavigation">
                    <Hash>
                    </Hash>
```

(下页继续)

(续上页)

```
                </DefaultItem>
                <Item>
                    <Hash>
                        <Item Key="Group">
                            <Array>
                                <Item>users</Item>
                            </Array>
                        </Item>
                        <Item Key="GroupRo">
                            <Array>
                            </Array>
                        </Item>
                        <Item Key="Description" Translatable="1">HelloWorld.</
↪Item>
                        <Item Key="Name" Translatable="1">HelloWorld</Item>
                        <Item Key="Link">Action=AgentHelloWorld</Item>
                        <Item Key="LinkOption"></Item>
                        <Item Key="NavBar">HelloWorld</Item>
                        <Item Key="Type">Menu</Item>
                        <Item Key="Block"></Item>
                        <Item Key="AccessKey"></Item>
                        <Item Key="Prio">8400</Item>
                    </Hash>
                </Item>
            </Array>
        </Value>
    </Setting>
</otrs_config>
```

3.1.3 前端模块

创建链接并执行系统配置后，将显示名为“HelloWorld”的新模块。调用它时，会显示一条错误消息，因为 OTRS 尚未找到匹配的前端模块。这是接下来要创建的东西。为此，我们创建以下文件：

```
# --
# Copyright (C) (year) (name of author) (email of author)
# --
# This software comes with ABSOLUTELY NO WARRANTY. For details, see
# the enclosed file COPYING for license information (GPL). If you
# did not receive this file, see https://www.gnu.org/licenses/gpl-3.0.txt.
# --

package Kernel::Modules::AgentHelloWorld;

use strict;
use warnings;

# Frontend modules are not handled by the ObjectManager.
our $ObjectManagerDisabled = 1;
```

(下页继续)

(续上页)

```
sub new {
    my ( $Type, %Param ) = @_;

    # allocate new hash for object
    my $Self = {%Param};
    bless ($Self, $Type);

    return $Self;
}

sub Run {
    my ( $Self, %Param ) = @_;
    my %Data = ();

    my $HelloWorldObject = $Kernel::OM->Get('Kernel::System::HelloWorld');
    my $LayoutObject     = $Kernel::OM->Get('Kernel::Output::HTML::Layout');

    $Data{HelloWorldText} = $HelloWorldObject->GetHelloWorldText();

    # build output
    my $Output = $LayoutObject->Header(Title => "HelloWorld");
    $Output   .= $LayoutObject->NavigationBar();
    $Output   .= $LayoutObject->Output(
        Data         => \%Data,
        TemplateFile => 'AgentHelloWorld',
    );
    $Output   .= $LayoutObject->Footer();

    return $Output;
}

1;
```

3.1.4 核心模块

接下来，我们使用以下内容为核心模块 /HelloWorld/Kernel/System/HelloWorld.pm 创建文件：

```
# --
# Copyright (C) (year) (name of author) (email of author)
# --
# This software comes with ABSOLUTELY NO WARRANTY. For details, see
# the enclosed file COPYING for license information (GPL). If you
# did not receive this file, see https://www.gnu.org/licenses/gpl-3.0.txt.
# --

package Kernel::System::HelloWorld;

use strict;
use warnings;
```

(下页继续)

(续上页)

```
# list your object dependencies (e.g. Kernel::System::DB) here
our @ObjectDependencies = (
    # 'Kernel::System::DB',
);

=head1 NAME

HelloWorld - Little "Hello World" module

=head1 DESCRIPTION

Little OTRS module that displays the text 'Hello World' when called up.

=head2 new()

Create an object. Do not use it directly, instead use:

    my $HelloWorldObject = $Kernel::OM->Get('Kernel::System::HelloWorld');

=cut

sub new {
    my ( $Type, %Param ) = @_;

    # allocate new hash for object
    my $Self = {};
    bless ($Self, $Type);

    return $Self;
}

=head2 GetHelloWorldText()

Return the "Hello World" text.

    my $HelloWorldText = $HelloWorldObject->GetHelloWorldText();

=cut

sub GetHelloWorldText {
    my ( $Self, %Param ) = @_;

    # Get the DBObject from the central object manager
    # my $DBObject = $Kernel::OM->Get('Kernel::System::DB');

    my $HelloWorld = $Self->_FormatHelloWorldText(
        String => 'Hello World',
    );

    return $HelloWorld;
}
```

(下页继续)

(续上页)

```
=begin Internal:

=head2 _FormatHelloWorldText()

Format "Hello World" text to uppercase

    my $HelloWorld = $Self->_FormatHelloWorldText(
        String => 'Hello World',
    );

=cut

sub _FormatHelloWorldText{
    my ( $Self, %Param ) = @_;

    my $HelloWorld = uc $Param{String};

    return $HelloWorld;

}

=end Internal:

1;
```

3.1.5 模板文件

新模块可以运行之前缺少的最后一件事是相关的 HTML 模板。因此，我们创建以下文件：

```
# --
# Copyright (C) (year) (name of author) (email of author)
# --
# This software comes with ABSOLUTELY NO WARRANTY. For details, see
# the enclosed file COPYING for license information (GPL). If you
# did not receive this file, see https://www.gnu.org/licenses/gpl-3.0.txt.
# --
<h1>[% Translate("Overview") | html %]: [% Translate("HelloWorld") %]</h1>
<p>
    [% Data.HelloWorldText | Translate() | html %]
</p>
```

该模块现在正在工作，并在调用时显示文本 *Hello World* 。

3.1.6 语言文件

如果要将文本 *Hello World*！翻译成其它语言如德语，您可以在 `HelloWorld/Kernel/Language/de_AgentHelloWorld.pm` 中为该语言创建翻译文件。例如：

```
package Kernel::Language::de_AgentHelloWorld;
```

(下页继续)

(续上页)

```
use strict;
use warnings;

sub Data {
    my $Self = shift;

    $Self->{Translation}->{'Hello World!'} = 'Hallo Welt!';

    return 1;
}
1;
```

3.1.7 摘要

上面给出的示例表明，为 OTRS 编写新模块并不困难 。但是，重要的是确保模块和文件名是唯一的，从而不会干扰框架或其他扩展模块 。模块完成后，必须从中生成一个 OPM 包（ 参见创建软件包 ）。

3.2 编写新的 OTRS 前端组件

在这个例子中，我们将尝试编写一个新的 OTRS 前端组件 。从 OTRS 7 开始，OTRS 框架支持用 Vue.js 编写的单页面应用程序前端，并基于新的 JavaScript 工具链 。第一次迭代包含新的外部接口，我们将尝试编写自定义组件 。您需要有一个正在运行的 OTRS 开发环境，如开发环境部分所述 。

3.2.1 目标

我们想要编写一个小的前端组件，在调用时显示文本 *Hello World* 。这将是一个路由组件，这意味着当使用精心设计的 URL 调用时，它将在外部人员界面中可用 。

3.2.2 使用 Skeleton 命令

为了加快开发速度，我们应该使用 skeleton 命令来获取我们可以构建的样板模板文件 。

在正在运行的 OTRS 实例上，调用以下命令以生成模板 。我们将使用 `HelloWorld` 作为新组件的名称：

```
bin/otrs.Console.pl Dev::Code::Generate::VueComponent --component-directory /
↪ws/MyPackage --component-subdirectory Apps/External/Components/Route --no-
↪docs HelloWorld
```

在命令中 `--component-directory` 是你的模块的目录，在 `Frontend/` 文件夹下的 `--component-subdirectory` 路径，将容纳组件文件 。现在，使用 `--no-docs` 开关跳过设计系统的文档组件的创建 。

此命令将生成具有以下路径的两个文件：

```
Generated: /ws/MyPackage/Frontend/Apps/External/Components/Route/HelloWorld.
↪vue
Generated: /ws/MyPackage/Frontend/Tests/Apps/External/Components/Route/
↪HelloWorld.js
Skipped creating documentation component.
```

3.2.3 路由配置

为了允许外部人员界面应用程序路由，我们需要添加指向组件的正确路由配置 。因此，我们使用以下定义创建文件 Kernel/Config/Files/XML/HelloWorld.xml：

```
<?xml version="1.0" encoding="utf-8" ?>
<otrs_config version="2.0" init="Application">
    <Setting Name="ExternalFrontend::Route###420-HelloWorld" Required="0"␣
↪Valid="1">
        <Description Translatable="1">Defines the application routes for the␣
↪external interface. Additional routes are defined by adding new items and␣
↪specifying their parameters. 'Group' and 'GroupRo' arrays can be used to␣
↪limit access of the route to members of certain groups with RW and RO␣
↪permissions respectively. 'Path' defines the relative path of the route, and
↪'Alias' can be used for specifying an alternative path. 'Component' is the␣
↪path of the Vue component responsible for displaying the route content,␣
↪relative to the Components/Route folder in the app. 'IsPublic' defines if␣
↪the route will be accessible for unauthenticated users and in case this is␣
↪set to '1', 'Group' and 'GroupRo' parameters will be ignored. 'Props' can␣
↪be used to signal that the path contain dynamic segments, and that their␣
↪values should be bound to the component as props (use '1' to turn on this␣
↪feature).</Description>
        <Navigation>Frontend::External::Route</Navigation>
        <Value>
            <Array>
                <DefaultItem ValueType="ApplicationRoute">
                    <Hash>
                    </Hash>
                </DefaultItem>
                <Item>
                    <Hash>
                        <Item Key="Group">
                            <Array>
                            </Array>
                        </Item>
                        <Item Key="GroupRo">
                            <Array>
                            </Array>
                        </Item>
                        <Item Key="Path">/hello-world/:headingText?</Item>
                        <Item Key="Alias"></Item>
                        <Item Key="Component">HelloWorld</Item>
                        <Item Key="IsPublic">1</Item>
                        <Item Key="Props">1</Item>
                    </Hash>
                </Item>
            </Array>
        </Value>
    </Setting>
</otrs_config>
```

- Group 和 GroupRo 可用于限制具有特定组的用户的路由屏幕 。请注意，这仅涉及经过身份验证的客户用户 。

- Path 实际上是路由组件可用的 slug。在本例中完整的 URL 将是 /external/hello-world，任何后续的路由组件都将作为名为 headingText 的参数传递。如果您的系统配置了 Frontend::PrefixPath，则会在其前面加上完整的 URL。
- Alias 可用于为同一路由提供别名（例如 /hello-world-alt）。它将指向相同的组件。
- Component 是组件标识符，文件名的第一部分，没有 .vue 扩展名。如果是组件文件夹，则它是根文件夹的名称。有关更多信息，请参阅组件目录。
- IsPublic 定义未经身份验证的用户是否可以访问该路由（0/空 - 不可访问，1 - 可访问）。
- Props 定义路由是否将 URI 参数作为 prop 值传递（0 / Empty - 未传递，1 - 传递）。有关更多信息，请参阅传递参数给路由组件。

3.2.4 组件模板代码

让我们现在启动代码编辑器，仔细看看我们的骨架命令创建的 HelloWorld.vue 文件。

该文件的顶部包含一个模板部分，其中应包含 Vue.js 模板代码。例如，让我们修改它，以便显示带有文本变量的标题：

```
<template>
    <main class="HelloWorld">
        <b-container>
            <b-row>
                <b-col>
                    <h1 class="HelloWorld__Heading">
                        {{ headingText | translate }}
                    </h1>
                </b-col>
            </b-row>
        </b-container>
    </main>
</template>
```

OTRS 支持多个过滤器，translate 就是其中之一。它甚至支持使用占位符值转换字符串文字，您可以像这样使用它：

```
{{ 'This is a %s.' | translate('string') }}
```

3.2.5 组件核心代码

接下来，我们为组件核心代码块添加对 prop 的支持，以下是适用于示例的已修改和删节版本：

```
<script>
export default {
    name: 'HelloWorld',

    props: {
        headingText: {
            type: String,
            default: translatable('Hello, world!'),
        },
    },
```

(下页继续)

(续上页)

```
};
</script>
```

这会向我们的组件添加一个名为 headingText 的 prop，它的类型为 string，并且具有合理的默认值 。

translatable() no-op 方法的用法仅限于标记出现在代码中的可翻译字符串 。请注意，对于通过管道传输到翻译过滤器的字符串文字，这不是必需的，因为这将从一开始就假定 。经验法则是在字符串未在定义位置翻译的任何位置使用标记 。

3.2.6 组件样式代码

最后，但并非最不重要，我们可以选择指定组件使用的样式 。为此，我们可以访问 SCSS，它是 SASS CSS 扩展集的一种风格 。要利用它，只需在组件文件的末尾添加样式标记：

```
<style lang="scss">
.HelloWorld {
    &__Heading {
        color: $primary;
    }
}
</style>
```

在样式块中，您可以访问某些全局变量和 mixin 。有关详细信息，请参阅框架代码(请参阅 Frontend/Styles/globals.js)。

请注意，虽然仅在引用组件时才会加载样式，但之后这些样式将全局可用，因为 CSS 对于同一页面本质上是全局的 。可以选择将样式范围仅限于组件，可以通过样式标记上的 scoped 属性来实现，但在设计类名时巧妙使用 BEM 方法可能不需要这样做 。【 译注：BEM–前端命名方法论,BEM 的意思就是块(block)、元素(element)、修饰符(modifier), 是由 Yandex 团队提出的一种前端命名方法论 。这种巧妙的命名方法让你的 CSS 类对其他开发者来说更加透明而且更有意义 。BEM 命名约定更加严格，而且包含更多的信息，它们用于一个团队开发一个耗时的大项目 。】

3.2.7 传递参数给路由组件

在上面的路由配置中，我们定义一个包含参数占位符(headingText)的路由路径 。通过激活 Props 标志，我们确保每次进入路由时，此参数的值将绑定到具有相同名称的组件 prop 。

例如，如果我们通过 /external/hello-world URL 进入路由，我们的组件 prop 将是未定义的，因此将获得其默认值 。

但是，如果我们通过 /external/hello-world/Value 访问路由，prop 将被设置为字符串 Value，甚至自动翻译为当前用户语言(如果已有翻译)。

3.2.8 组件目录

如果是自封闭组件，您可能需要随附一些其他文件 。有时最好模块化代码库，因为它更容易维护 。对于前端组件，您可以使用一种非常简单的方法：组件目录，而不是组件的单个 .vue 文件，将名为 index.vue 的文件包含在名为组件的目录中 。像这样的：

```
HelloWorld/
HelloWorld/index.vue
```

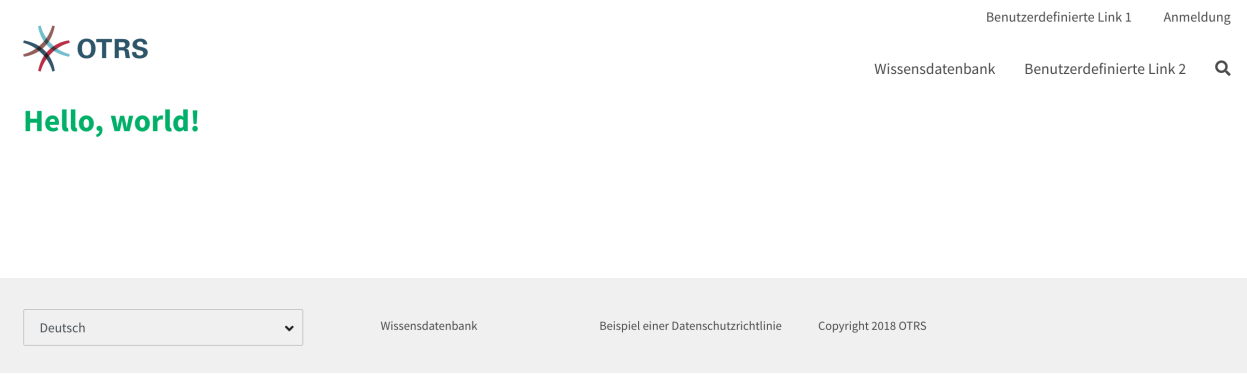

图 1: 传递参数 - 默认 Prop 值

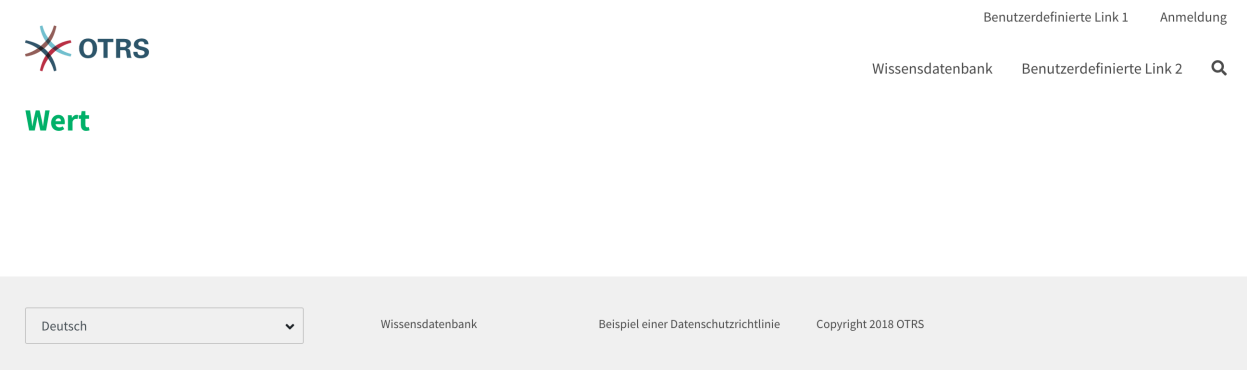

图 2: 传递参数 - 已翻译的 Prop 值

然后，只需按照理智的结构在同一目录中添加新文件：

```
HelloWorld/
HelloWorld/index.vue
HelloWorld/Styles/_mystyles.scss
HelloWorld/Images/foobar.png
HelloWorld/Fonts/awesome-font.woff
HelloWorld/Fonts/awesome-font.woff2
HelloWorld/ChildComponent1.vue
HelloWorld/ChildComponent2/index.vue
HelloWorld/ChildComponent2/Styles/_childstyles2.scss
```

你懂的 。然后可以通过相对路径引用新文件，以便在父组件(`index.vue`)中实现类似的功能：

```
<template>
    <img src="./Images/foobar.png" alt="Foobar" />
</template>
```

或者，像这样：

```
<script>
export default {
    name: 'HelloWorld',

    components: {
        ChildComponent1: () => import('./ChildComponent1'),
        ChildComponent2: () => import('./ChildComponent2'),
    },
...
```

甚至可以在正确的块中引用外部样式：

```
<style lang="scss">
@import './Styles/mystyles';
</style>
```

使用这种方法，您将在遵循逻辑树层次结构的单个目录中保留一个打包的组件，并使所有资源在需要时都易于查找 。

3.2.9 打包其它供应商模块

在某些情况下，您可能需要在包中附加其它 Node.js 模块 。遗憾的是，NPM 和 OTRS 都不支持在根 `node_modules/` 目录中轻松添加模块，但是有一种机制可以提供预先打包的模块文件 。

只需在包中创建一个 `Frontend/Vendor` 目录，然后在其中的子目录中添加模块资源 。

例如，假设我们想要发送一个有用的 `vue-full-calendar` 组件及其依赖项作为我们的包的一部分 。该组件具有以下 NPM 依赖项：

```
$ npm view vue-full-calendar dependencies
{ 'babel-plugin-transform-runtime': '^6.23.0', fullcalendar: '^3.4.0',
↪'lodash.defaultsdeep': '^4.6.0' }
```

但是，它的一些依赖项可能具有更多依赖项，我们也可以检查它们：

```
$ npm view babel-plugin-transform-runtime dependencies
{ 'babel-runtime': '^6.22.0' }

$ npm view fullcalendar dependencies
{ jquery: '2 - 3', moment: '^2.20.1' }

$ npm view lodash.defaultsdeep dependencies
```

快速检查会告诉我们 `babel-runtime` 和 `moment` 实际上都是 OTRS 框架依赖项的一部分：

```
/opt/otrs $ npm list babel-runtime
otrs-frontend@7.0.0-dev /ws/otrs7-mojo
  bootstrap-vue@2.0.0-rc.11
    opencollective@1.0.3
      babel-polyfill@6.23.0
        babel-runtime@6.26.0  deduped
  esdoc2@2.1.5
    babel-generator@6.26.0
      babel-messages@6.23.0
        babel-runtime@6.26.0  deduped
...

/opt/otrs $ npm list moment
otrs-frontend@7.0.0-dev /ws/otrs7-mojo
  moment-timezone@0.5.21
    moment@2.22.2
```

这意味着我们也不必打包这些模块，因为它们可以开箱即用。虽然检查所有依赖项很麻烦，但这是值得的，因为我们的包会更小。我们还将防止覆盖框架依赖性的问题，因为 `Frontend/Vendor` 总是获胜。

现在安装我们需要的东西并丢弃我们不需要的东西。最简单的方法是通过以下 NPM 命令：

```
/ws/MyPackage $ npm install vue-full-calendar --no-save
+ vue-full-calendar@2.7.0
added 9 packages from 14 contributors in 1.883s

/ws/MyPackage $ ls -1 node_modules/
babel-plugin-transform-runtime
babel-runtime
core-js
fullcalendar
jquery
lodash.defaultsdeep
moment
regenerator-runtime
vue-full-calendar
```

现在删除那些我们知道由框架提供的模块：

```
/ws/MyPackage $ rm -rf node_modules/babel-runtime node_modules/core-js node_
↪modules/moment node_modules/regenerator-runtime

/ws/MyPackage $ ls -1 node_modules/
babel-plugin-transform-runtime
```

(下页继续)

(续上页)

```
fullcalendar
jquery
lodash.defaultsdeep
vue-full-calendar
```

好多了。现在将模块移动到正确的位置：

```
/ws/MyPackage $ mkdir -p Frontend/Vendor
/ws/MyPackage $ mv node_modules/* Frontend/Vendor/
/ws/MyPackage $ rmdir node_modules/
```

最终的优化是从特定模块文件夹中删除不需要的文件。这可能会很复杂，但它值得，因为它会进一步减少模块的大小和需要包含在包中的文件数量。

例如，让我们从 `fullcalendar` 模块中删除最小化的 JS 文件，因为我们发现 Vue 组件仅使用完整的 dist 文件：

```
/ws/MyPackage $ rm Frontend/Vendor/fullcalendar/dist/*.min.*
```

由于 `fullcalendar` 也使用原始 dist 文件，因此删除 jQuery 源和最小化文件也是安全的：

```
/ws/MyPackage $ rm Frontend/Vendor/jquery/dist/*.min.*
/ws/MyPackage $ rm Frontend/Vendor/jquery/external/sizzle/dist/*.min.*
/ws/MyPackage $ rm -rf Frontend/Vendor/jquery/src
```

我们留下约 100 多个需要包含在我们的 SOPM 文件中的文件，就像任何其他常规包文件一样。完成此操作后，这些依赖项将在目标系统中存在并可解析：

```
/ws/MyPackage $ ls -la Frontend/Vendor
Frontend/Vendor
Frontend/Vendor/vue-full-calendar
Frontend/Vendor/vue-full-calendar/.babelrc
Frontend/Vendor/vue-full-calendar/LICENSE
Frontend/Vendor/vue-full-calendar/tests
Frontend/Vendor/vue-full-calendar/tests/fullcalendar.spec.js
Frontend/Vendor/vue-full-calendar/index.js
...
```

3.3 使用 OTRS 模块层的功能

OTRS 具有大量所谓的 模块层，这使得在不修补现有代码的情况下扩展系统变得非常容易。一个例子是工单的数字生成机制。它是一个带有可插拔模块的 模块层，如果您愿意，可以添加自己的自定义数字生成器模块。让我们详细看看不同的层！

3.3.1 服务人员身份验证模块

OTRS 框架附带了几个服务人员身份验证模块（ DB 、LDAP 和 HTTPBasicAuth ）。您也可以开发自己的身份验证模块。服务人员验证模块位于 `Kernel/System/Auth/*.pm`。有关其配置的更多信息，请参阅管理员手册。下面是一个简单服务人员验证模块的示例。将它保存为 `Kernel/System/Auth/Simple.pm`。你只需要 3 个函数：`new()` 、`GetOption()` 和 `Auth()`。返回 uid，然后验证就可以了。

服务人员身份验证模块代码示例

接口类称为 Kernel::System::Auth。示例服务人员身份验证可以叫做 Kernel::System::Auth::CustomAuth。你可以在下面找到一个例子。

```
# --
# Copyright (C) 2001-2020 OTRS AG, https://otrs.com/
# --
# This software comes with ABSOLUTELY NO WARRANTY. For details, see
# the enclosed file COPYING for license information (GPL). If you
# did not receive this file, see https://www.gnu.org/licenses/gpl-3.0.txt.
# --

package Kernel::System::Auth::CustomAuth;

use strict;
use warnings;

use Authen::CustomAuth;

sub new {
    my ( $Type, %Param ) = @_;

    # allocate new hash for object
    my $Self = {};
    bless( $Self, $Type );

    # check needed objects
    for (qw(LogObject ConfigObject DBObject)) {
        $Self->{$_} = $Param{$_} || die "No $_!";
    }

    # Debug 0=off 1=on
    $Self->{Debug} = 0;

    # get config
    $Self->{Die} = $Self->{ConfigObject}->Get( 'AuthModule::CustomAuth::Die' .
↪$Param{Count} );

    # get user table
    $Self->{CustomAuthHost} = $Self->{ConfigObject}->Get(
↪'AuthModule::CustomAuth::Host' . $Param{Count} )
        || die "Need AuthModule::CustomAuth::Host$Param{Count}.";
    $Self->{CustomAuthSecret}
        = $Self->{ConfigObject}->Get( 'AuthModule::CustomAuth::Password' .
↪$Param{Count} )
        || die "Need AuthModule::CustomAuth::Password$Param{Count}.";

    return $Self;
}

sub GetOption {
    my ( $Self, %Param ) = @_;
```

(下页继续)

(续上页)

```
    # check needed stuff
    if ( !$Param{What} ) {
        $Self->{LogObject}->Log( Priority => 'error', Message => "Need What!"␣
↪);
        return;
    }

    # module options
    my %Option = ( PreAuth => 0, );

    # return option
    return $Option{ $Param{What} };
}

sub Auth {
    my ( $Self, %Param ) = @_;

    # check needed stuff
    if ( !$Param{User} ) {
        $Self->{LogObject}->Log( Priority => 'error', Message => "Need User!"␣
↪);
        return;
    }

    # get params
    my $User       = $Param{User}       || '';
    my $Pw         = $Param{Pw}         || '';
    my $RemoteAddr = $ENV{REMOTE_ADDR} || 'Got no REMOTE_ADDR env!';
    my $UserID     = '';
    my $GetPw      = '';

    # just in case for debug!
    if ( $Self->{Debug} > 0 ) {
        $Self->{LogObject}->Log(
            Priority => 'notice',
            Message  => "User: '$User' tried to authenticate with Pw: '$Pw' (
↪$RemoteAddr)",
        );
    }

    # just a note
    if ( !$User ) {
        $Self->{LogObject}->Log(
            Priority => 'notice',
            Message  => "No User given!!! (REMOTE_ADDR: $RemoteAddr)",
        );
        return;
    }

    # just a note
    if ( !$Pw ) {
```

(下页继续)

(续上页)

```
        $Self->{LogObject}->Log(
            Priority => 'notice',
            Message  => "User: $User authentication without Pw!!! (REMOTE_
↪ADDR: $RemoteAddr)",
        );
        return;
    }

    # Create a RADIUS object
    my $CustomAuth = Authen::CustomAuth->new(
        Host   => $Self->{CustomAuthHost},
        Secret => $Self->{CustomAuthecret},
    );
    if ( !$CustomAuth ) {
        if ( $Self->{Die} ) {
            die "Can't connect to $Self->{CustomAuthHost}: $@";
        }
        else {
            $Self->{LogObject}->Log(
                Priority => 'error',
                Message  => "Can't connect to $Self->{CustomAuthHost}: $@",
            );
            return;
        }
    }
    my $AuthResult = $CustomAuth->check_pwd( $User, $Pw );

    # login note
    if ( defined($AuthResult) && $AuthResult == 1 ) {
        $Self->{LogObject}->Log(
            Priority => 'notice',
            Message  => "User: $User authentication ok (REMOTE_ADDR:
↪$RemoteAddr).",
        );
        return $User;
    }

    # just a note
    else {
        $Self->{LogObject}->Log(
            Priority => 'notice',
            Message  => "User: $User authentication with wrong Pw!!! (REMOTE_
↪ADDR: $RemoteAddr)"
        );
        return;
    }
}

1;
```

服务人员身份验证模块配置示例

需要激活自定义服务人员身份验证模块。这可以使用下面的 Perl 配置来完成。建议不要使用 XML 配置，因为您可能通过系统配置将您关在外面。

```
$Self->{'AuthModule'} = 'Kernel::System::Auth::CustomAuth';
```

服务人员身份验证模块用例示例

身份验证实现的有用示例可以是一个 SOAP 后端。

3.3.2 身份验证同步模块

OTRS 框架自带有一个 LDAP 认证同步模块。您也可以开发自己的身份验证模块。认证同步模块位于 `Kernel/System/Auth/Sync/*.pm` 下。有关其配置的更多信息，请参阅管理员手册。下面是一个身份验证同步模块的示例。将其保存在 `Kernel/System/Auth/Sync/CustomAuthSync.pm` 下。你只需要 2 个函数：`new()` 和 `Sync()`。返回 1，然后同步就可以了。

身份验证同步模块代码示例

接口类称为 `Kernel::System::Auth`。示例服务人员身份验证可以叫做 `Kernel::System::Auth::Sync::CustomAuthSync`。你可以在下面找到一个例子。

```
# --
# Copyright (C) 2001-2020 OTRS AG, https://otrs.com/
# --
# This software comes with ABSOLUTELY NO WARRANTY. For details, see
# the enclosed file COPYING for license information (GPL). If you
# did not receive this file, see https://www.gnu.org/licenses/gpl-3.0.txt.
# --

package Kernel::System::Auth::Sync::CustomAuthSync;

use strict;
use warnings;
use Net::LDAP;

sub new {
    my ( $Type, %Param ) = @_;

    # allocate new hash for object
    my $Self = {};
    bless( $Self, $Type );

    # check needed objects
    for (qw(LogObject ConfigObject DBObject UserObject GroupObject␣
↪EncodeObject)) {
        $Self->{$_} = $Param{$_} || die "No $_!";
    }

    # Debug 0=off 1=on
```

(下页继续)

(续上页)

```
    $Self->{Debug} = 0;

...

    return $Self;
}

sub Sync {
    my ( $Self, %Param ) = @_;

    # check needed stuff
    for (qw(User)) {
        if ( !$Param{$_} ) {
            $Self->{LogObject}->Log( Priority => 'error', Message => "Need $_!
↪" );
            return;
        }
    }
...
    return 1;
}
```

身份验证同步模块配置示例

您应该激活自定义同步模块。这可以使用下面的 Perl 配置来完成。建议不要使用 XML 配置，因为您可能通过系统配置将您关在外面。

```
$Self->{'AuthSyncModule'} = 'Kernel::System::Auth::Sync::LDAP';
```

身份验证同步模块用例示例

同步实现的有用示例可以是一个 SOAP 或 RADIUS 后端。

3.3.3 客户身份验证模块

OTRS 框架附带了几个客户身份验证模块（ DB 、LDAP 和 HTTPBasicAuth ）。您也可以开发自己的身份验证模块。客户认证模块位于 `Kernel/System/CustomerAuth/*.pm` 下。有关其配置的更多信息，请参阅管理员手册。下面是一个简单的客户验证模块的示例。将它保存为 `Kernel/System/CustomerAuth/Simple.pm`。你只需要 3 个函数：`new()`、`GetOption()` 和 `Auth()`。返回 uid，然后验证就可以了。

客户身份验证模块代码示例

接口类称为 `Kernel::System::CustomerAuth`。示例客户认证可以叫做 `Kernel::System::CustomerAuth::CustomAuth`。你可以在下面找到一个例子。

```
# --
# Copyright (C) 2001-2020 OTRS AG, https://otrs.com/
# --
```

(下页继续)

(续上页)

```
# This software comes with ABSOLUTELY NO WARRANTY. For details, see
# the enclosed file COPYING for license information (GPL). If you
# did not receive this file, see https://www.gnu.org/licenses/gpl-3.0.txt.
# --

package Kernel::System::CustomerAuth::CustomAuth;

use strict;
use warnings;

use Authen::CustomAuth;

sub new {
    my ( $Type, %Param ) = @_;

    # allocate new hash for object
    my $Self = {};
    bless( $Self, $Type );

    # check needed objects
    for (qw(LogObject ConfigObject DBObject)) {
        $Self->{$_} = $Param{$_} || die "No $_!";
    }

    # Debug 0=off 1=on
    $Self->{Debug} = 0;

    # get config
    $Self->{Die}
        = $Self->{ConfigObject}->Get( 'Customer::AuthModule::CustomAuth::Die' .
↪ $Param{Count} );

    # get user table
    $Self->{CustomAuthHost}
        = $Self->{ConfigObject}->Get( 'Customer::AuthModule::CustomAuth::Host'␣
↪. $Param{Count} )
        || die "Need Customer::AuthModule::CustomAuth::Host$Param{Count} in␣
↪Kernel/Config.pm";
    $Self->{CustomAuthSecret}
        = $Self->{ConfigObject}->Get(
↪'Customer::AuthModule::CustomAuth::Password' . $Param{Count} )
        || die "Need Customer::AuthModule::CustomAuth::Password$Param{Count}␣
↪in Kernel/Config.pm";

    return $Self;
}

sub GetOption {
    my ( $Self, %Param ) = @_;

    # check needed stuff
    if ( !$Param{What} ) {
```

(下页继续)

(续上页)

```
        $Self->{LogObject}->Log( Priority => 'error', Message => "Need What!"␣
↪);
        return;
    }

    # module options
    my %Option = ( PreAuth => 0, );

    # return option
    return $Option{ $Param{What} };
}

sub Auth {
    my ( $Self, %Param ) = @_;

    # check needed stuff
    if ( !$Param{User} ) {
        $Self->{LogObject}->Log( Priority => 'error', Message => "Need User!"␣
↪);
        return;
    }

    # get params
    my $User       = $Param{User}       || '';
    my $Pw         = $Param{Pw}         || '';
    my $RemoteAddr = $ENV{REMOTE_ADDR} || 'Got no REMOTE_ADDR env!';
    my $UserID     = '';
    my $GetPw      = '';

    # just in case for debug!
    if ( $Self->{Debug} > 0 ) {
        $Self->{LogObject}->Log(
            Priority => 'notice',
            Message  => "User: '$User' tried to authentificate with Pw: '$Pw' (
↪$RemoteAddr)",
        );
    }

    # just a note
    if ( !$User ) {
        $Self->{LogObject}->Log(
            Priority => 'notice',
            Message  => "No User given!!! (REMOTE_ADDR: $RemoteAddr)",
        );
        return;
    }

    # just a note
    if ( !$Pw ) {
        $Self->{LogObject}->Log(
            Priority => 'notice',
            Message  => "User: $User Authentication without Pw!!! (REMOTE_
↪ADDR: $RemoteAddr)",
```

(下页继续)

(续上页)

```
        );
        return;
    }

    # Create a custom object
    my $CustomAuth = Authen::CustomAuth->new(
        Host   => $Self->{CustomAuthHost},
        Secret => $Self->{CustomAuthSecret},
    );
    if ( !$CustomAuth ) {
        if ( $Self->{Die} ) {
            die "Can't connect to $Self->{CustomAuthHost}: $@";
        }
        else {
            $Self->{LogObject}->Log(
                Priority => 'error',
                Message  => "Can't connect to $Self->{CustomAuthHost}: $@",
            );
            return;
        }
    }
    my $AuthResult = $CustomAuth->check_pwd( $User, $Pw );

    # login note
    if ( defined($AuthResult) && $AuthResult == 1 ) {
        $Self->{LogObject}->Log(
            Priority => 'notice',
            Message  => "User: $User Authentication ok (REMOTE_ADDR:
↪$RemoteAddr).",
        );
        return $User;
    }

    # just a note
    else {
        $Self->{LogObject}->Log(
            Priority => 'notice',
            Message  => "User: $User Authentication with wrong Pw!!! (REMOTE_
↪ADDR: $RemoteAddr)"
        );
        return;
    }
}

1;
```

客户身份验证模块配置示例

需要激活自定义客户身份验证模块。这可以使用下面的 XML 配置来完成。

```
<ConfigItem Name="AuthModule" Required="1" Valid="1">
    <Description Translatable="1">Module to authenticate customers.</
↪Description>
    <Group>Framework</Group>
    <SubGroup>Frontend::CustomerAuthAuth</SubGroup>
    <Setting>
        <Option Location="Kernel/System/CustomerAuth/*.pm" SelectedID=
↪"Kernel::System::CustomerAuth::CustomAuth"></Option>
    </Setting>
</ConfigItem>
```

客户身份验证模块用例示例

有用的身份验证实现可以是一个 SOAP 后端 。

3.3.4 客户用户首选项模块

OTRS 框架自带有一个 DB 客户用户首选项模块。您也可以开发自己的客户用户首选项模块。客户用户首选项模块位于 Kernel/System/CustomerUser/Preferences/*.pm 下。有关其配置的更多信息，请参阅管理员手册。下面是一个客户用户首选项模块的示例。将其保存为 Kernel/System/CustomerUser/Preferences/Custom.pm。你只需要 4 个函数：new()、SearchPreferences()、SetPreferences() 和 GetPreferences()。

客户用户首选项模块代码示例

接口类称为 Kernel::System::CustomerUser。示例客户用户首选项可以叫做 Kernel::System::CustomerUser::Preferences::Custom。你可以在下面找到一个例子 。

```
# --
# Copyright (C) 2001-2020 OTRS AG, https://otrs.com/
# --
# This software comes with ABSOLUTELY NO WARRANTY. For details, see
# the enclosed file COPYING for license information (GPL). If you
# did not receive this file, see https://www.gnu.org/licenses/gpl-3.0.txt.
# --

package Kernel::System::CustomerUser::Preferences::Custom;

use strict;
use warnings;

use vars qw(@ISA $VERSION);

sub new {
    my ( $Type, %Param ) = @_;

    # allocate new hash for object
    my $Self = {};
    bless( $Self, $Type );
```

(下页继续)

(续上页)

```
    # check needed objects
    for my $Object (qw(DBObject ConfigObject LogObject)) {
        $Self->{$Object} = $Param{$Object} || die "Got no $Object!";
    }

    # preferences table data
    $Self->{PreferencesTable} = $Self->{ConfigObject}->Get('CustomerPreferences
↪')->{Params}->{Table}
        || 'customer_preferences';
    $Self->{PreferencesTableKey}
        = $Self->{ConfigObject}->Get('CustomerPreferences')->{Params}->
↪{TableKey}
        || 'preferences_key';
    $Self->{PreferencesTableValue}
        = $Self->{ConfigObject}->Get('CustomerPreferences')->{Params}->
↪{TableValue}
        || 'preferences_value';
    $Self->{PreferencesTableUserID}
        = $Self->{ConfigObject}->Get('CustomerPreferences')->{Params}->
↪{TableUserID}
        || 'user_id';

    return $Self;
}

sub SetPreferences {
    my ( $Self, %Param ) = @_;

    my $UserID = $Param{UserID} || return;
    my $Key    = $Param{Key}    || return;
    my $Value = defined( $Param{Value} ) ? $Param{Value} : '';

    # delete old data
    return if !$Self->{DBObject}->Do(
        SQL => "DELETE FROM $Self->{PreferencesTable} WHERE "
            . " $Self->{PreferencesTableUserID} = ? AND $Self->
↪{PreferencesTableKey} = ?",
        Bind => [ \$UserID, \$Key ],
    );

    $Value .= 'Custom';

    # insert new data
    return if !$Self->{DBObject}->Do(
        SQL => "INSERT INTO $Self->{PreferencesTable} ($Self->
↪{PreferencesTableUserID}, "
            . " $Self->{PreferencesTableKey}, $Self->{PreferencesTableValue})
↪"
            . " VALUES (?, ?, ?)",
        Bind => [ \$UserID, \$Key, \$Value ],
    );
```

(下页继续)

(续上页)

```
    return 1;
}

sub GetPreferences {
    my ( $Self, %Param ) = @_;

    my $UserID = $Param{UserID} || return;
    my %Data;

    # get preferences

    return if !$Self->{DBObject}->Prepare(
        SQL => "SELECT $Self->{PreferencesTableKey}, $Self->
↪{PreferencesTableValue} "
            . " FROM $Self->{PreferencesTable} WHERE $Self->
↪{PreferencesTableUserID} = ?",
        Bind => [ \$UserID ],
    );
    while ( my @Row = $Self->{DBObject}->FetchrowArray() ) {
        $Data{ $Row[0] } = $Row[1];
    }

    # return data
    return %Data;
}

sub SearchPreferences {
    my ( $Self, %Param ) = @_;

    my %UserID;
    my $Key   = $Param{Key}   || '';
    my $Value = $Param{Value} || '';

    # get preferences
    my $SQL = "SELECT $Self->{PreferencesTableUserID}, $Self->
↪{PreferencesTableValue} "
        . " FROM "
        . " $Self->{PreferencesTable} "
        . " WHERE "
        . " $Self->{PreferencesTableKey} = '"
        . $Self->{DBObject}->Quote($Key) . "'" . " AND "
        . " LOWER($Self->{PreferencesTableValue}) LIKE LOWER('"
        . $Self->{DBObject}->Quote( $Value, 'Like' ) . "')";

    return if !$Self->{DBObject}->Prepare( SQL => $SQL );
    while ( my @Row = $Self->{DBObject}->FetchrowArray() ) {
        $UserID{ $Row[0] } = $Row[1];
    }

    # return data
    return %UserID;
}
```

(下页继续)

(续上页)

```
1;
```

客户用户首选项模块配置示例

需要激活自定义客户用户首选项模块。这可以使用下面的 XML 配置来完成。

```
<ConfigItem Name="CustomerPreferences" Required="1" Valid="1">
    <Description Translatable="1">Parameters for the customer preference␣
↪table.</Description>
    <Group>Framework</Group>
    <SubGroup>Frontend::Customer::Preferences</SubGroup>
    <Setting>
        <Hash>
            <Item Key="Module">
↪Kernel::System::CustomerUser::Preferences::Custom</Item>
            <Item  Key="Params">
                <Hash>
                    <Item Key="Table">customer_preferences</Item>
                    <Item Key="TableKey">preferences_key</Item>
                    <Item Key="TableValue">preferences_value</Item>
                    <Item Key="TableUserID">user_id</Item>
                </Hash>
            </Item>
        </Hash>
    </Setting>
</ConfigItem>
```

客户用户首选项模块用例示例

有用的首选项实现可以是一个 SOAP 或 LDAP 后端。

3.3.5 队列首选项模块

有一个与 OTRS 框架一起提供的 DB 队列首选项模块。还可以开发自己的队列首选项模块。队列首选项模块位于 `kernel/system/queue/*.pm` 下。有关其配置的更多信息，请参阅管理手册。下面是队列首选项模块的示例。将其保存在 `kernel/system/queue/preferrencescustom.pm` 下。您只需要 3 个函数：`new()`、`queuePreferrencesSet()` 和 `queuePreferrencesGet()`。返回 1，则同步正常。

队列首选项代码示例

接口类称为 `Kernel::System::Queue`。示例队列首选项可以叫做 `Kernel::System::Queue::PreferencesCustom`。您可以在下面找到一个示例。

```
# --
# Copyright (C) 2001-2020 OTRS AG, https://otrs.com/
# --
# This software comes with ABSOLUTELY NO WARRANTY. For details, see
```

(下页继续)

(续上页)

```
# the enclosed file COPYING for license information (GPL). If you
# did not receive this file, see https://www.gnu.org/licenses/gpl-3.0.txt.
# --

package Kernel::System::Queue::PreferencesCustom;

use strict;
use warnings;

use vars qw(@ISA $VERSION);

sub new {
    my ( $Type, %Param ) = @_;

    # allocate new hash for object
    my $Self = {};
    bless( $Self, $Type );

    # check needed objects
    for (qw(DBObject ConfigObject LogObject)) {
        $Self->{$_} = $Param{$_} || die "Got no $_!";
    }

    # preferences table data
    $Self->{PreferencesTable}        = 'queue_preferences';
    $Self->{PreferencesTableKey}     = 'preferences_key';
    $Self->{PreferencesTableValue}   = 'preferences_value';
    $Self->{PreferencesTableQueueID} = 'queue_id';

    return $Self;
}

sub QueuePreferencesSet {
    my ( $Self, %Param ) = @_;

    # check needed stuff
    for (qw(QueueID Key Value)) {
        if ( !defined( $Param{$_} ) ) {
            $Self->{LogObject}->Log( Priority => 'error', Message => "Need $_!
↪" );
            return;
        }
    }

    # delete old data
    return if !$Self->{DBObject}->Do(
        SQL => "DELETE FROM $Self->{PreferencesTable} WHERE "
            . "$Self->{PreferencesTableQueueID} = ? AND $Self->
↪{PreferencesTableKey} = ?",
        Bind => [ \$Param{QueueID}, \$Param{Key} ],
    );
```

(下页继续)

(续上页)

```
    $Self->{PreferencesTableValue} .= 'PreferencesCustom';

    # insert new data
    return $Self->{DBObject}->Do(
        SQL => "INSERT INTO $Self->{PreferencesTable} ($Self->
↪{PreferencesTableQueueID}, "
            . " $Self->{PreferencesTableKey}, $Self->{PreferencesTableValue})
↪"
            . " VALUES (?, ?, ?)",
        Bind => [ \$Param{QueueID}, \$Param{Key}, \$Param{Value} ],
    );
}

sub QueuePreferencesGet {
    my ( $Self, %Param ) = @_;

    # check needed stuff
    for (qw(QueueID)) {
        if ( !$Param{$_} ) {
            $Self->{LogObject}->Log( Priority => 'error', Message => "Need $_!
↪" );
            return;
        }
    }

    # check if queue preferences are available
    if ( !$Self->{ConfigObject}->Get('QueuePreferences') ) {
        return;
    }

    # get preferences
    return if !$Self->{DBObject}->Prepare(
        SQL => "SELECT $Self->{PreferencesTableKey}, $Self->
↪{PreferencesTableValue} "
            . " FROM $Self->{PreferencesTable} WHERE $Self->
↪{PreferencesTableQueueID} = ?",
        Bind => [ \$Param{QueueID} ],
    );
    my %Data;
    while ( my @Row = $Self->{DBObject}->FetchrowArray() ) {
        $Data{ $Row[0] } = $Row[1];
    }

    # return data
    return %Data;
}

1;
```

队列首选项配置示例

需要激活自定义的队列首选项模块。这可以使用下面的 XML 配置来完成。

```
<ConfigItem Name="Queue::PreferencesModule" Required="1" Valid="1">
    <Description Translatable="1">Default queue preferences module.</
↪Description>
    <Group>Ticket</Group>
    <SubGroup>Frontend::Queue::Preferences</SubGroup>
    <Setting>
        <String Regex="">Kernel::System::Queue::PreferencesCustom</String>
    </Setting>
</ConfigItem>
```

队列首选项用例示例

有用的首选项实现可以是 SOAP 或 RADIUS 后端。

3.3.6 服务首选项模块

有一个与 OTRS 框架一起提供的 DB 服务首选项模块。还可以开发自己的服务首选项模块。服务首选项模块位于 `kernel/system/service/*.pm` 下。有关其配置的更多信息，请参阅管理手册。下面是服务首选项模块的示例。保存在 `kernel/system/service/preferrencescustom.pm` 下。您只需要 3 个函数：`new()`、`ServicePreferrencesSet()` 和 `ServicePreferrencesGet()`。返回 1，则同步正常。

服务首选项代码示例

接口类称为 `Kernel::System::Service`。示例服务首选项可以叫做 `Kernel::System::Service::PreferencesCustom`。您可以在下面找到一个示例。

```
# --
# Copyright (C) 2001-2020 OTRS AG, https://otrs.com/
# --
# This software comes with ABSOLUTELY NO WARRANTY. For details, see
# the enclosed file COPYING for license information (GPL). If you
# did not receive this file, see https://www.gnu.org/licenses/gpl-3.0.txt.
# --

package Kernel::System::Service::PreferencesCustom;

use strict;
use warnings;

use vars qw(@ISA $VERSION);

sub new {
    my ( $Type, %Param ) = @_;

    # allocate new hash for object
    my $Self = {};
    bless( $Self, $Type );
```

(下页继续)

(续上页)

```
    # check needed objects
    for (qw(DBObject ConfigObject LogObject)) {
        $Self->{$_} = $Param{$_} || die "Got no $_!";
    }

    # preferences table data
    $Self->{PreferencesTable}          = 'service_preferences';
    $Self->{PreferencesTableKey}       = 'preferences_key';
    $Self->{PreferencesTableValue}     = 'preferences_value';
    $Self->{PreferencesTableServiceID} = 'service_id';

    return $Self;
}

sub ServicePreferencesSet {
    my ( $Self, %Param ) = @_;

    # check needed stuff
    for (qw(ServiceID Key Value)) {
        if ( !defined( $Param{$_} ) ) {
            $Self->{LogObject}->Log( Priority => 'error', Message => "Need $_!
↪" );
            return;
        }
    }

    # delete old data
    return if !$Self->{DBObject}->Do(
        SQL => "DELETE FROM $Self->{PreferencesTable} WHERE "
            . "$Self->{PreferencesTableServiceID} = ? AND $Self->
↪{PreferencesTableKey} = ?",
        Bind => [ \$Param{ServiceID}, \$Param{Key} ],
    );

$Self->{PreferencesTableValue} .= 'PreferencesCustom';

    # insert new data
    return $Self->{DBObject}->Do(
        SQL => "INSERT INTO $Self->{PreferencesTable} ($Self->
↪{PreferencesTableServiceID}, "
            . " $Self->{PreferencesTableKey}, $Self->{PreferencesTableValue})
↪"
            . " VALUES (?, ?, ?)",
        Bind => [ \$Param{ServiceID}, \$Param{Key}, \$Param{Value} ],
    );
}

sub ServicePreferencesGet {
    my ( $Self, %Param ) = @_;

    # check needed stuff
```

(下页继续)

(续上页)

```
    for (qw(ServiceID)) {
        if ( !$Param{$_} ) {
            $Self->{LogObject}->Log( Priority => 'error', Message => "Need $_!
↪" );
            return;
        }
    }

    # check if service preferences are available
    if ( !$Self->{ConfigObject}->Get('ServicePreferences') ) {
        return;
    }

    # get preferences
    return if !$Self->{DBObject}->Prepare(
        SQL => "SELECT $Self->{PreferencesTableKey}, $Self->
↪{PreferencesTableValue} "
            . " FROM $Self->{PreferencesTable} WHERE $Self->
↪{PreferencesTableServiceID} = ?",
        Bind => [ \$Param{ServiceID} ],
    );
    my %Data;
    while ( my @Row = $Self->{DBObject}->FetchrowArray() ) {
        $Data{ $Row[0] } = $Row[1];
    }

    # return data
    return %Data;
}

1;
```

服务首选项配置示例

需要激活自定义的服务首选项模块。这可以使用下面的 XML 配置来完成。

```
<ConfigItem Name="Service::PreferencesModule" Required="1" Valid="1">
    <Description Translatable="1">Default service preferences module.</
↪Description>
    <Group>Ticket</Group>
    <SubGroup>Frontend::Service::Preferences</SubGroup>
    <Setting>
        <String Regex="">Kernel::System::Service::PreferencesCustom</String>
    </Setting>
</ConfigItem>
```

服务首选项用例示例

有用的首选项实现可以是 SOAP 或 RADIUS 后端。

3.3.7 SLA 首选项模块

有一个与 OTRS 框架一起提供的 DB SLA 首选项模块。还可以开发自己的 SLA 首选项模块。SLA 首选项模块位于 kernel/system/sla/*.pm 下。有关其配置的更多信息，请参阅管理手册。下面是一个 SLA 首选项模块的示例。将其保存在 kernel/system/sla/preferrencescustom.pm 下。您只需要 3 个函数：new()、SLAPreferencesSet() 和 SLAPreferencesGet()。确保函数返回 1。

SLA 首选项代码示例

接口类称为 Kernel::System::SLA。示例 SLA 首选项可以叫做 Kernel::System::SLA::PreferencesCustom。您可以在下面找到一个示例。

```
# --
# Copyright (C) 2001-2020 OTRS AG, https://otrs.com/
# --
# This software comes with ABSOLUTELY NO WARRANTY. For details, see
# the enclosed file COPYING for license information (GPL). If you
# did not receive this file, see https://www.gnu.org/licenses/gpl-3.0.txt.
# --

package Kernel::System::SLA::PreferencesCustom;

use strict;
use warnings;

use vars qw(@ISA);

sub new {
    my ( $Type, %Param ) = @_;

    # allocate new hash for object
    my $Self = {};
    bless( $Self, $Type );

    # check needed objects
    for (qw(DBObject ConfigObject LogObject)) {
        $Self->{$_} = $Param{$_} || die "Got no $_!";
    }

    # preferences table data
    $Self->{PreferencesTable}      = 'sla_preferences';
    $Self->{PreferencesTableKey}   = 'preferences_key';
    $Self->{PreferencesTableValue} = 'preferences_value';
    $Self->{PreferencesTableSLAID} = 'sla_id';

    return $Self;
}

sub SLAPreferencesSet {
    my ( $Self, %Param ) = @_;

    # check needed stuff
```

(下页继续)

(续上页)

```
    for (qw(SLAID Key Value)) {
        if ( !defined( $Param{$_} ) ) {
            $Self->{LogObject}->Log( Priority => 'error', Message => "Need $_!
↪" );
            return;
        }
    }

    # delete old data
    return if !$Self->{DBObject}->Do(
        SQL => "DELETE FROM $Self->{PreferencesTable} WHERE "
            . "$Self->{PreferencesTableSLAID} = ? AND $Self->
↪{PreferencesTableKey} = ?",
        Bind => [ \$Param{SLAID}, \$Param{Key} ],
    );

$Self->{PreferencesTableValue} .= 'PreferencesCustom';

    # insert new data
    return $Self->{DBObject}->Do(
        SQL => "INSERT INTO $Self->{PreferencesTable} ($Self->
↪{PreferencesTableSLAID}, "
            . " $Self->{PreferencesTableKey}, $Self->{PreferencesTableValue})
↪"
            . " VALUES (?, ?, ?)",
        Bind => [ \$Param{SLAID}, \$Param{Key}, \$Param{Value} ],
    );
}

sub SLAPreferencesGet {
    my ( $Self, %Param ) = @_;

    # check needed stuff
    for (qw(SLAID)) {
        if ( !$Param{$_} ) {
            $Self->{LogObject}->Log( Priority => 'error', Message => "Need $_!
↪" );
            return;
        }
    }

    # check if SLA preferences are available
    if ( !$Self->{ConfigObject}->Get('SLAPreferences') ) {
        return;
    }

    # get preferences
    return if !$Self->{DBObject}->Prepare(
        SQL => "SELECT $Self->{PreferencesTableKey}, $Self->
↪{PreferencesTableValue} "
            . " FROM $Self->{PreferencesTable} WHERE $Self->
↪{PreferencesTableSLAID} = ?",
```

(下页继续)

(续上页)

```
        Bind => [ \$Param{SLAID} ],
    );
    my %Data;
    while ( my @Row = $Self->{DBObject}->FetchrowArray() ) {
        $Data{ $Row[0] } = $Row[1];
    }

    # return data
    return %Data;
}

1;
```

SLA 首选项配置示例

需要激活自定义的 SLA 首选项模块。这可以使用下面的 XML 配置来完成。

```
<ConfigItem Name="SLA::PreferencesModule" Required="1" Valid="1">
    <Description Translatable="1">Default SLA preferences module.</
↪Description>
    <Group>Ticket</Group>
    <SubGroup>Frontend::SLA::Preferences</SubGroup>
    <Setting>
        <String Regex="">Kernel::System::SLA::PreferencesCustom</String>
    </Setting>
</ConfigItem>
```

SLA 首选项用例示例

有用的首选项实现可以在 SLA 上存储额外的值。

3.3.8 日志模块

OTRS 有一个全局日志接口，可以创建自己的日志后端。

编写自己的日志记录后端就像重新实现 `Kernel::System::Log::Log()` 方法一样简单。

日志模块代码示例

在这个小例子中，我们将编写一个小的文件日志后端，其工作方式类似于 `Kernel::System::Log::File`，但是为每个日志条目添加一个字符串。

```
# --
# Copyright (C) 2001-2020 OTRS AG, https://otrs.com/
# --
# This software comes with ABSOLUTELY NO WARRANTY. For details, see
# the enclosed file COPYING for license information (GPL). If you
# did not receive this file, see https://www.gnu.org/licenses/gpl-3.0.txt.
# --
```

(下页继续)

(续上页)

```
package Kernel::System::Log::CustomFile;

use strict;
use warnings;

umask "002";

sub new {
    my ( $Type, %Param ) = @_;

    # allocate new hash for object
    my $Self = {};
    bless( $Self, $Type );

    # get needed objects
    for (qw(ConfigObject EncodeObject)) {
        if ( $Param{$_} ) {
            $Self->{$_} = $Param{$_};
        }
        else {
            die "Got no $_!";
        }
    }

    # get logfile location
    $Self->{LogFile} = '/var/log/CustomFile.log';

    # set custom prefix
    $Self->{CustomPrefix} = 'CustomFileExample';

    # Fixed bug# 2265 - For IIS we need to create a own error log file.
    # Bind stderr to log file, because IIS do print stderr to web page.
    if ( $ENV{SERVER_SOFTWARE} && $ENV{SERVER_SOFTWARE} =~ /^microsoft\-iis/i␣
↪) {
        if ( !open STDERR, '>>', $Self->{LogFile} . '.error' ) {
            print STDERR "ERROR: Can't write $Self->{LogFile}.error: $!";
        }
    }

    return $Self;
}

sub Log {
    my ( $Self, %Param ) = @_;

    my $FH;

    # open logfile
    if ( !open $FH, '>>', $Self->{LogFile} ) {

        # print error screen
```

(下页继续)

(续上页)

```
        print STDERR "\n";
        print STDERR " >> Can't write $Self->{LogFile}: $! <<\n";
        print STDERR "\n";
        return;
    }

    # write log file
    $Self->{EncodeObject}->SetIO($FH);
    print $FH '[' . localtime() . ']';
    if ( lc $Param{Priority} eq 'debug' ) {
        print $FH "[Debug][$Param{Module}][$Param{Line}] $Self->{CustomPrefix}
↪$Param{Message}\n";
    }
    elsif ( lc $Param{Priority} eq 'info' ) {
        print $FH "[Info][$Param{Module}]  $Self->{CustomPrefix} $Param
↪{Message}\n";
    }
    elsif ( lc $Param{Priority} eq 'notice' ) {
        print $FH "[Notice][$Param{Module}] $Self->{CustomPrefix} $Param
↪{Message}\n";
    }
    elsif ( lc $Param{Priority} eq 'error' ) {
        print $FH "[Error][$Param{Module}][$Param{Line}] $Self->{CustomPrefix}
↪$Param{Message}\n";
    }
    else {

        # print error messages to STDERR
        print STDERR
            "[Error][$Param{Module}] $Self->{CustomPrefix} Priority: '$Param
↪{Priority}' not defined! Message: $Param{Message}\n";

        # and of course to logfile
        print $FH
            "[Error][$Param{Module}] $Self->{CustomPrefix} Priority: '$Param
↪{Priority}' not defined! Message: $Param{Message}\n";
    }

    # close file handle
    close $FH;
    return 1;
}

1;
```

日志模块配置示例

要激活我们的自定义日志模块，管理员可以手动将现有配置项 LogModule 设置为 Kernel::System::Log::CustomFile。要自动实现此功能，您可以提供覆盖默认设置的 XML 配置文件。

```
<ConfigItem Name="LogModule" Required="1" Valid="1">
    <Description Translatable="1">Set Kernel::System::Log::CustomFile as␣
↪default logging backend.</Description>
    <Group>Framework</Group>
    <SubGroup>Core::Log</SubGroup>
    <Setting>
        <Option Location="Kernel/System/Log/*.pm" SelectedID=
↪"Kernel::System::Log::CustomFile"></Option>
    </Setting>
</ConfigItem>
```

日志模块用例示例

有用的日志后端可以记录到 Web 服务或加密文件 。

注解: `Kernel::System::Log` 除了 `Log()` 之外还有其它方法，它们不能重新实现，例如用于处理共享内存段和日志数据缓存的代码 。

3.3.9 输出过滤器

输出过滤器允许动态修改 HTML 。最佳做法是使用输出过滤器而不是直接修改 `.tt` 文件 。这有三个很好的理由 。当必须将相同的适配应用于多个前端模块时，则仅需要实现一次适配 。第二个优点是当 OTRS 升级时，当相关模式没有改变时，有可能不必更新过滤器 。当两个扩展修改同一文件时，在安装第二个包期间会发生冲突 。通过使用修改相同前端模块的两个输出过滤器可以解决此冲突 。

有三种不同的输出过滤器 。他们在生成 HTML 内容的不同阶段激活 。

FilterElementPost

这些过滤器允许在渲染模板后修改模板的输出 。

要翻译内容，您可以直接运行 `$LayoutObject->Translate()` 。如果您需要其它模板功能，只需为输出过滤器定义一个小的模板文件，并在将其注入主数据之前使用它来呈现您的内容 。在某些情况下使用 jQuery DOM 操作来重新排序/替换屏幕上的内容，而不是使用正则表达式也是有帮助的 。在这种情况下，您会将新代码注入页面中的某个地方作为不可见的内容(例如，使用 `Hidden` 类)，然后将其与 jQuery 一起移动到 DOM 中的正确位置并显示它 。

为了更容易地使用渲染后输出过滤器，还有一种机制来为某些模板/块请求 HTML 注释钩子 。您可以在模块中添加配置 XML，如：

```
<Setting Name="Frontend::Template::GenerateBlockHooks###100-OTRSBusiness-
↪ContactWithData" Required="1" Valid="1">
    <Description Translatable="1">Generate HTML comment hooks for the␣
↪specified blocks so that filters can use them.</Description>
    <Navigation>Frontend::Base::OutputFilter</Navigation>
    <Value>
        <Hash>
            <Item Key="AgentTicketZoom">
                <Array>
                    <Item>CustomerTable</Item>
```

(下页继续)

(续上页)

```
                </Array>
            </Item>
        </Hash>
    </Value>
</Setting>
```

这将导致 AgentTicketZoom.tt 中的 CustomerTable 块在每次渲染时都封装在 HTML 注释中：

```
<!--HookStartCustomerTable-->
... block output ...
<!--HookEndCustomerTable-->
```

使用这种机制，每个包都可以请求它所需的块钩子，并且它们一致地被渲染。然后，可以在输出过滤器中使用这些 HTML 注释，以便于正则表达式匹配。

FilterContent

这种过滤器允许在将请求发送到浏览器之前处理请求的完整 HTML 输出。这可以用于全局转换。

FilterText

这种输出过滤器是 Kernel::Output::HTML::Layout::Ascii2HTML() 方法的插件，仅当参数 LinkFeature 设置为 1 时才有效。这样 FilterText 输出过滤器目前仅用于显示纯文本信件的正文。纯文本信件由传入的非 HTML 邮件生成，并且 OTRS 配置为不在前端使用富文本功能。

输出过滤器代码示例

请参阅软件包 TemplateModule。

输出过滤器配置示例

请参阅软件包 TemplateModule。

输出过滤器用例示例

在 **AgentTicketZoom** 中显示其它工单属性 这可以通过 FilterElementPost 输出过滤器来实现。

将服务选择显示为多级菜单 对此功能使用 FilterElementPost。可以从处理的模板输出中解析可选服务的列表。可以从服务列表构造多级选择并将其插入到模板内容中。必须使用 FilterElementPost 输出过滤器。

在纯文本信件正文中创建链接 一家生物技术公司在纯文章文章中使用 IPI00217472 等基因名称。可以使用 FilterText 输出过滤器来创建到序列数据库的链接，例如，用于基因名称的 http://srs.ebi.ac.uk/srsbin/cgi-bin/wgetz?-e+{[}IPI-acc:IPI00217472]+-vn+2。

禁止有效内容 防火墙规则禁止所有活动内容。为了避免被防火墙拒绝，可以使用 FilterContent 输出过滤器过滤 HTML 的 <applet> 标签。

注解: 为当前请求所需的每个配置模板构造并运行每个 FilterElementPost 输出过滤器。因此，低性能的输出过滤器或大量的过滤器会严重降低性能。

最佳实践

为了增加灵活性，应在系统配置中配置受影响的模板列表 。

3.3.10 统计模块

有两种不同类型的内部统计模块-动态和静态 。本节描述如何开发这些统计模块 。

动态统计

与静态统计模块不同，动态统计可以通过 OTRS Web 界面进行配置 。本节开发了一个简单的统计模块 。每个动态统计模块都必须实现这些子程序：

- new
- GetObjectName
- GetObjectAttributes
- ExportWrapper
- ImportWrapper

此外，模块必须实现 *GetStatElement*‘ 或 GetStatTable 。如果结果表的标题行应该更改，则必须开发名为 GetHeaderLine 的子程序 。

统计代码示例

在本节中，将显示一个统计模块的样本，并解释每个子程序 。

```
# --
# Copyright (C) 2001-2020 OTRS AG, https://otrs.com/
# --
# This software comes with ABSOLUTELY NO WARRANTY. For details, see
# the enclosed file COPYING for license information (GPL). If you
# did not receive this file, see https://www.gnu.org/licenses/gpl-3.0.txt.
# --

package Kernel::System::Stats::Dynamic::DynamicStatsTemplate;

use strict;
use warnings;

use Kernel::System::Queue;
use Kernel::System::State;
use Kernel::System::Ticket;
```

这是一个常见的样板文件，可以在常见的 OTRS 模块中找到 。类/包名称通过 package 关键字声明 。然后通过关键字 use 使用所需的模块 。

```
sub new {
    my ( $Type, %Param ) = @_;

    # allocate new hash for object
```

(下页继续)

(续上页)

```
    my $Self = {};
    bless( $Self, $Type );

    # check needed objects
    for my $Object (
        qw(DBObject ConfigObject LogObject UserObject TimeObject MainObject␣
↪EncodeObject)
        )
    {
        $Self->{$Object} = $Param{$Object} || die "Got no $Object!";
    }

    # created needed objects
    $Self->{QueueObject}     = Kernel::System::Queue->new( %{$Self} );
    $Self->{TicketObject}    = Kernel::System::Ticket->new( %{$Self} );
    $Self->{StateObject}     = Kernel::System::State->new( %{$Self} );

    return $Self;
}
```

new 是这个统计模块的构造函数。它创建了一个新的类实例。根据编码指南，必须在 new 中创建此模块中所需的其它类的对象。在第 27 至 29 行中，创建了统计模块的对象。第 31 到 37 行检查是否传递了此代码中所需的对象 - 用于创建其它对象或在此模块中。之后，创建其它对象。

```
sub GetObjectName {
    my ( $Self, %Param ) = @_;

    return 'Sample Statistics';
}
```

GetObjectName 返回统计模块的名称。这是在配置的下拉列表中以及现有统计信息列表(对象列)中显示的标签。

```
sub GetObjectAttributes {
    my ( $Self, %Param ) = @_;

    # get state list
    my %StateList = $Self->{StateObject}->StateList(
        UserID => 1,
    );

    # get queue list
    my %QueueList = $Self->{QueueObject}->GetAllQueues();

    # get current time to fix bug#3830
    my $TimeStamp = $Self->{TimeObject}->CurrentTimestamp();
    my ($Date) = split /\s+/, $TimeStamp;
    my $Today = sprintf "%s 23:59:59", $Date;

    my @ObjectAttributes = (
        {
            Name              => 'State',
```

(下页继续)

(续上页)

```
            UseAsXvalue      => 1,
            UseAsValueSeries => 1,
            UseAsRestriction => 1,
            Element          => 'StateIDs',
            Block            => 'MultiSelectField',
            Values           => \%StateList,
        },
        {
            Name             => 'Created in Queue',
            UseAsXvalue      => 1,
            UseAsValueSeries => 1,
            UseAsRestriction => 1,
            Element          => 'CreatedQueueIDs',
            Block            => 'MultiSelectField',
            Translation      => 0,
            Values           => \%QueueList,
        },
        {
            Name             => 'Create Time',
            UseAsXvalue      => 1,
            UseAsValueSeries => 1,
            UseAsRestriction => 1,
            Element          => 'CreateTime',
            TimePeriodFormat => 'DateInputFormat',     # 'DateInputFormatLong',
            Block            => 'Time',
            TimeStop         => $Today,
            Values           => {
                TimeStart => 'TicketCreateTimeNewerDate',
                TimeStop  => 'TicketCreateTimeOlderDate',
            },
        },
    );

    return @ObjectAttributes;
}
```

在此示例统计模块中，我们希望提供用户可以选择的三个属性：队列列表 、状态列表和时间下拉列表 。要获得下拉列表中显示的值，需要执行一些操作 。在本例中是调用 `StateList` 和 `GetAllQueues` 。

然后创建属性列表 。每个属性都通过哈希引用定义 。您可以使用以下键：

Name Web 界面中的标签 。

UseAsXvalue 该属性可以在 X 轴上使用 。

UseAsValueSeries 该属性可以在 Y 轴上使用 。

UseAsRestriction 此属性可用于限制 。

Element HTML 字段名 。

Block 模板文件中的块名称（例如：`<OTRS_HOME>/Kernel/Output/HTML/Standard/AgentStatsEditXaxis.tt` ）。

Values 属性中显示的值 。

提示：如果您安装此示例并配置带有某些队列的统计信息 - 比如说’队列 A’和’队列 B’ - 那么这些队列是

用户在启动统计信息时显示的唯一队列 。有时需要动态下拉或多选字段 。在这种情况下，您可以在属性的定义中设置 SelectedValues：

```
{
    Name              => 'Created in Queue',
    UseAsXvalue       => 1,
    UseAsValueSeries  => 1,
    UseAsRestriction  => 1,
    Element           => 'CreatedQueueIDs',
    Block             => 'MultiSelectField',
    Translation       => 0,
    Values            => \%QueueList,
    SelectedValues    => [ @SelectedQueues ],
},
```

```
sub GetStatElement {
    my ( $Self, %Param ) = @_;

    # search tickets
    return $Self->{TicketObject}->TicketSearch(
        UserID     => 1,
        Result     => 'COUNT',
        Permission => 'ro',
        Limit      => 100_000_000,
        %Param,
    );
}
```

为结果表中的每个单元格调用 GetStatElement 。所以它应该是一个数值 。在此示例中，它执行简单的工单搜索 。哈希 %Param 包含有关 当前 x 值和 y 值以及任何限制的信息 。因此，对于一个计算使用状态为 处理中 、已创建的 *Misc* 队列的工单的单元格，传递的参数哈希看起来像这样：

```
'CreatedQueueIDs' => [
    '4'
],
'StateIDs' => [
    '2'
]
```

如果应该避免 按单元格的计算，GetStatTable 是另一种选择 。GetStatTable 返回一个行列表，因此返回一个数组引用数组 。这导致与使用 GetStatElement 相同的结果 。

```
sub GetStatTable {
    my ( $Self, %Param ) = @_;

    my @StatData;

    for my $StateName ( keys %{ $Param{TableStructure} } ) {
        my @Row;
        for my $Params ( @{ $Param{TableStructure}->{$StateName} } ) {
            my $Tickets = $Self->{TicketObject}->TicketSearch(
                UserID     => 1,
                Result     => 'COUNT',
```

(下页继续)

(续上页)

```
                Permission => 'ro',
                Limit      => 100_000_000,
                %{$Params},
            );

            push @Row, $Tickets;
        }

        push @StatData, [ $StateName, @Row ];
    }

    return @StatData;
}
```

GetStatTable 获取有关所需统计信息查询的所有信息。传递的参数包含有关属性的信息(Restrictions，用于 x / y 轴的属性)和表结构。表结构是一个哈希引用，其中键是 y 轴的值，它们的值是哈希引用，其参数用于 GetStatElement 子程序。

```
'Restrictions' => {},
'TableStructure' => {
    'closed successful' => [
        {
            'CreatedQueueIDs' => [
                '3'
            ],
            'StateIDs' => [
                '2'
            ]
        },
    ],
    'closed unsuccessful' => [
        {
            'CreatedQueueIDs' => [
                '3'
            ],
            'StateIDs' => [
                '3'
            ]
        },
    ],
},
'ValueSeries' => [
    {
        'Block' => 'MultiSelectField',
        'Element' => 'StateIDs',
        'Name' => 'State',
        'SelectedValues' => [
            '5',
            '3',
            '2',
            '1',
            '4'
```

(下页继续)

(续上页)

```
        ],
        'Translation' => 1,
        'Values' => {
            '1' => 'new',
            '10' => 'closed with workaround',
            '2' => 'closed successful',
            '3' => 'closed unsuccessful',
            '4' => 'open',
            '5' => 'removed',
            '6' => 'pending reminder',
            '7' => 'pending auto close+',
            '8' => 'pending auto close-',
            '9' => 'merged'
        }
    }
],
'XValue' => {
    'Block' => 'MultiSelectField',
    'Element' => 'CreatedQueueIDs',
    'Name' => 'Created in Queue',
    'SelectedValues' => [
        '3',
        '4',
        '1',
        '2'
    ],
    'Translation' => 0,
    'Values' => {
        '1' => 'Postmaster',
        '2' => 'Raw',
        '3' => 'Junk',
        '4' => 'Misc'
    }
}
```

有时候必须更改表格的标题。在这种情况下，必须实现一个名为 `GetHeaderLine` 的子程序。该子程序必须返回一个数组引用，并将列标题作为元素。它获取有关传递的 x 值的信息。

```
sub GetHeaderLine {
    my ( $Self, %Param ) = @_;

    my @HeaderLine = ('');
    for my $SelectedXValue ( @{ $Param{XValue}->{SelectedValues} } ) {
        push @HeaderLine, $Param{XValue}->{Values}->{$SelectedXValue};
    }

    return \@HeaderLine;
}
```

```
sub ExportWrapper {
    my ( $Self, %Param ) = @_;

```

(下页继续)

(续上页)

```
    # wrap ids to used spelling
    for my $Use (qw(UseAsValueSeries UseAsRestriction UseAsXvalue)) {
        ELEMENT:
        for my $Element ( @{ $Param{$Use} } ) {
            next ELEMENT if !$Element || !$Element->{SelectedValues};
            my $ElementName = $Element->{Element};
            my $Values      = $Element->{SelectedValues};

            if ( $ElementName eq 'QueueIDs' || $ElementName eq
↪'CreatedQueueIDs' ) {
                ID:
                for my $ID ( @{$Values} ) {
                    next ID if !$ID;
                    $ID->{Content} = $Self->{QueueObject}->QueueLookup(␣
↪QueueID => $ID->{Content} );
                }
            }
            elsif ( $ElementName eq 'StateIDs' || $ElementName eq
↪'CreatedStateIDs' ) {
                my %StateList = $Self->{StateObject}->StateList( UserID => 1␣
↪);
                ID:
                for my $ID ( @{$Values} ) {
                    next ID if !$ID;
                    $ID->{Content} = $StateList{ $ID->{Content} };
                }
            }
        }
    }
    return \%Param;
}
```

配置的统计信息可以导出为 XML 格式。但是由于具有相同队列名称的队列在不同的 OTRS 实例上可以具有不同的 ID，因此导出 ID 将非常痛苦（ 统计数据将计算错误的数字 ）。因此，应编写导出封装器以使用名称而不是 ID。应该对统计模块的每个 维度（ x 轴 、y 轴和限制 ）进行此操作 。

ImportWrapper 以相反的方式工作 - 它将名称转换为导入配置的实例中的 ID 。

这是一个示例导出：

```
<?xml version="1.0" encoding="utf-8"?>

<otrs_stats>
<Cache>0</Cache>
<Description>Sample stats module</Description>
<File></File>
<Format>CSV</Format>
<Format>Print</Format>
<Object>DeveloperManualSample</Object>
<ObjectModule>Kernel::System::Stats::Dynamic::DynamicStatsTemplate</
↪ObjectModule>
<ObjectName>Sample Statistics</ObjectName>
<Permission>stats</Permission>
```

(下页继续)

(续上页)

```
<StatType>dynamic</StatType>
<SumCol>0</SumCol>
<SumRow>0</SumRow>
<Title>Sample 1</Title>
<UseAsValueSeries Element="StateIDs" Fixed="1">
<SelectedValues>removed</SelectedValues>
<SelectedValues>closed unsuccessful</SelectedValues>
<SelectedValues>closed successful</SelectedValues>
<SelectedValues>new</SelectedValues>
<SelectedValues>open</SelectedValues>
</UseAsValueSeries>
<UseAsXvalue Element="CreatedQueueIDs" Fixed="1">
<SelectedValues>Junk</SelectedValues>
<SelectedValues>Misc</SelectedValues>
<SelectedValues>Postmaster</SelectedValues>
<SelectedValues>Raw</SelectedValues>
</UseAsXvalue>
<Valid>1</Valid>
</otrs_stats>
```

现在，解释了所有子程序，这是完整的样本统计模块 。

```
# --
# Copyright (C) 2001-2020 OTRS AG, https://otrs.com/
# --
# This software comes with ABSOLUTELY NO WARRANTY. For details, see
# the enclosed file COPYING for license information (GPL). If you
# did not receive this file, see https://www.gnu.org/licenses/gpl-3.0.txt.
# --

package Kernel::System::Stats::Dynamic::DynamicStatsTemplate;

use strict;
use warnings;

use Kernel::System::Queue;
use Kernel::System::State;
use Kernel::System::Ticket;

sub new {
    my ( $Type, %Param ) = @_;

    # allocate new hash for object
    my $Self = {};
    bless( $Self, $Type );

    # check needed objects
    for my $Object (
        qw(DBObject ConfigObject LogObject UserObject TimeObject MainObject␣
↪EncodeObject)
        )
    {
```

(下页继续)

(续上页)

```
        $Self->{$Object} = $Param{$Object} || die "Got no $Object!";
    }

    # created needed objects
    $Self->{QueueObject}    = Kernel::System::Queue->new( %{$Self} );
    $Self->{TicketObject}   = Kernel::System::Ticket->new( %{$Self} );
    $Self->{StateObject}    = Kernel::System::State->new( %{$Self} );

    return $Self;
}

sub GetObjectName {
    my ( $Self, %Param ) = @_;

    return 'Sample Statistics';
}

sub GetObjectAttributes {
    my ( $Self, %Param ) = @_;

    # get state list
    my %StateList = $Self->{StateObject}->StateList(
        UserID => 1,
    );

    # get queue list
    my %QueueList = $Self->{QueueObject}->GetAllQueues();

    # get current time to fix bug#3830
    my $TimeStamp = $Self->{TimeObject}->CurrentTimestamp();
    my ($Date) = split /\s+/, $TimeStamp;
    my $Today = sprintf "%s 23:59:59", $Date;

    my @ObjectAttributes = (
        {
            Name             => 'State',
            UseAsXvalue      => 1,
            UseAsValueSeries => 1,
            UseAsRestriction => 1,
            Element          => 'StateIDs',
            Block            => 'MultiSelectField',
            Values           => \%StateList,
        },
        {
            Name             => 'Created in Queue',
            UseAsXvalue      => 1,
            UseAsValueSeries => 1,
            UseAsRestriction => 1,
            Element          => 'CreatedQueueIDs',
            Block            => 'MultiSelectField',
            Translation      => 0,
            Values           => \%QueueList,
```

(下页继续)

(续上页)

```
        },
        {
            Name             => 'Create Time',
            UseAsXvalue      => 1,
            UseAsValueSeries => 1,
            UseAsRestriction => 1,
            Element          => 'CreateTime',
            TimePeriodFormat => 'DateInputFormat',     # 'DateInputFormatLong',
            Block            => 'Time',
            TimeStop         => $Today,
            Values           => {
                TimeStart => 'TicketCreateTimeNewerDate',
                TimeStop  => 'TicketCreateTimeOlderDate',
            },
        },
    );

    return @ObjectAttributes;
}

sub GetStatElement {
    my ( $Self, %Param ) = @_;

    # search tickets
    return $Self->{TicketObject}->TicketSearch(
        UserID     => 1,
        Result     => 'COUNT',
        Permission => 'ro',
        Limit      => 100_000_000,
        %Param,
    );
}

sub ExportWrapper {
    my ( $Self, %Param ) = @_;

    # wrap ids to used spelling
    for my $Use (qw(UseAsValueSeries UseAsRestriction UseAsXvalue)) {
        ELEMENT:
        for my $Element ( @{ $Param{$Use} } ) {
            next ELEMENT if !$Element || !$Element->{SelectedValues};
            my $ElementName = $Element->{Element};
            my $Values      = $Element->{SelectedValues};

            if ( $ElementName eq 'QueueIDs' || $ElementName eq
↪'CreatedQueueIDs' ) {
                ID:
                for my $ID ( @{$Values} ) {
                    next ID if !$ID;
                    $ID->{Content} = $Self->{QueueObject}->QueueLookup(␣
↪QueueID => $ID->{Content} );
                }
```

(下页继续)

(续上页)

```
            }
            elsif ( $ElementName eq 'StateIDs' || $ElementName eq
↪'CreatedStateIDs' ) {
                my %StateList = $Self->{StateObject}->StateList( UserID => 1␣
↪);
                ID:
                for my $ID ( @{$Values} ) {
                    next ID if !$ID;
                    $ID->{Content} = $StateList{ $ID->{Content} };
                }
            }
        }
    }
    return \%Param;
}

sub ImportWrapper {
    my ( $Self, %Param ) = @_;

    # wrap used spelling to ids
    for my $Use (qw(UseAsValueSeries UseAsRestriction UseAsXvalue)) {
        ELEMENT:
        for my $Element ( @{ $Param{$Use} } ) {
            next ELEMENT if !$Element || !$Element->{SelectedValues};
            my $ElementName = $Element->{Element};
            my $Values      = $Element->{SelectedValues};

            if ( $ElementName eq 'QueueIDs' || $ElementName eq
↪'CreatedQueueIDs' ) {
                ID:
                for my $ID ( @{$Values} ) {
                    next ID if !$ID;
                    if ( $Self->{QueueObject}->QueueLookup( Queue => $ID->
↪{Content} ) ) {
                        $ID->{Content}
                            = $Self->{QueueObject}->QueueLookup( Queue => $ID-
↪>{Content} );
                    }
                    else {
                        $Self->{LogObject}->Log(
                            Priority => 'error',
                            Message  => "Import: Can' find the queue $ID->
↪{Content}!"
                        );
                        $ID = undef;
                    }
                }
            }
            elsif ( $ElementName eq 'StateIDs' || $ElementName eq
↪'CreatedStateIDs' ) {
                ID:
                for my $ID ( @{$Values} ) {
```

(下页继续)

(续上页)

```
                    next ID if !$ID;

                    my %State = $Self->{StateObject}->StateGet(
                        Name  => $ID->{Content},
                        Cache => 1,
                    );
                    if ( $State{ID} ) {
                        $ID->{Content} = $State{ID};
                    }
                    else {
                        $Self->{LogObject}->Log(
                            Priority => 'error',
                            Message  => "Import: Can' find state $ID->{Content}
↪!"
                        );
                        $ID = undef;
                    }
                }
            }
        }
    }
    return \%Param;
}

1;
```

统计配置示例

```
<?xml version="1.0" encoding="utf-8" ?>
<otrs_config version="1.0" init="Config">
    <ConfigItem Name="Stats::DynamicObjectRegistration###DynamicStatsTemplate"␣
↪Required="0" Valid="1">
        <Description Translatable="1">Here you can decide if the common stats␣
↪module may generate stats about the number of default tickets a requester␣
↪created.</Description>
        <Group>Framework</Group>
        <SubGroup>Core::Stats</SubGroup>
        <Setting>
            <Hash>
                <Item Key="Module">
↪Kernel::System::Stats::Dynamic::DynamicStatsTemplate</Item>
            </Hash>
        </Setting>
    </ConfigItem>
</otrs_config>
```

注解: 如果结果表中有很多单元格并且 GetStatElement 非常复杂，那么请求可能需要很长时间 。

静态统计

后续段落描述了静态统计数据 。静态统计信息很容易创建，因为这些模块只需要实现三个子程序 。

- new
- Param
- Run

静态统计代码示例

以下段落描述了静态统计信息中所需的子程序 。

```
sub new {
    my ( $Type, %Param ) = @_;

    # allocate new hash for object
    my $Self = {%Param};
    bless( $Self, $Type );

    # check all needed objects
    for my $Needed (
        qw(DBObject ConfigObject LogObject
        TimeObject MainObject EncodeObject)
        )
    {
        $Self->{$Needed} = $Param{$Needed} || die "Got no $Needed";
    }

    # create needed objects
    $Self->{TypeObject}   = Kernel::System::Type->new( %{$Self} );
    $Self->{TicketObject} = Kernel::System::Ticket->new( %{$Self} );
    $Self->{QueueObject}  = Kernel::System::Queue->new( %{$Self} );

    return $Self;
}
```

new 创建静态统计类的新实例 。首先，它创建一个新对象，然后检查所需的对象 。

```
sub Param {
    my $Self = shift;

    my %Queues = $Self->{QueueObject}->GetAllQueues();
    my %Types  = $Self->{TypeObject}->TypeList(
        Valid => 1,
    );

    my @Params = (
        {
            Frontend  => 'Type',
            Name      => 'TypeIDs',
            Multiple  => 1,
            Size      => 3,
```

(下页继续)

(续上页)

```
            Data       => \%Types,
        },
        {
            Frontend   => 'Queue',
            Name       => 'QueueIDs',
            Multiple   => 1,
            Size       => 3,
            Data       => \%Queues,
        },
    );

    return @Params;
}
```

Param 方法提供了可以选择创建静态统计的所有参数/属性的列表。它传递了一些参数：请求中提供的统计属性的值，统计的格式和对象的名称（ 模块的名称 ）。

参数/属性必须是具有这些键值对的哈希引用：

Frontend Web 界面中的标签。

Name HTML 字段名。

Data 属性中显示的值。

可以使用 LayoutObject 的 BuildSelection 方法的其他参数，比如在此示例模块中使用了 Size 和 Multiple。

```
sub Run {
    my ( $Self, %Param ) = @_;

    # check needed stuff
    for my $Needed (qw(TypeIDs QueueIDs)) {
        if ( !$Param{$Needed} ) {
            $Self->{LogObject}->Log(
                Priority => 'error',
                Message  => "Need $Needed!",
            );
            return;
        }
    }

    # set report title
    my $Title = 'Tickets per Queue';

    # table headlines
    my @HeadData = (
        'Ticket Number',
        'Queue',
        'Type',
    );

    my @Data;
    my @TicketIDs = $Self->{TicketObject}->TicketSearch(
```

(下页继续)

(续上页)

```
        UserID     => 1,
        Result     => 'ARRAY',
        Permission => 'ro',
        %Param,
    );

    for my $TicketID ( @TicketIDs ) {
        my %Ticket = $Self->{TicketObject}->TicketGet(
            UserID => 1,
            TicketID => $TicketID,
        );
        push @Data, [ $Ticket{TicketNumber}, $Ticket{Queue}, $Ticket{Type} ];
    }

    return ( [$Title], [@HeadData], @Data );
}
```

Run 方法实际上生成了统计数据的表数据。它获取传递此统计数据的属性。在这个示例中的 %Param 存在一个 TypeIDs 键和一个 QueueIDs 键（参见 Param 方法中的属性），它们的值是数组引用。返回的数据由三部分组成：两个数组引用和一个数组。在第一个数组引用中存储统计信息的标题，第二个数组引用包含表中列的标题。然后表体的数据如下。

```
# --
# Copyright (C) 2001-2020 OTRS AG, https://otrs.com/
# --
# This software comes with ABSOLUTELY NO WARRANTY. For details, see
# the enclosed file COPYING for license information (GPL). If you
# did not receive this file, see https://www.gnu.org/licenses/gpl-3.0.txt.
# --

package Kernel::System::Stats::Static::StaticStatsTemplate;

use strict;
use warnings;

use Kernel::System::Type;
use Kernel::System::Ticket;
use Kernel::System::Queue;

=head1 NAME

StaticStatsTemplate.pm - the module that creates the stats about tickets in a␣
↪queue

=head1 SYNOPSIS

All functions

=head1 PUBLIC INTERFACE

=over 4
```

(下页继续)

(续上页)

```
=cut

=item new()

create an object

    use Kernel::Config;
    use Kernel::System::Encode;
    use Kernel::System::Log;
    use Kernel::System::Main;
    use Kernel::System::Time;
    use Kernel::System::DB;
    use Kernel::System::Stats::Static::StaticStatsTemplate;

    my $ConfigObject = Kernel::Config->new();
    my $EncodeObject = Kernel::System::Encode->new(
        ConfigObject => $ConfigObject,
    );
    my $LogObject    = Kernel::System::Log->new(
        ConfigObject => $ConfigObject,
    );
    my $MainObject = Kernel::System::Main->new(
        ConfigObject => $ConfigObject,
        LogObject    => $LogObject,
    );
    my $TimeObject = Kernel::System::Time->new(
        ConfigObject => $ConfigObject,
        LogObject    => $LogObject,
    );
    my $DBObject = Kernel::System::DB->new(
        ConfigObject => $ConfigObject,
        LogObject    => $LogObject,
        MainObject   => $MainObject,
    );
    my $StatsObject = Kernel::System::Stats::Static::StaticStatsTemplate->new(
        ConfigObject => $ConfigObject,
        LogObject    => $LogObject,
        MainObject   => $MainObject,
        TimeObject   => $TimeObject,
        DBObject     => $DBObject,
        EncodeObject => $EncodeObject,
    );

=cut

sub new {
    my ( $Type, %Param ) = @_;

    # allocate new hash for object
    my $Self = {%Param};
    bless( $Self, $Type );
```

(下页继续)

(续上页)

```
    # check all needed objects
    for my $Needed (
        qw(DBObject ConfigObject LogObject
        TimeObject MainObject EncodeObject)
        )
    {
        $Self->{$Needed} = $Param{$Needed} || die "Got no $Needed";
    }

    # create needed objects
    $Self->{TypeObject}   = Kernel::System::Type->new( %{$Self} );
    $Self->{TicketObject} = Kernel::System::Ticket->new( %{$Self} );
    $Self->{QueueObject}  = Kernel::System::Queue->new( %{$Self} );

    return $Self;
}

=item Param()

Get all parameters a user can specify.

    my @Params = $StatsObject->Param();

=cut

sub Param {
    my $Self = shift;

    my %Queues = $Self->{QueueObject}->GetAllQueues();
    my %Types  = $Self->{TypeObject}->TypeList(
        Valid => 1,
    );

    my @Params = (
        {
            Frontend  => 'Type',
            Name      => 'TypeIDs',
            Multiple  => 1,
            Size      => 3,
            Data      => \%Types,
        },
        {
            Frontend  => 'Queue',
            Name      => 'QueueIDs',
            Multiple  => 1,
            Size      => 3,
            Data      => \%Queues,
        },
    );

    return @Params;
}
```

(下页继续)

(续上页)

```
=item Run()

generate the statistic.

    my $StatsInfo = $StatsObject->Run(
        TypeIDs  => [
            1, 2, 4
        ],
        QueueIDs => [
            3, 4, 6
        ],
    );

=cut

sub Run {
    my ( $Self, %Param ) = @_;

    # check needed stuff
    for my $Needed (qw(TypeIDs QueueIDs)) {
        if ( !$Param{$Needed} ) {
            $Self->{LogObject}->Log(
                Priority => 'error',
                Message  => "Need $Needed!",
            );
            return;
        }
    }

    # set report title
    my $Title = 'Tickets per Queue';

    # table headlines
    my @HeadData = (
        'Ticket Number',
        'Queue',
        'Type',
    );

    my @Data;
    my @TicketIDs = $Self->{TicketObject}->TicketSearch(
        UserID     => 1,
        Result     => 'ARRAY',
        Permission => 'ro',
        %Param,
    );

    for my $TicketID ( @TicketIDs ) {
        my %Ticket = $Self->{TicketObject}->TicketGet(
            UserID => 1,
            TicketID => $TicketID,
```

(下页继续)

(续上页)

```
        );
        push @Data, [ $Ticket{TicketNumber}, $Ticket{Queue}, $Ticket{Type} ];
    }

    return ( [$Title], [@HeadData], @Data );
}

1;
```

静态统计配置示例

无需配置。安装后，该模块可用于为此模块创建统计信息。

3.3.11 工单编号生成器模块

票号生成器用于创建不同的标识符，即新票证的票号。任何创建数字串的方法都是可能的，你应该使用关于结果字符串长度的常识（参考：5-10）。

创建工单编号时，请确保结果以系统配置变量 SystemID 为前缀，以便检测入站电子邮件响应中的工单编号。工单编号生成器模块需要两个函数 TicketCreateNumber() 和 GetTNByString()。

不带参数调用 TicketCreateNumber() 方法，并返回新的工单编号。

使用 String 参数调用 GetTNByString() 方法，其中包含要为工单编号解析的字符串，如果找到则返回工单编号。

工单编号生成器代码示例

请参见源代码的 Kernel/System/Ticket/Number 目录中的文件。

工单编号生成器配置示例

请参见 Kernel/Config/Files/XML/Ticket.xml 中以名称 Ticket::NumberGenerator 开头的设置。

工单编号生成器用例示例

工单编号应遵循特定方案 如果默认模块未提供您要使用的工单编号方案，则需要创建新的工单编号生成器。

注解: 你应该坚持现有工单编号生成器中使用的 GetTNByString() 代码，以防止工单编号解析出现问题。此外，检测 TicketCreateNumber() 中循环的程序应保持不变，以防止重复的工单编号。

3.3.12 工单事件模块

工单事件模块在工单操作发生后立即运行。按照惯例，这些模块位于 Kernel/System/Ticket/Event 目录中。工单事件模块只需要两个函数：new() 和 Run()。Run() 方法至少接收参数 Event、UserID 和 Data。Data 是包含工单数据的哈希引用，如果事件有相关的信件也包含信件数据。

工单事件模块代码示例

请参见源代码的 `Kernel/System/Ticket/Event` 目录中的文件 。

工单事件模块配置示例

请参见 `Kernel/Config/Files/XML/Ticket.xml` 中以名称 `Ticket::EventModulePost###` 开头的设置 。

工单事件模块用例示例

移动操作后解锁工单 这个标准功能已经使用工单事件模块 `Kernel::System::Ticket::Event::ForceUnlock` 实现 。当不需要此功能时，可以通过取消系统配置条目 `Ticket::EventModulePost###910-ForceUnlockOnMove` 设置来关闭它 。

删除工单时执行额外的清理操作 自定义 OTRS 可能会在其他数据库表中保存非标准数据 。删除工单后，需要删除此附加数据 。使用工单事件模块监听 `TicketDelete` 事件可以实现此功能 。

新的工单通过 **twitter** 发布 监听 `TicketCreate` 的工单事件模块可以发送推文 。

工单和信件事件

可用的工单事件：

- `TicketCreate`
- `TicketDelete`
- `TicketTitleUpdate`
- `TicketUnlockTimeoutUpdate`
- `TicketQueueUpdate`
- `TicketTypeUpdate`
- `TicketServiceUpdate`
- `TicketSLAUpdate`
- `TicketCustomerUpdate`
- `TicketPendingTimeUpdate`
- `TicketLockUpdate`
- `TicketArchiveFlagUpdate`
- `TicketStateUpdate`
- `TicketOwnerUpdate`
- `TicketResponsibleUpdate`
- `TicketPriorityUpdate`
- `HistoryAdd`
- `HistoryDelete`
- `TicketAccountTime`
- `TicketMerge`

- TicketSubscribe
- TicketUnsubscribe
- TicketFlagSet
- TicketFlagDelete
- EscalationResponseTimeNotifyBefore
- EscalationUpdateTimeNotifyBefore
- EscalationSolutionTimeNotifyBefore
- EscalationResponseTimeStart
- EscalationUpdateTimeStart
- EscalationSolutionTimeStart
- EscalationResponseTimeStop
- EscalationUpdateTimeStop
- EscalationSolutionTimeStop
- NotificationNewTicket
- NotificationFollowUp
- NotificationLockTimeout
- NotificationOwnerUpdate
- NotificationResponsibleUpdate
- NotificationAddNote
- NotificationMove
- NotificationPendingReminder
- NotificationEscalation
- NotificationEscalationNotifyBefore
- NotificationServiceUpdate

可用的信件事件：

- ArticleCreate
- ArticleUpdate
- ArticleSend
- ArticleBounce
- ArticleAgentNotification
- ArticleCustomerNotification
- ArticleAutoResponse
- ArticleFlagSet
- ArticleFlagDelete
- ArticleAgentNotification
- ArticleCustomerNotification

3.3.13 仪表板模块

仪表板模块以折线图的形式显示统计信息 。

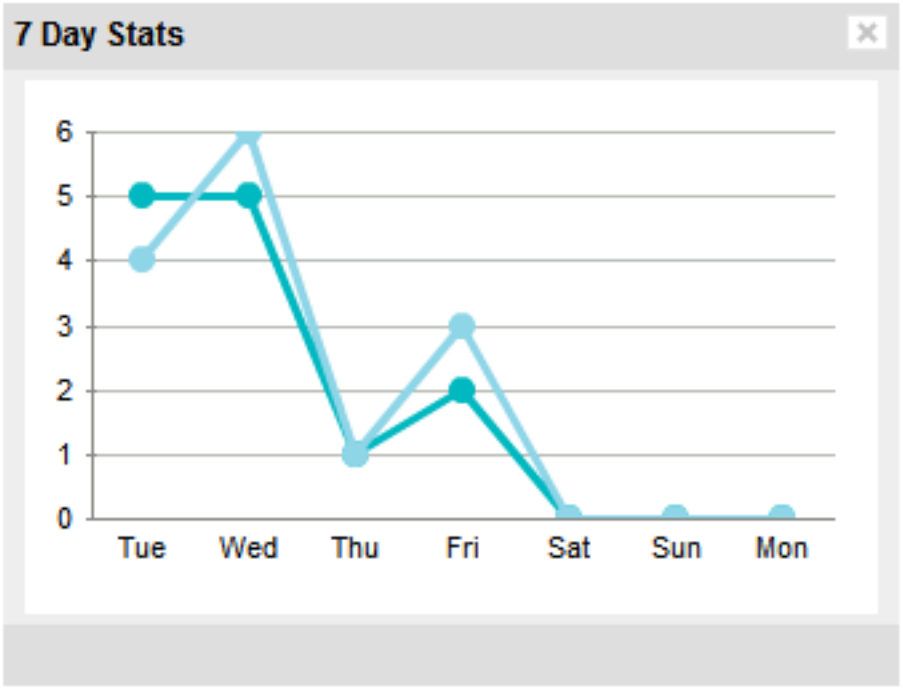

图 3: 仪表板小部件

```
# --
# Copyright (C) 2001-2020 OTRS AG, https://otrs.com/
# --
# This software comes with ABSOLUTELY NO WARRANTY. For details, see
# the enclosed file COPYING for license information (GPL). If you
# did not receive this file, see https://www.gnu.org/licenses/gpl-3.0.txt.
# --

package Kernel::Output::HTML::DashboardTicketStatsGeneric;

use strict;
use warnings;

sub new {
    my ( $Type, %Param ) = @_;

    # allocate new hash for object
    my $Self = {%Param};
    bless( $Self, $Type );

    # get needed objects
    for (
        qw(Config Name ConfigObject LogObject DBObject LayoutObject ParamObject␣
↪TicketObject UserID)
```

(下页继续)

(续上页)

```
        )
    {
        die "Got no $_!" if !$Self->{$_};
    }

    return $Self;
}

sub Preferences {
    my ( $Self, %Param ) = @_;

    return;
}

sub Config {
    my ( $Self, %Param ) = @_;

    my $Key = $Self->{LayoutObject}->{UserLanguage} . '-' . $Self->{Name};
    return (
        %{ $Self->{Config} },
        CacheKey => 'TicketStats' . '-' . $Self->{UserID} . '-' . $Key,
    );

}

sub Run {
    my ( $Self, %Param ) = @_;

    my %Axis = (
        '7Day' => {
            0 => { Day => 'Sun', Created => 0, Closed => 0, },
            1 => { Day => 'Mon', Created => 0, Closed => 0, },
            2 => { Day => 'Tue', Created => 0, Closed => 0, },
            3 => { Day => 'Wed', Created => 0, Closed => 0, },
            4 => { Day => 'Thu', Created => 0, Closed => 0, },
            5 => { Day => 'Fri', Created => 0, Closed => 0, },
            6 => { Day => 'Sat', Created => 0, Closed => 0, },
        },
    );

    my @Data;
    my $Max = 1;
    for my $Key ( 0 .. 6 ) {

        my $TimeNow = $Self->{TimeObject}->SystemTime();
        if ($Key) {
            $TimeNow = $TimeNow - ( 60 * 60 * 24 * $Key );
        }
        my ( $Sec, $Min, $Hour, $Day, $Month, $Year, $WeekDay )
            = $Self->{TimeObject}->SystemTime2Date(
            SystemTime => $TimeNow,
            );
```

(下页继续)

(续上页)

```
        $Data[$Key]->{Day} = $Self->{LayoutObject}->{LanguageObject}->Get(
            $Axis{'7Day'}->{$WeekDay}->{Day}
        );

        my $CountCreated = $Self->{TicketObject}->TicketSearch(

            # cache search result 20 min
            CacheTTL => 60 * 20,

            # tickets with create time after ... (ticket newer than this␣
↪date) (optional)
            TicketCreateTimeNewerDate => "$Year-$Month-$Day 00:00:00",

            # tickets with created time before ... (ticket older than this␣
↪date) (optional)
            TicketCreateTimeOlderDate => "$Year-$Month-$Day 23:59:59",

            CustomerID => $Param{Data}->{UserCustomerID},
            Result     => 'COUNT',

            # search with user permissions
            Permission => $Self->{Config}->{Permission} || 'ro',
            UserID => $Self->{UserID},
        );
        $Data[$Key]->{Created} = $CountCreated;
        if ( $CountCreated > $Max ) {
            $Max = $CountCreated;
        }

        my $CountClosed = $Self->{TicketObject}->TicketSearch(

            # cache search result 20 min
            CacheTTL => 60 * 20,

            # tickets with create time after ... (ticket newer than this␣
↪date) (optional)
            TicketCloseTimeNewerDate => "$Year-$Month-$Day 00:00:00",

            # tickets with created time before ... (ticket older than this␣
↪date) (optional)
            TicketCloseTimeOlderDate => "$Year-$Month-$Day 23:59:59",

            CustomerID => $Param{Data}->{UserCustomerID},
            Result     => 'COUNT',

            # search with user permissions
            Permission => $Self->{Config}->{Permission} || 'ro',
            UserID => $Self->{UserID},
        );
        $Data[$Key]->{Closed} = $CountClosed;
        if ( $CountClosed > $Max ) {
```

(下页继续)

(续上页)

```
            $Max = $CountClosed;
        }
    }

    @Data = reverse @Data;
    my $Source = $Self->{LayoutObject}->JSONEncode(
        Data => \@Data,
    );

    my $Content = $Self->{LayoutObject}->Output(
        TemplateFile => 'AgentDashboardTicketStats',
        Data         => {
            %{ $Self->{Config} },
            Key    => int rand 99999,
            Max    => $Max,
            Source => $Source,
        },
    );

    return $Content;
}

1;
```

要使用此模块，请将以下内容添加到 `Kernel/Config.pm` 并重新启动 Web 服务器（ 如果使用 `mod_perl` ）。

```
<ConfigItem Name="DashboardBackend###0250-TicketStats" Required="0" Valid="1">
    <Description Translatable="1">Parameters for the dashboard backend. "Group
↪" are used to restricted access to the plugin (e. g. Group: admin;group1;
↪group2;). "Default" means if the plugin is enabled per default or if the␣
↪user needs to enable it manually. "CacheTTL" means the cache time in␣
↪minutes for the plugin.</Description>
    <Group>Ticket</Group>
    <SubGroup>Frontend::Agent::Dashboard</SubGroup>
    <Setting>
        <Hash>
            <Item Key="Module">
↪Kernel::Output::HTML::DashboardTicketStatsGeneric</Item>
            <Item Key="Title">7 Day Stats</Item>
            <Item Key="Created">1</Item>
            <Item Key="Closed">1</Item>
            <Item Key="Permission">rw</Item>
            <Item Key="Block">ContentSmall</Item>
            <Item Key="Group"></Item>
            <Item Key="Default">1</Item>
            <Item Key="CacheTTL">45</Item>
        </Hash>
    </Setting>
</ConfigItem>
```

注解: 天数或个别的折线过多可能会导致性能下降 。

3.3.14 通知模块

通知模块用于在主导航下方显示通知。您可以编写和注册自己的通知模块。OTRS 框架中目前有 5 个工单菜单。

- AgentOnline
- AgentTicketEscalation
- CharsetCheck
- CustomerOnline
- UIDCheck

通知模块代码示例

通知模块位于 Kernel/Output/HTML/TicketNotification*.pm。下面是一个通知模块的示例。将它保存为 Kernel/Output/HTML/TicketNotificationCustom.pm。你只需要 2 个函数：new() 和 Run()。

```
# --
# Copyright (C) 2001-2020 OTRS AG, https://otrs.com/
# --
# This software comes with ABSOLUTELY NO WARRANTY. For details, see
# the enclosed file COPYING for license information (GPL). If you
# did not receive this file, see https://www.gnu.org/licenses/gpl-3.0.txt.
# --

package Kernel::Output::HTML::NotificationCustom;

use strict;
use warnings;

use Kernel::System::Custom;

sub new {
    my ( $Type, %Param ) = @_;

    # allocate new hash for object
    my $Self = {};
    bless( $Self, $Type );

    # get needed objects
    for my $Object (qw(ConfigObject LogObject DBObject LayoutObject TimeObject␣
↪UserID)) {
        $Self->{$Object} = $Param{$Object} || die "Got no $Object!";
    }
    $Self->{CustomObject} = Kernel::System::Custom->new(%Param);
    return $Self;
}

sub Run {
    my ( $Self, %Param ) = @_;
```

(下页继续)

(续上页)

```
    # get session info
    my %CustomParam      = ();
    my @Customs     = $Self->{CustomObject}->GetAllCustomIDs();
    my $IdleMinutes = $Param{Config}->{IdleMinutes} || 60 * 2;
    for (@Customs) {
        my %Data = $Self->{CustomObject}->GetCustomIDData( CustomID => $_, );
        if (
            $Self->{UserID} ne $Data{UserID}
            && $Data{UserType} eq 'User'
            && $Data{UserLastRequest}
            && $Data{UserLastRequest} + ( $IdleMinutes * 60 ) > $Self->
↪{TimeObject}->SystemTime()
            && $Data{UserFirstname}
            && $Data{UserLastname}
            )
        {
            $CustomParam{ $Data{UserID} } = "$Data{UserFirstname} $Data
↪{UserLastname}";
            if ( $Param{Config}->{ShowEmail} ) {
                $CustomParam{ $Data{UserID} } .= " ($Data{UserEmail})";
            }
        }
    }
    for ( sort { $CustomParam{$a} cmp $CustomParam{$b} } keys %CustomParam ) {
        if ( $Param{Message} ) {
            $Param{Message} .= ', ';
        }
        $Param{Message} .= "$CustomParam{$_}";
    }
    if ( $Param{Message} ) {
        return $Self->{LayoutObject}->Notify( Info => 'Custom Message: %s', "
↪' . $Param{Message} );
    }
    else {
        return '';
    }
}

1;
```

通知模块配置示例

需要激活自定义通知模块。这可以使用下面的 XML 配置来完成。通知模块的配置哈希中可能还有其它参数。

```
<ConfigItem Name="Frontend::NotifyModule###3-Custom" Required="0" Valid="0">
    <Description Translatable="1">Module to show custom message in the agent␣
↪interface.</Description>
    <Group>Framework</Group>
    <SubGroup>Frontend::Agent::ModuleNotify</SubGroup>
    <Setting>
```

(下页继续)

(续上页)

```
        <Hash>
            <Item Key="Module">Kernel::Output::HTML::NotificationCustom</Item>
            <Item Key="Key1">1</Item>
            <Item Key="Key2">2</Item>
        </Hash>
    </Setting>
</ConfigItem>
```

通知模块用例示例

如果已设置参数(例如 FreeTextField)，则有用的工单菜单实现可以是指向外部工具的链接 。

3.3.15 工单菜单模块

工单菜单模块用于在工单上方的菜单中显示其他链接 。您可以编写并注册自己的工单菜单模块 。OTRS 框架附带 4 个工单菜单(通用，锁定，负责人和 关注人)。有关更多信息，请查看 OTRS 管理手册 。

工单菜单模块代码示例

工单菜单模块位于 Kernel/Output/HTML/TicketMenu*.pm 下 。下面是一个工单菜单模块的示例 。将它保存在 Kernel/Output/HTML/TicketMenuCustom.pm 下 。你只需要 2 个函数：new() 和 Run() 。

```
# --
# Copyright (C) 2001-2020 OTRS AG, https://otrs.com/
# --
# This software comes with ABSOLUTELY NO WARRANTY. For details, see
# the enclosed file COPYING for license information (GPL). If you
# did not receive this file, see https://www.gnu.org/licenses/gpl-3.0.txt.
# --

package Kernel::Output::HTML::TicketMenuCustom;

use strict;
use warnings;

sub new {
    my ( $Type, %Param ) = @_;

    # allocate new hash for object
    my $Self = {};
    bless( $Self, $Type );

    # get needed objects
    for my $Object (qw(ConfigObject LogObject DBObject LayoutObject UserID␣
↪TicketObject)) {
        $Self->{$Object} = $Param{$Object} || die "Got no $Object!";
    }

    return $Self;
```

(下页继续)

(续上页)

```
}

sub Run {
    my ( $Self, %Param ) = @_;

    # check needed stuff
    if ( !$Param{Ticket} ) {
        $Self->{LogObject}->Log(
            Priority => 'error',
            Message  => 'Need Ticket!'
        );
        return;
    }

    # check if frontend module registered, if not, do not show action
    if ( $Param{Config}->{Action} ) {
        my $Module = $Self->{ConfigObject}->Get('Frontend::Module')->{ $Param
↪{Config}->{Action} };
        return if !$Module;
    }

    # check permission
    my $AccessOk = $Self->{TicketObject}->Permission(
        Type     => 'rw',
        TicketID => $Param{Ticket}->{TicketID},
        UserID   => $Self->{UserID},
        LogNo    => 1,
    );
    return if !$AccessOk;

    # check permission
    if ( $Self->{TicketObject}->CustomIsTicketCustom( TicketID => $Param
↪{Ticket}->{TicketID} ) ) {
        my $AccessOk = $Self->{TicketObject}->OwnerCheck(
            TicketID => $Param{Ticket}->{TicketID},
            OwnerID  => $Self->{UserID},
        );
        return if !$AccessOk;
    }

    # check acl
    return
        if defined $Param{ACL}->{ $Param{Config}->{Action} }
            && !$Param{ACL}->{ $Param{Config}->{Action} };

    # if ticket is customized
    if ( $Param{Ticket}->{Custom} eq 'lock' ) {

        # if it is locked for somebody else
        return if $Param{Ticket}->{OwnerID} ne $Self->{UserID};

        # show custom action
```

(下页继续)

(续上页)

```
        return {
            %{ $Param{Config} },
            %{ $Param{Ticket} },
            %Param,
            Name        => 'Custom',
            Description => 'Custom to give it back to the queue!',
            Link        => 'Action=AgentTicketCustom;Subaction=Custom;
↪TicketID=$QData{"TicketID"}',
        };
    }

    # if ticket is customized
    return {
        %{ $Param{Config} },
        %{ $Param{Ticket} },
        %Param,
        Name        => 'Custom',
        Description => 'Custom it to work on it!',
        Link        => 'Action=AgentTicketCustom;Subaction=Custom;TicketID=
↪$QData{"TicketID"}',
    };
}

1;
```

工单菜单模块配置示例

需要激活自定义的工单菜单模块。这可以使用下面的 XML 配置来完成。您的工单菜单模块的配置哈希中可能还有其他参数。

```
<ConfigItem Name="Ticket::Frontend::MenuModule###110-Custom" Required="0"␣
↪Valid="1">
    <Description Translatable="1">Module to show custom link in menu.</
↪Description>
    <Group>Ticket</Group>
    <SubGroup>Frontend::Agent::Ticket::MenuModule</SubGroup>
    <Setting>
        <Hash>
            <Item Key="Module">Kernel::Output::HTML::TicketMenuCustom</Item>
            <Item Key="Name">Custom</Item>
            <Item Key="Action">AgentTicketCustom</Item>
        </Hash>
    </Setting>
</ConfigItem>
```

工单菜单模块用例示例

如果已设置参数（ 例如 `FreeTextField` ），则有用的工单菜单实现可以是指向外部工具的链接 。

注解: 工单菜单指向可以处理的 URL 。如果要通过 OTRS 框架处理该请求，则必须编写自己的前端模块 。

3.3.16 网络传输

网络传输用作在 OTRS 和远程系统之间发送和接收信息的方法 。通用接口配置允许 Web 服务为提供程序和请求程序使用不同的网络传输模块，但最常见的情况是两者都使用相同的传输模块 。

OTRS 作为服务提供方 OTRS 使用网络传输模块从远程系统获取数据并执行操作 。执行操作后，OTRS 再次使用网络传输模块将响应发送回远程系统 。

OTRS 作为服务请求方 OTRS 使用网络传输模块向远程系统发送请求以执行远程操作以及所需数据 。OTRS 等待远程系统响应并将其发送回请求者模块 。

在这两种方式中，网络传输模块处理远程系统格式的数据 。建议不要在此模块中进行任何数据转换，因为映射层负责执行通信期间所需的任何数据转换 。例外情况是传输中特别需要的数据转换，例如从或到 Perl 转换的 XML 或 JSON 。

传输后端

接下来我们将展示如何开发新的传输后端 。每个传输后端都必须实现这些子程序：

- `new`
- `ProviderProcessRequest`
- `ProviderGenerateResponse`
- `RequesterPerformRequest`

我们应该实现这些方法中的每一种，以便能够以两种方式与远程系统正确通信 。所有网络传输后端都由传输模块(`Kernel/GenericInterface/Transport.pm`)处理 。

当前通用接口实现 HTTP SOAP 和 HTTP REST 传输 。如果计划的 Web 服务可以使用 HTTP SOAP 或 HTTP REST，则无需创建新的网络传输模块，而是建议您查看 HTTP SOAP 或 HTTP REST 配置以检查其设置以及如何根据远程系统进行调整 。

网络传输代码示例

如果提供的网络传输与 Web 服务需求不匹配，则在本节中展示一个示例网络传输模块，并解释每个子程序 。通常，传输模块使用 CPAN 模块作为后端 。例如，HTTP SOAP 传输模块使用 `SOAP::Lite` 模块作为后端 。

对于此示例，自定义包用于返回数据而不向远程系统发出真实的网络请求，而是将此自定义模块用作环回接口 。

```
# --
# Copyright (C) 2001-2020 OTRS AG, https://otrs.com/
# --
# This software comes with ABSOLUTELY NO WARRANTY. For details, see
# the enclosed file COPYING for license information (GPL). If you
# did not receive this file, see https://www.gnu.org/licenses/gpl-3.0.txt.
# --

package Kernel::GenericInterface::Transport::HTTP::Test;
```

(下页继续)

(续上页)

```
use strict;
use warnings;

use HTTP::Request::Common;
use LWP::UserAgent;
use LWP::Protocol;

# prevent 'Used once' warning for Kernel::OM
use Kernel::System::ObjectManager;

our $ObjectManagerDisabled = 1;
```

这是可在常见 OTRS 模块中找到的通用标头。类/包名称通过 package 关键字声明。传输无法由对象管理器实例化。

```
sub new {
    my ( $Type, %Param ) = @_;

    my $Self = {};
    bless( $Self, $Type );

    for my $Needed (qw( DebuggerObject TransportConfig)) {
        $Self->{$Needed} = $Param{$Needed} || return {
            Success      => 0,
            ErrorMessage => "Got no $Needed!"
        };
    }

    return $Self;
}
```

构造函数 new 创建了一个新的类实例。根据编码指南，只有对象管理器没有处理的其它类的对象才需要在在 new 中创建。

```
sub ProviderProcessRequest {
    my ( $Self, %Param ) = @_;

    if ( $Self->{TransportConfig}->{Config}->{Fail} ) {

        return {
            Success      => 0,
            ErrorMessage => "HTTP status code: 500",
            Data         => {},
        };
    }

    my $ParamObject = $Kernel::OM->Get('Kernel::System::Web::Request');

    my %Result;
    for my $ParamName ( $ParamObject->GetParamNames() ) {
        $Result{$ParamName} = $ParamObject->GetParam( Param => $ParamName );
    }
```

(下页继续)

(续上页)

```
    # special handling for empty post request
    if ( scalar keys %Result == 1 && exists $Result{POSTDATA} && !$Result
↪{POSTDATA} ) {
        %Result = ();
    }

    if ( !%Result ) {

        return $Self->{DebuggerObject}->Error(
            Summary => 'No request data found.',
        );
    }

    return {
        Success   => 1,
        Data      => \%Result,
        Operation => 'test_operation',
    };
}
```

ProviderProcessRequest 函数从远程系统获取请求（ 在本例中与 OTRS 相同 ），并从请求中提取数据和要执行的操作 。对于这个例子，操作总是 test_operation 。

此函数解析请求以获取数据和操作名称的方式，完全取决于要实现的协议和用于的外部模块 。

```
sub ProviderGenerateResponse {
    my ( $Self, %Param ) = @_;

    if ( $Self->{TransportConfig}->{Config}->{Fail} ) {

        return {
            Success      => 0,
            ErrorMessage => 'Test response generation failed',
        };
    }

    my $Response;

    if ( !$Param{Success} ) {
        $Response
            = HTTP::Response->new( 500 => ( $Param{ErrorMessage} || 'Internal␣
↪Server Error' ) );
        $Response->protocol('HTTP/1.0');
        $Response->content_type("text/plain; charset=UTF-8");
        $Response->date(time);
    }
    else {

        # generate a request string from the data
        my $Request
            = HTTP::Request::Common::POST( 'http://testhost.local/', Content␣
↪=> $Param{Data} );
```

(下页继续)

(续上页)

```
        $Response = HTTP::Response->new( 200 => "OK" );
        $Response->protocol('HTTP/1.0');
        $Response->content_type("text/plain; charset=UTF-8");
        $Response->add_content_utf8( $Request->content() );
        $Response->date(time);
    }

    $Self->{DebuggerObject}->Debug(
        Summary => 'Sending HTTP response',
        Data    => $Response->as_string(),
    );

    # now send response to client
    print STDOUT $Response->as_string();

    return {
        Success => 1,
    };
}
```

此函数将响应发送回远程系统以进行请求的操作 。

对于此特定示例，我们返回标准 HTTP 响应成功(200)或不成功(500)，以及每种情况下所需的数据 。

```
sub RequesterPerformRequest {
    my ( $Self, %Param ) = @_;

    if ( $Self->{TransportConfig}->{Config}->{Fail} ) {

        return {
            Success      => 0,
            ErrorMessage => "HTTP status code: 500",
            Data         => {},
        };
    }

    # use custom protocol handler to avoid sending out real network requests
    LWP::Protocol::implementor(
        testhttp =>
↪'Kernel::GenericInterface::Transport::HTTP::Test::CustomHTTPProtocol'
    );
    my $UserAgent = LWP::UserAgent->new();
    my $Response = $UserAgent->post( 'testhttp://localhost.local/', Content =>
↪ $Param{Data} );

    return {
        Success => 1,
        Data    => {
            ResponseContent => $Response->content(),
        },
    };
}
```

这是 OTRS 作为请求者使用的唯一函数。它将请求发送到远程系统并等待其响应。

对于此示例，我们使用自定义协议处理程序来避免将请求发送到真实网络。此自定义协议如下所示。

```
package Kernel::GenericInterface::Transport::HTTP::Test::CustomHTTPProtocol;

use base qw(LWP::Protocol);

sub new {
    my $Class = shift;

    return $Class->SUPER::new(@_);
}

sub request {    ## no critic
    my $Self = shift;

    my ( $Request, $Proxy, $Arg, $Size, $Timeout ) = @_;

    my $Response = HTTP::Response->new( 200 => "OK" );
    $Response->protocol('HTTP/1.0');
    $Response->content_type("text/plain; charset=UTF-8");
    $Response->add_content_utf8( $Request->content() );
    $Response->date(time);

    #print $Request->as_string();
    #print $Response->as_string();

    return $Response;
}
```

这是我们使用的自定义协议的代码。此方法仅适用于培训或测试远程系统不可用的环境。

对于新模块开发，我们不建议使用此方法，需要实现真正的协议。

网络传输配置示例

需要将该网络传输模块注册以便在 OTRS GUI 中访问。这可以使用下面的 XML 配置来完成。

```
<ConfigItem Name="GenericInterface::Transport::Module###HTTP::Test" Required="0
↪" Valid="1">
    <Description Translatable="1">GenericInterface module registration for␣
↪the transport layer.</Description>
    <Group>GenericInterface</Group>
    <SubGroup>GenericInterface::Transport::ModuleRegistration</SubGroup>
    <Setting>
        <Hash>
            <Item Key="Name">Test</Item>
            <Item Key="Protocol">HTTP</Item>
            <Item Key="ConfigDialog">AdminGenericInterfaceTransportHTTPTest</
↪Item>
        </Hash>
    </Setting>
</ConfigItem>
```

3.3.17 映射

映射用于将数据从 OTRS 转换为外部系统，反之亦然 。该数据可以表示为 key => value 对 。可以开发映射模块，不仅可以转换值，还可以转换键 。

例如：

映射自	映射至
Prio => Warning（ 优先级设为警告 ）	PriorityID => 3（ 优先级 ID 设置为 3 ）

映射层不是绝对必要的，Web 服务可以完全跳过它，具体取决于 Web 服务配置以及如何实现调用程序和操作 。但是，如果需要进行某些数据转换，强烈建议使用现有的映射模块或创建新的映射模块 。

在正常通信期间，可以多次调用映射模块，请看下面的示例 。

OTRS 作为提供程序示例

1. 远程系统使用远程系统格式的数据发送请求 。
2. 数据从远程系统格式映射到 OTRS 格式 。
3. OTRS 执行操作并以 OTRS 格式返回响应 。
4. 数据从 OTRS 格式映射到远程系统格式 。
5. 将远程系统格式的数据响应发送到远程系统 。

OTRS 作为请求程序的例子

1. OTRS 使用 OTRS 格式的数据准备对远程系统的请求 。
2. 数据从 OTRS 格式映射到远程系统格式 。
3. 请求被发送到执行操作的远程系统，并使用远程系统格式的数据将响应发送回 OTRS 。
4. 数据以远程系统格式（ 再次 ）映射到 OTRS 格式 。
5. OTRS 处理响应内容 。

映射后端

通用接口提供了一个名为 *Simple* 的映射模块 。使用此模块，可以完成大多数数据转换，包括键和值映射，还定义了处理键和值的默认映射的规则 。

因此，您很可能不需要开发自定义映射模块 。在继续之前，请检查 *Simple* 映射模块（ `Kernel/GenericInterface/Mapping/Simple.pm` ）及其在线文档 。

如果 *Simple* 映射模块不符合您的需求，那么我们将展示如何开发新的映射后端 。每个映射后端都必须实现这些子程序：

- `new`
- `Map`

我们应该实现这些方法中的每一个，以便能够映射通信中的数据，由请求者或提供者处理 。所有映射后端都由映射模块处理（ `Kernel/GenericInterface/Mapping.pm` ）。

映射码示例

在本节中，将显示一个映射模块的样本，并解释每个子程序 。

```
# --
# Copyright (C) 2001-2020 OTRS AG, https://otrs.com/
# --
# This software comes with ABSOLUTELY NO WARRANTY. For details, see
# the enclosed file COPYING for license information (GPL). If you
# did not receive this file, see https://www.gnu.org/licenses/gpl-3.0.txt.
# --

package Kernel::GenericInterface::Mapping::Test;

use strict;
use warnings;

use Kernel::System::VariableCheck qw(IsHashRefWithData IsStringWithData);

our $ObjectManagerDisabled = 1;
```

这是可在常见 OTRS 模块中找到的通用标头。类/包名称通过 `package` 关键字声明。

我们还包括 `VariableCheck` 模块，用于对某些变量执行某些验证。对象管理器无法实例化映射。

```
sub new {
    my ( $Type, %Param ) = @_;

    # allocate new hash for object
    my $Self = {};
    bless( $Self, $Type );

    # check needed params
    for my $Needed (qw(DebuggerObject MappingConfig)) {
        if ( !$Param{$Needed} ) {

            return {
                Success      => 0,
                ErrorMessage => "Got no $Needed!"
            };
        }
        $Self->{$Needed} = $Param{$Needed};
    }

    # check mapping config
    if ( !IsHashRefWithData( $Param{MappingConfig} ) ) {

        return $Self->{DebuggerObject}->Error(
            Summary => 'Got no MappingConfig as hash ref with content!',
        );
    }

    # check config - if we have a map config, it has to be a non-empty hash ref
    if (
        defined $Param{MappingConfig}->{Config}
        && !IsHashRefWithData( $Param{MappingConfig}->{Config} )
        )
```

(下页继续)

(续上页)

```
    {

        return $Self->{DebuggerObject}->Error(
            Summary => 'Got MappingConfig with Data, but Data is no hash ref␣
↪with content!',
        );
    }

    return $Self;
}
```

构造函数 new 创建了一个新的类实例 。根据编码指南，只有对象管理器没有处理的其它类的对象才需要在在 new 中创建 。

```
sub Map {
    my ( $Self, %Param ) = @_;

    # check data - only accept undef or hash ref
    if ( defined $Param{Data} && ref $Param{Data} ne 'HASH' ) {

        return $Self->{DebuggerObject}->Error(
            Summary => 'Got Data but it is not a hash ref in Mapping Test␣
↪backend!'
        );
    }

    # return if data is empty
    if ( !defined $Param{Data} || !%{ $Param{Data} } ) {

        return {
            Success => 1,
            Data    => {},
        };
    }

    # no config means that we just return input data
    if (
        !defined $Self->{MappingConfig}->{Config}
        || !defined $Self->{MappingConfig}->{Config}->{TestOption}
        )
    {

        return {
            Success => 1,
            Data    => $Param{Data},
        };
    }

    # check TestOption format
    if ( !IsStringWithData( $Self->{MappingConfig}->{Config}->{TestOption} ) ) {

        return $Self->{DebuggerObject}->Error(
```

(下页继续)

(续上页)

```
            Summary => 'Got no TestOption as string with value!',
        );
    }

    # parse data according to configuration
    my $ReturnData = {};
    if ( $Self->{MappingConfig}->{Config}->{TestOption} eq 'ToUpper' ) {
        $ReturnData = $Self->_ToUpper( Data => $Param{Data} );
    }
    elsif ( $Self->{MappingConfig}->{Config}->{TestOption} eq 'ToLower' ) {
        $ReturnData = $Self->_ToLower( Data => $Param{Data} );
    }
    elsif ( $Self->{MappingConfig}->{Config}->{TestOption} eq 'Empty' ) {
        $ReturnData = $Self->_Empty( Data => $Param{Data} );
    }
    else {
        $ReturnData = $Param{Data};
    }

    # return result
    return {
        Success => 1,
        Data    => $ReturnData,
    };
}
```

Map 函数是每个映射模块的主要部分。它接收映射配置（规则）和原始格式的数据（OTRS 或远程系统格式）并将其转换为新格式，在映射过程中甚至可以更改数据结构。

在此特定示例中，有三个规则来映射值。此规则在映射配置键 TestOption 中设置，它们是 ToUpper、ToLower 和 Empty。

- ToUpper：将每个数据值转换为大写。
- ToLower：将每个数据值转换为小写。
- Empty：将每个数据值转换为空字符串。

在此示例中，未实现任何数据键转换。

```
sub _ToUpper {
    my ( $Self, %Param ) = @_;

    my $ReturnData = {};
    for my $Key ( sort keys %{ $Param{Data} } ) {
        $ReturnData->{$Key} = uc $Param{Data}->{$Key};
    }

    return $ReturnData;
}

sub _ToLower {
    my ( $Self, %Param ) = @_;
```

(下页继续)

(续上页)

```
    my $ReturnData = {};
    for my $Key ( sort keys %{ $Param{Data} } ) {
        $ReturnData->{$Key} = lc $Param{Data}->{$Key};
    }

    return $ReturnData;
}

sub _Empty {
    my ( $Self, %Param ) = @_;

    my $ReturnData = {};
    for my $Key ( sort keys %{ $Param{Data} } ) {
        $ReturnData->{$Key} = '';
    }

    return $ReturnData;
}
```

这是实际执行字符串转换的辅助函数 。

映射配置示例

需要将此映射模块注册以便在 OTRS GUI 中访问 。这可以使用下面的 XML 配置来完成 。

```
<ConfigItem Name="GenericInterface::Mapping::Module###Test" Required="0" Valid=
↪"1">
    <Description Translatable="1">GenericInterface module registration for␣
↪the mapping layer.</Description>
    <Group>GenericInterface</Group>
    <SubGroup>GenericInterface::Mapping::ModuleRegistration</SubGroup>
    <Setting>
        <Hash>
            <Item Key="ConfigDialog"></Item>
        </Hash>
    </Setting>
</ConfigItem>
```

3.3.18 调用程序

调用程序用于创建从 OTRS 到远程系统的请求 。通用接口的这一部分负责在 OTRS 侧执行必要的任务，以收集必要的数据以构建请求 。

调用程序后端

接下来我们将展示如何开发一个新的调用程序 。每个调用程序都必须实现这些子程序：

- `new`
- `PrepareRequest`

- HandleResponse

我们应该实现这些方法中的每一个，以便能够使用请求处理程序(Kernel/GenericInterface/Requester.pm)执行请求 。

调用程序代码示例

在本节中，将显示一个调用程序模块的样本，并解释每个子程序 。

```
# --
# Copyright (C) 2001-2020 OTRS AG, https://otrs.com/
# --
# This software comes with ABSOLUTELY NO WARRANTY. For details, see
# the enclosed file COPYING for license information (GPL). If you
# did not receive this file, see https://www.gnu.org/licenses/gpl-3.0.txt.
# --

package Kernel::GenericInterface::Invoker::Test::Test;

use strict;
use warnings;

use Kernel::System::VariableCheck qw(IsString IsStringWithData);

# prevent 'Used once' warning for Kernel::OM
use Kernel::System::ObjectManager;

our $ObjectManagerDisabled = 1;
```

这是可在常见 OTRS 模块中找到的通用标头 。类/包名称通过 package 关键字声明 。调用程序无法由对象管理器实例化 。

```
sub new {
    my ( $Type, %Param ) = @_;

    # allocate new hash for object
    my $Self = {};
    bless( $Self, $Type );

    # check needed params
    if ( !$Param{DebuggerObject} ) {
        return {
            Success      => 0,
            ErrorMessage => "Got no DebuggerObject!"
        };
    }

    $Self->{DebuggerObject} = $Param{DebuggerObject};

    return $Self;
}
```

构造函数 new 创建了一个新的类实例 。根据编码指南，只有对象管理器没有处理的其它类的对象才需要在在 new 中创建 。

```
sub PrepareRequest {
    my ( $Self, %Param ) = @_;

    # we need a TicketNumber
    if ( !IsStringWithData( $Param{Data}->{TicketNumber} ) ) {
        return $Self->{DebuggerObject}->Error( Summary => 'Got no TicketNumber
↪' );
    }

    my %ReturnData;

    $ReturnData{TicketNumber} = $Param{Data}->{TicketNumber};

    # check Action
    if ( IsStringWithData( $Param{Data}->{Action} ) ) {
        $ReturnData{Action} = $Param{Data}->{Action} . 'Test';
    }

    # check request for system time
    if ( IsStringWithData( $Param{Data}->{GetSystemTime} ) && $Param{Data}->
↪{GetSystemTime} ) {
        $ReturnData{SystemTime} = $Kernel::OM->Get('Kernel::System::Time')->
↪SystemTime();
    }

    return {
        Success => 1,
        Data    => \%ReturnData,
    };
}
```

`PrepareRequest` 函数用于处理和收集要发送到请求中的所有需要的数据。在这里，我们可以从请求处理程序接收数据，使用它、扩展它、生成新数据并随后将结果传输到映射层。

对于此示例，我们希望收到一个工单编号。如果没有那么我们使用调试器方法 `Error()` 在调试日志中创建一个条目，并返回一个参数 `Success` 为 0 的结构和一个错误消息作为传递 `Summary`。

此示例还将单词 *Test* 附加到 `Action` 参数，如果请求 `GetSystemTime`，它将使用当前系统时间填充 `SystemTime` 参数。这部分代码是准备要发送的数据。在一个真正的调用程序上，应该在这里调用核心模块 (`Kernel/System/*.pm`)。

如果在 `PrepareRequest` 函数的任何部分期间需要停止请求而不生成并在调试日志中输入错误，则可以使用以下代码：

```
# stop requester communication
return {
    Success           => 1,
    StopCommunication => 1,
};
```

使用此方法，请求者将理解请求不应继续（它不会被发送到映射层，也不会被发送到网络传输）。请求者不会在调试日志上发送错误，它只会静默停止。

```
sub HandleResponse {
```

(下页继续)

(续上页)

```
    my ( $Self, %Param ) = @_;

    # if there was an error in the response, forward it
    if ( !$Param{ResponseSuccess} ) {
        if ( !IsStringWithData( $Param{ResponseErrorMessage} ) ) {

            return $Self->{DebuggerObject}->Error(
                Summary => 'Got response error, but no response error message!
↪',
            );
        }

        return {
            Success      => 0,
            ErrorMessage => $Param{ResponseErrorMessage},
        };
    }

    # we need a TicketNumber
    if ( !IsStringWithData( $Param{Data}->{TicketNumber} ) ) {

        return $Self->{DebuggerObject}->Error( Summary => 'Got no␣
↪TicketNumber!' );
    }

    # prepare TicketNumber
    my %ReturnData = (
        TicketNumber => $Param{Data}->{TicketNumber},
    );

    # check Action
    if ( IsStringWithData( $Param{Data}->{Action} ) ) {
        if ( $Param{Data}->{Action} !~ m{ \A ( .*? ) Test \z }xms ) {

            return $Self->{DebuggerObject}->Error(
                Summary => 'Got Action but it is not in required format!',
            );
        }
        $ReturnData{Action} = $1;
    }

    return {
        Success => 1,
        Data    => \%ReturnData,
    };
}
```

`HandleResponse` 函数用于接收和处理来自先前请求的数据，该数据是对远程系统进行的 。此数据已由映射层传递，以将其从远程系统格式转换为 OTRS 格式(如果需要)。

对于这个特定的例子，它再次检查工单编号并检查操作是否以单词 *Test* 结束(如在 `PrepareRequest` 函数中所做的那样)。

注解：这个调用程序仅用于测试，真正的调用程序将检查响应是否采用远程系统描述的格式，并且可以执行以下操作：调用另一个调用者 、执行对核心模块的调用 、更新数据库 、发送一个错误等 。

调用程序配置示例

需要注册此调用程序模块才能在 OTRS GUI 中访问 。这可以使用下面的 XML 配置来完成 。

```
<ConfigItem Name="GenericInterface::Invoker::Module###Test::Test" Required="0"␣
↪Valid="1">
    <Description Translatable="1">GenericInterface module registration for␣
↪the invoker layer.</Description>
    <Group>GenericInterface</Group>
    <SubGroup>GenericInterface::Invoker::ModuleRegistration</SubGroup>
    <Setting>
        <Hash>
            <Item Key="Name">Test</Item>
            <Item Key="Controller">Test</Item>
            <Item Key="ConfigDialog">AdminGenericInterfaceInvokerDefault</Item>
        </Hash>
    </Setting>
</ConfigItem>
```

3.3.19 操作

操作（ operation ）用于在 OTRS 内执行一个动作（ action ）。此动作由外部系统请求，并且可以包含特殊参数以便正确执行动作 。执行此动作后，OTRS 会向外部系统发送已定义的确认 。

操作的后端

接下来我们将展示如何开发一个新操作，每个操作都必须实现这些子程序：

- new
- Run

我们应该实现这些方法中的每一个子程序，以便能够执行提供程序处理的动作（ Kernel/GenericInterface/Provider.pm ）。

操作代码示例

在本节中，展示了一个示例操作模块，并解释了每个子程序 。

```
# --
# Copyright (C) 2001-2020 OTRS AG, https://otrs.com/
# --
# This software comes with ABSOLUTELY NO WARRANTY. For details, see
# the enclosed file COPYING for license information (GPL). If you
# did not receive this file, see https://www.gnu.org/licenses/gpl-3.0.txt.
# --
```

(下页继续)

(续上页)

```
package Kernel::GenericInterface::Operation::Test::Test;

use strict;
use warnings;

use Kernel::System::VariableCheck qw(IsHashRefWithData);

our $ObjectManagerDisabled = 1;
```

这是可在常见 OTRS 模块中找到的通用标头。类/包名称通过 package 关键字声明。

我们还使用 VariableCheck 模块来对某些变量执行某些验证。对象管理器无法实例化操作。

```
sub new {
    my ( $Type, %Param ) = @_;

    my $Self = {};
    bless( $Self, $Type );

    # check needed objects
    for my $Needed (qw(DebuggerObject)) {
        if ( !$Param{$Needed} ) {
            return {
                Success      => 0,
                ErrorMessage => "Got no $Needed!"
            };
        }

        $Self->{$Needed} = $Param{$Needed};
    }

    return $Self;
}
```

构造函数 new 创建了一个新的类实例。根据编码指南，只有对象管理器没有处理的其它类的对象才需要在在 new 中创建。

```
sub Run {
    my ( $Self, %Param ) = @_;

    # check data - only accept undef or hash ref
    if ( defined $Param{Data} && ref $Param{Data} ne 'HASH' ) {

        return $Self->{DebuggerObject}->Error(
            Summary => 'Got Data but it is not a hash ref in Operation Test␣
↪backend)!'
        );
    }

    if ( defined $Param{Data} && $Param{Data}->{TestError} ) {

        return {
```

(下页继续)

(续上页)

```
            Success      => 0,
            ErrorMessage => "Error message for error code: $Param{Data}->
↪{TestError}",
            Data         => {
                ErrorData => $Param{Data}->{ErrorData},
            },
        };
    }

    # copy data
    my $ReturnData;

    if ( ref $Param{Data} eq 'HASH' ) {
        $ReturnData = \%{ $Param{Data} };
    }
    else {
        $ReturnData = undef;
    }

    # return result
    return {
        Success => 1,
        Data    => $ReturnData,
    };
}
```

Run 函数是每个操作的主要部分。它接收来自提供程序执行操作所需的远程系统的所有内部映射数据，它执行动作并将结果返回给提供程序以进行外部映射并将其传送回远程系统。

此特定示例返回与远程系统相同的数据，除非传递了 TestError 参数。在这种情况下，它会返回错误。

操作配置示例

需要将此操作模块注册以便在 OTRS GUI 中访问。这可以使用下面的 XML 配置来完成。

```
<ConfigItem Name="GenericInterface::Operation::Module###Test::Test" Required="0
↪" Valid="1">
    <Description Translatable="1">GenericInterface module registration for␣
↪the operation layer.</Description>
    <Group>GenericInterface</Group>
    <SubGroup>GenericInterface::Operation::ModuleRegistration</SubGroup>
    <Setting>
        <Hash>
            <Item Key="Name">Test</Item>
            <Item Key="Controller">Test</Item>
            <Item Key="ConfigDialog">AdminGenericInterfaceOperationDefault</
↪Item>
        </Hash>
    </Setting>
</ConfigItem>
```

单元测试示例

通用接口操作的单元测试与其它单元测试没有区别，但需要考虑本地测试，还需要模拟一个远程连接 。由于结果可能略有不同，因此分别测试两者是一个好习惯 。

参见:

要学习有关单元测试的更多信息，请查看单元测试 一章 。

以下只是单元测试的起点：

```
# --
# Copyright (C) 2001-2020 OTRS AG, https://otrs.com/
# --
# This software comes with ABSOLUTELY NO WARRANTY. For details, see
# the enclosed file COPYING for license information (GPL). If you
# did not receive this file, see https://www.gnu.org/licenses/gpl-3.0.txt.
# --

## no critic (Modules::RequireExplicitPackage)
use strict;
use warnings;
use utf8;

use vars (qw($Self));

use Kernel::GenericInterface::Debugger;
use Kernel::GenericInterface::Operation::Test::Test;

use Kernel::System::VariableCheck qw(:all);

# Skip SSL certificate verification (RestoreDatabase must not be used in this␣
↪test).
$Kernel::OM->ObjectParamAdd(
    'Kernel::System::UnitTest::Helper' => {
        SkipSSLVerify => 1,
    },
);
my $Helper = $Kernel::OM->Get('Kernel::System::UnitTest::Helper');

# get a random number
my $RandomID = $Helper->GetRandomNumber();

# create a new user for current test
my $UserLogin = $Helper->TestUserCreate(
    Groups => ['users'],
);
my $Password = $UserLogin;

my $UserID = $Kernel::OM->Get('Kernel::System::User')->UserLookup(
    UserLogin => $UserLogin,
);

# set web-service name
```

(下页继续)

(续上页)

```
my $WebserviceName = '-Test-' . $RandomID;

# create web-service object
my $WebserviceObject = $Kernel::OM->Get(
↪'Kernel::System::GenericInterface::Webservice');
$Self->Is(
    'Kernel::System::GenericInterface::Webservice',
    ref $WebserviceObject,
    "Create web service object",
);

my $WebserviceID = $WebserviceObject->WebserviceAdd(
    Name    => $WebserviceName,
    Config => {
        Debugger => {
            DebugThreshold => 'debug',
        },
        Provider => {
            Transport => {
                Type => '',
            },
        },
    },
    ValidID => 1,
    UserID  => 1,
);
$Self->True(
    $WebserviceID,
    "Added Web Service",
);

# get remote host with some precautions for certain unit test systems
my $Host = $Helper->GetTestHTTPHostname();

my $ConfigObject = $Kernel::OM->Get('Kernel::Config');

# prepare web-service config
my $RemoteSystem =
    $ConfigObject->Get('HttpType')
    . '://'
    . $Host
    . '/'
    . $ConfigObject->Get('ScriptAlias')
    . '/nph-genericinterface.pl/WebserviceID/'
    . $WebserviceID;

my $WebserviceConfig = {
    Description =>
        'Test for Ticket Connector using SOAP transport backend.',
    Debugger => {
        DebugThreshold => 'debug',
        TestMode       => 1,
```

(下页继续)

(续上页)

```
    },
    Provider => {
        Transport => {
            Type   => 'HTTP::SOAP',
            Config => {
                MaxLength => 10000000,
                NameSpace => 'http://otrs.org/SoapTestInterface/',
                Endpoint  => $RemoteSystem,
            },
        },
        Operation => {
            Test => {
                Type => 'Test::Test',
            },
        },
    },
    Requester => {
        Transport => {
            Type   => 'HTTP::SOAP',
            Config => {
                NameSpace => 'http://otrs.org/SoapTestInterface/',
                Encoding  => 'UTF-8',
                Endpoint  => $RemoteSystem,
            },
        },
        Invoker => {
            Test => {
                Type => 'Test::TestSimple'
                ,    # requester needs to be Test::TestSimple in order to␣
↪simulate a request to a remote system
            },
        },
    },
};

# update web-service with real config
# the update is needed because we are using
# the WebserviceID for the Endpoint in config
my $WebserviceUpdate = $WebserviceObject->WebserviceUpdate(
    ID      => $WebserviceID,
    Name    => $WebserviceName,
    Config  => $WebserviceConfig,
    ValidID => 1,
    UserID  => $UserID,
);
$Self->True(
    $WebserviceUpdate,
    "Updated Web Service $WebserviceID - $WebserviceName",
);

# debugger object
my $DebuggerObject = Kernel::GenericInterface::Debugger->new(
```

(下页继续)

(续上页)

```
    DebuggerConfig => {
        DebugThreshold => 'debug',
        TestMode       => 1,
    },
    WebserviceID      => $WebserviceID,
    CommunicationType => 'Provider',
);
$Self->Is(
    ref $DebuggerObject,
    'Kernel::GenericInterface::Debugger',
    'DebuggerObject instantiate correctly',
);

# define test cases
my @Tests = (
    {
        Name           => 'Test case name',
        SuccessRequest => 1,                    # 1 or 0
        RequestData    => {

            # ... add test data
        },
        ExpectedReturnLocalData => {
            Data => {

                # ... add expected local results
            },
            Success => 1,                       # 1 or 0
        },
        ExpectedReturnRemoteData => {
            Data => {

                # ... add expected remote results
            },
            Success => 1,                       # 1 or 0
        },
        Operation => 'Test',
    },

    # ... add more test cases
);

TEST:
for my $Test (@Tests) {

    # create local object
    my $LocalObject = "Kernel::GenericInterface::Operation::Test::$Test->
↪{Operation}"->new(
        DebuggerObject => $DebuggerObject,
        WebserviceID   => $WebserviceID,
    );
```

(下页继续)

(续上页)

```
    $Self->Is(
        "Kernel::GenericInterface::Operation::Test::$Test->{Operation}",
        ref $LocalObject,
        "$Test->{Name} - Create local object",
    );

    my %Auth = (
        UserLogin => $UserLogin,
        Password  => $Password,
    );
    if ( IsHashRefWithData( $Test->{Auth} ) ) {
        %Auth = %{ $Test->{Auth} };
    }

    # start requester with our web-service
    my $LocalResult = $LocalObject->Run(
        WebserviceID => $WebserviceID,
        Invoker      => $Test->{Operation},
        Data         => {
            %Auth,
            %{ $Test->{RequestData} },
        },
    );

    # check result
    $Self->Is(
        'HASH',
        ref $LocalResult,
        "$Test->{Name} - Local result structure is valid",
    );

    # create requester object
    my $RequesterObject = $Kernel::OM->Get(
↪'Kernel::GenericInterface::Requester');
    $Self->Is(
        'Kernel::GenericInterface::Requester',
        ref $RequesterObject,
        "$Test->{Name} - Create requester object",
    );

    # start requester with our web-service
    my $RequesterResult = $RequesterObject->Run(
        WebserviceID => $WebserviceID,
        Invoker      => $Test->{Operation},
        Data         => {
            %Auth,
            %{ $Test->{RequestData} },
        },
    );

    # check result
    $Self->Is(
```

(下页继续)

(续上页)

```
        'HASH',
        ref $RequesterResult,
        "$Test->{Name} - Requester result structure is valid",
    );

    $Self->Is(
        $RequesterResult->{Success},
        $Test->{SuccessRequest},
        "$Test->{Name} - Requester successful result",
    );

    # ... add tests for the results
}

# delete web service
my $WebserviceDelete = $WebserviceObject->WebserviceDelete(
    ID     => $WebserviceID,
    UserID => $UserID,
);
$Self->True(
    $WebserviceDelete,
    "Deleted Web Service $WebserviceID",
);

# also delete any other added data during the this test, since␣
↪RestoreDatabase must not be used.

1;
```

WSDL 扩展示例

WSDL 文件包含 Web 服务的定义及其 SOAP 消息的操作，本例中我们将在部分地方扩展 `development/webservices/GenericTickeConnectorSOAP.wsdl`：

端口类型：

```
<wsdl:portType name="GenericTicketConnector_PortType">
    <!-- ... -->
    <wsdl:operation name="Test">
        <wsdl:input message="tns:TestRequest"/>
        <wsdl:output message="tns:TestResponse"/>
    </wsdl:operation>
<!-- ... -->
```

绑定：

```
<wsdl:binding name="GenericTicketConnector_Binding" type=
↪"tns:GenericTicketConnector_PortType">
    <soap:binding style="document" transport="http://schemas.xmlsoap.org/soap/
↪http"/>
    <!-- ... -->
```

(下页继续)

(续上页)

```
    <wsdl:operation name="Test">
        <soap:operation soapAction="http://www.otrs.org/TicketConnector/Test"/
↪>
        <wsdl:input>
            <soap:body use="literal"/>
        </wsdl:input>
        <wsdl:output>
            <soap:body use="literal"/>
        </wsdl:output>
    </wsdl:operation>
    <!-- ... -->
</wsdl:binding>
```

类型：

```
<wsdl:types>
    <xsd:schema targetNamespace="http://www.otrs.org/TicketConnector/"␣
↪xmlns:xsd="http://www.w3.org/2001/XMLSchema">
    <!-- ... -->
        <xsd:element name="Test">
            <xsd:complexType>
                <xsd:sequence>
                        <xsd:element minOccurs="0" name="Param1" type=
↪"xsd:string"/>
                        <xsd:element minOccurs="0" name="Param2" type=
↪"xsd:positiveInteger"/>
                </xsd:sequence>
            </xsd:complexType>
        </xsd:element>
        <xsd:element name="TestResponse">
            <xsd:complexType>
                <xsd:sequence>
                    <xsd:element maxOccurs="unbounded" minOccurs="1" name=
↪"Attribute1" type="xsd:string"/>
                </xsd:sequence>
            </xsd:complexType>
        </xsd:element>
    <!-- ... -->
    </xsd:schema>
</wsdl:types>
```

消息：

```
<!-- ... -->
<wsdl:message name="TestRequest">
    <wsdl:part element="tns:Test" name="parameters"/>
</wsdl:message>
<wsdl:message name="TestResponse">
    <wsdl:part element="tns:TestResponse" name="parameters"/>
</wsdl:message>
<!-- ... -->
```

WADL 扩展示例

WADL 文件包含 Web 服务的定义及其 REST 接口的操作，向 development/webservices/GenericTickeConnectorREST.wadl 添加新资源。

```
<resources base="http://localhost/otrs/nph-genericinterface.pl/Webservice/
↪GenericTicketConnectorREST">
  <!-- ... -->
  <resource path="Test" id="Test">
    <doc xml:lang="en" title="Test"/>
      <param name="Param1" type="xs:string" required="false" default="" style=
↪"query" xmlns:xs="http://www.w3.org/2001/XMLSchema"/>
      <param name="Param2" type="xs:string" required="false" default="" style=
↪"query" xmlns:xs="http://www.w3.org/2001/XMLSchema"/>
      <method name="GET" id="GET_Test">
        <doc xml:lang="en" title="GET_Test"/>
        <request/>
        <response status="200">
          <representation mediaType="application/json; charset=UTF-8"/>
        </response>
      </method>
    </resource>
  </resource>
  <!-- ... -->
</resources>
```

Web 服务 SOAP 扩展示例

Web 服务可以通过具有预定义结构的 YAML 导入到 OTRS 中，在本例中我们将为 SOAP Web 服务扩展 development/webservices/GenericTickeConnectorSOAP.yml。

```
Provider:
  Operation:
    # ...
    Test:
      Description: This is only a test
      MappingInbound: {}
      MappingOutbound: {}
      Type: Test::Test
```

Web 服务 REST 扩展示例

Web 服务可以通过具有预定义结构的 YAML 导入到 OTRS 中，在本例中我们将为 REST Web 服务扩展 development/webservices/GenericTickeConnectorREST.yml。

```
Provider:
  Operation:
    # ...
    Test:
      Description: This is only a test
```

(下页继续)

(续上页)

```
      MappingInbound: {}
      MappingOutbound: {}
      Type: Test::Test
  # ...
  Transport:
    Config:
      # ...
      RouteOperationMapping:
        # ..
        Test:
          RequestMethod:
          - GET
          Route: /Test
```

3.3.20 OTRS 守护进程

OTRS 守护进程是一个独立的进程，可帮助 OTRS 异步执行某些操作并独立于 Web 服务器进程，但共享同一个数据库 。

OTRS 守护进程模块

OTRS 守护进程 `bin/otrs.Daemon.pl` 的主要目的是在系统配置中调用(守护进程)所有已注册的守护进程模块 。

每个守护程序模块必须实现一个通用 API 才能被 OTRS 守护程序正确调用，并且是系统中的半持久进程 。持久化进程的大小和内存使用量会随着时间的推移而增长，通常它们不会响应配置中的更改 。这就是为什么守护程序模块应该实现丢弃机制以便不时地中止并重新生成，从而释放系统资源并重新读取配置 。

守护进程模块可以是执行特定作业的一体化解决方案，但由于其复杂性，可能存在解决方案需要不同守护进程模块的情况 。这正是 OTRS 调度程序守护进程的情况，它被分成几个守护程序模块，包括用于任务管理和任务执行的一些守护进程模块 。

并不总是需要创建一个新的守护进程模块来执行某些任务，通常 OTRS 调度程序守护进程可以处理它们中的大多数，如果它是需要定期执行的 OTRS 函数(类似 CRON)或者如果它是由 OTRS 事件触发的，那么 OTRS 调度程序应该能够开箱即用或者通过添加新的调度程序任务工作模块来处理它 。

创建一个新的守护进程模块

所有守护进程模块都需要在系统配置中注册，以便由主 OTRS 守护进程调用 。

守护进程模块注册代码示例

```
<Setting Name="DaemonModules###TestDaemon" Required="1" Valid="1">
    <Description Translatable="1">The daemon registration for the scheduler␣
↪generic agent task manager.</Description>
    <Navigation>Daemon::ModuleRegistration</Navigation>
    <Value>
        <Hash>
            <Item Key="Module">
↪Kernel::System::Daemon::DaemonModules::TestDaemon</Item>
```

(下页继续)

(续上页)

```
        </Hash>
    </Value>
</Setting>
```

守护进程模块代码示例

以下代码实现了一个守护进程模块，该模块每 2 秒显示一次系统时间 。

```
# --
# Copyright (C) 2001-2020 OTRS AG, https://otrs.com/
# --
# This software comes with ABSOLUTELY NO WARRANTY. For details, see
# the enclosed file COPYING for license information (GPL). If you
# did not receive this file, see https://www.gnu.org/licenses/gpl-3.0.txt.
# --

package Kernel::System::Daemon::DaemonModules::TestDaemon;

use strict;
use warnings;
use utf8;

use Kernel::System::VariableCheck qw(:all);

use parent qw(Kernel::System::Daemon::BaseDaemon);

our @ObjectDependencies = (
    'Kernel::Config',
    'Kernel::System::Cache',
    'Kernel::System::DB',
);
```

这是可在常见 OTRS 模块中找到的通用标头 。类/包名称通过 `package` 关键字声明 。

在这个例子中，我们继承自 `BaseDaemon` 类，并设置了对象管理器依赖项 。

```
sub new {
    my ( $Type, %Param ) = @_;

    # Allocate new hash for object.
    my $Self = {};
    bless $Self, $Type;

    # Get objects in constructor to save performance.
    $Self->{ConfigObject} = $Kernel::OM->Get('Kernel::Config');
    $Self->{CacheObject}  = $Kernel::OM->Get('Kernel::System::Cache');
    $Self->{DBObject}     = $Kernel::OM->Get('Kernel::System::DB');

    # Disable in memory cache to be clusterable.
    $Self->{CacheObject}->Configure(
        CacheInMemory  => 0,
```

(下页继续)

(续上页)

```
        CacheInBackend => 1,
    );

    $Self->{SleepPost} = 2;           # sleep 2 seconds after each loop
    $Self->{Discard}   = 60 * 60;     # discard every hour

    $Self->{DiscardCount} = $Self->{Discard} / $Self->{SleepPost};

    $Self->{Debug}      = $Param{Debug};
    $Self->{DaemonName} = 'Daemon: TestDaemon';

    return $Self;
}
```

构造函数 new 创建了一个新的类实例。这里也创建了一些使用过的对象。强烈建议在守护进程模块中禁用内存高速缓存，尤其是在 OTRS 在集群环境中运行时。

为了使该守护进程模块每两秒执行一次，必须相应地定义休眠时间，否则将会尽快执行。

必须不时刷新守护进程模块，以便定义何时应该丢弃它。

对于以下函数（ PreRun、Run 和 PostRun ）如果它们返回 false，主 OTRS 守护进程将丢弃该对象并尽快创建一个新对象。

```
sub PreRun {
    my ( $Self, %Param ) = @_;

    # Check if database is on-line.
    return 1 if $Self->{DBObject}->Ping();

    sleep 10;

    return;
}
```

PreRun 方法在主守护进程模块方法之前执行，其目的是在实际操作之前执行一些测试。在这个例子中，对数据库进行检查（ 始终建议 ），否则它将休眠 10 秒。这是为了等待重新建立 DB 连接所必需的。

```
sub Run {
    my ( $Self, %Param ) = @_;

    print "Current time " . localtime . "\n";

    return 1;
}
```

Run 方法是主守护进程模块代码所在的位置，在这个例子中它只打印当前时间。

```
sub PostRun {
    my ( $Self, %Param ) = @_;
    sleep $Self->{SleepPost};
    $Self->{DiscardCount}--;

    if ( $Self->{Debug} ) {
```

(下页继续)

(续上页)

```
        print "  $Self->{DaemonName} Discard Count: $Self->{DiscardCount}\n";
    }

    return if $Self->{DiscardCount} <= 0;

    return 1;
}
```

PostRun 方法用于执行休眠（ 防止守护进程模块执行得太频繁 ），并且还用于管理对象的安全丢弃 。其它操作如验证或清理可以在这里完成 。

```
sub Summary {
    my ( $Self, %Param ) = @_;

    my %Summary = (
        Header => 'Test Daemon Summary:',
        Column => [
            {
                Name        => 'SomeColumn',
                DisplayName => 'Some Column',
                Size        => 15,
            },
            {
                Name        => 'AnotherColumn',
                DisplayName => 'Another Column',
                Size        => 15,
            },
            # ...
        ],
        Data => [
            {
                SomeColumn    => 'Some Data 1',
                AnotherColumn => 'Another Data 1',
            },
            {
                SomeColumn    => 'Some Data 2',
                AnotherColumn => 'Another Data 2',
            },
            # ...
        ],
        NoDataMesssage => '',
    );

    return \%Summary;
}
```

Summary 方法由控制台命令 Maint::Daemon::Summary 调用，它需要返回 Header 、Column 、Data 和 NoDataMessages 键 。Column 和 Data 需要是一个哈希数组 。它用于显示守护进程模块当前正在执行的操作的有用信息，或者到目前为止已完成的操作 。此方法是可选的 。

```
1;
```

文件结束 。

3.3.21 OTRS 调度程序

OTRS 调度程序是守护进程模块和任务工作程序的组合，它们一起运行，以便从 Web 服务器进程异步执行所有需要的 OTRS 任务 。

OTRS 调度程序任务管理器

SchedulerCronTaskManager 这将从 OTRS 系统配置中读取已注册的 cron 任务，并确定创建要执行的任务的正确时间 。

SchedulerFutureTaskManager 这将检查设置为将来仅执行一次的任务，并将此任务设置为及时执行 。例如，当通用接口调用程序无法访问远程服务器时，它可以自行调度在 5 分钟后再次运行 。

SchedulerGenericAgentTaskManager 这将持续读取设置为定期运行的自动任务的任务，并相应地设置其执行时间 。

只要这些任务管理器不够，就可以创建一个新的守护进程模块 。在它的 `Run()` 方法的某一点，它需要从 `chedulerDB` 对象调用 `TaskAdd()` 来注册一个任务，一旦它被注册，它就会在 `SchedulerTaskWorker` 的下一个空闲槽中被执行 。

OTRS 调度程序任务工作程序

SchedulerTaskWorker 这将执行先前任务管理器计划的所有任务以及通过异步执行程序直接来自代码的任务 。

为了执行每个任务，`SchedulerTaskWorker` 调用一个后端模块(任务工作者)来执行特定任务 。工作程序模块由任务类型确定 。如果添加了新任务类型，则需要新的任务工作程序 。

创建一个新的调度程序任务工作程序

放在 `Kernel/System/Daemon/DaemonModules/SchedulerTaskWorker` 下的所有文件都可能是任务工作程序，并且它们不需要在系统配置中进行任何注册 。

调度程序任务工作程序代码示例

```
# --
# Copyright (C) 2001-2020 OTRS AG, https://otrs.com/
# --
# This software comes with ABSOLUTELY NO WARRANTY. For details, see
# the enclosed file COPYING for license information (GPL). If you
# did not receive this file, see https://www.gnu.org/licenses/gpl-3.0.txt.
# --

package␣
 ↪Kernel::System::Daemon::DaemonModules::SchedulerTaskWorker::TestWorker;

use strict;
use warnings;

use parent qw(Kernel::System::Daemon::DaemonModules::BaseTaskWorker);
```

(下页继续)

(续上页)

```
our @ObjectDependencies = (
    'Kernel::System::Log',
);
```

这是可在常见 OTRS 模块中找到的通用标头。类/包名称通过 package 关键字声明。

在这个例子中，我们继承自 BaseTaskWorker 类，并设置了对象管理器依赖项。

```
sub new {
    my ( $Type, %Param ) = @_;

    my $Self = {};
    bless( $Self, $Type );

    $Self->{Debug}      = $Param{Debug};
    $Self->{WorkerName} = 'Worker: Test';

    return $Self;
}
```

构造函数 new 创建了一个新的类实例。

```
sub Run {
    my ( $Self, %Param ) = @_;

    # Check task params.
    my $CheckResult = $Self->_CheckTaskParams(
        %Param,
        NeededDataAttributes => [ 'NeededAtrribute1', 'NeededAtrribute2' ],
        DataParamsRef        => 'HASH', # or 'ARRAT'
    );

    # Stop execution if an error in params is detected.
    return if !$CheckResult;

    my $Success;
    my $ErrorMessage;

    if ( $Self->{Debug} ) {
        print "    $Self->{WorkerName} executes task: $Param{TaskName}\n";
    }

    do {

        # Localize the standard error.
        local *STDERR;

        # Redirect the standard error to a variable.
        open STDERR, ">>", \$ErrorMessage;

        $Success = $Kernel::OM->Get('Kernel::System::MyPackage')->Run(
            Param1 => 'someparam',
        );
```

(下页继续)

(续上页)

```
    };

    if ( !$Success ) {

        $ErrorMessage ||= "$Param{TaskName} execution failed without an error␣
↪message!";

        $Self->_HandleError(
            TaskName     => $Param{TaskName},
            TaskType     => 'Test',
            LogMessage   => "There was an error executing $Param{TaskName}:
↪$ErrorMessage",
            ErrorMessage => "$ErrorMessage",
        );
    }

    return $Success;
}
```

Run 是主要方法。从基类调用 _CheckTaskParams() 将节约一些代码行。在捕获 STDERR 时执行任务是一种非常好的做法，因为 OTRS 调度程序通常无人值守运行，并且将所有错误保存到变量将使其可用于进一步处理。_HandleError() 提供了一个通用接口，可以将错误消息作为电子邮件发送给系统配置中指定的收件人。

```
1;
```

文件结束。

3.3.22 概览

动态字段是可以添加到屏幕的自定义字段，以增强和添加信息到对象（例如工单或信件）。

工单或信件可以根据需要包含多个字段。也可以将动态字段框架用于其它对象，而不仅仅是工单或信件。

由于其模块化设计，每个动态字段类型都可以看作框架的插件，这个插件可以是 OTRS 标准包，用于扩展动态字段的可用类型，甚至可以扩展当前动态字段更多的功能。

3.3.23 动态字段框架

在创建新的动态字段之前，有必要了解它的框架以及 OTRS 屏幕如何与它们以及它们的底层 API 交互。

下图显示了动态字段框架的体系结构。

动态字段后端模块

动态字段（后端）

通常在前端模块中称为 BackendObject，是前端模块与每个特定动态字段实现或驱动程序之间的中介。它为所有动态字段驱动程序定义了通用的中间 API，并且每个驱动程序都有责任为字段的特定需求实现中间 API。

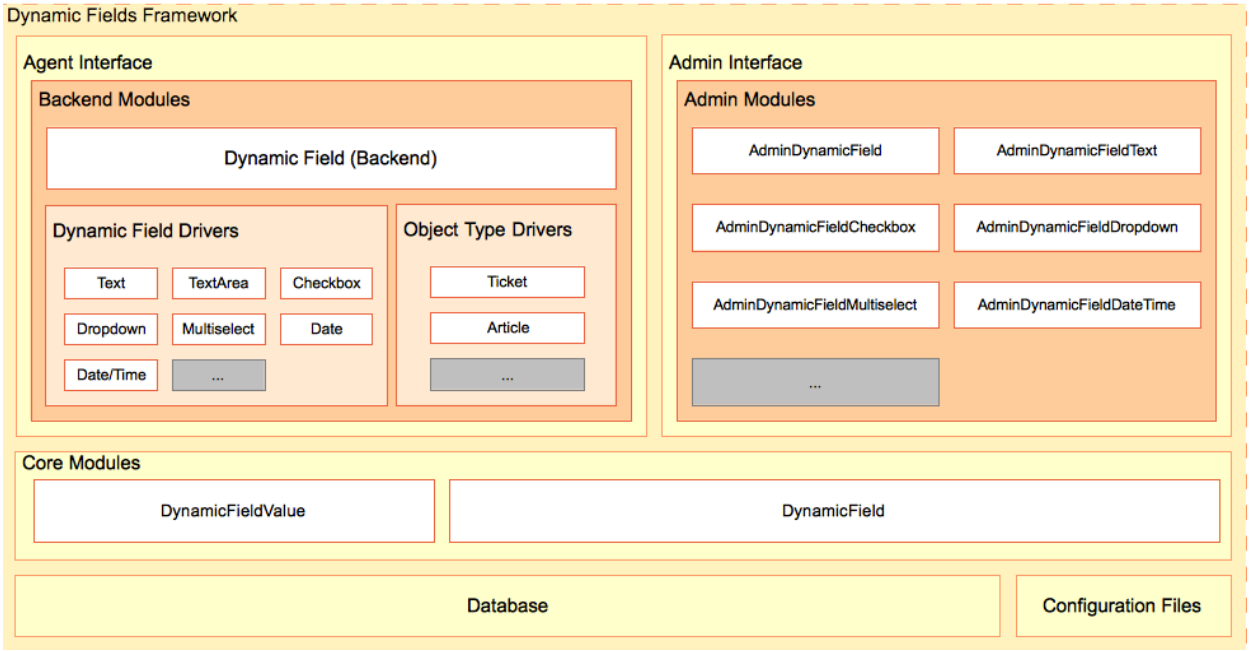

图 4: 动态字段框架

动态字段后端是所有驱动程序的主控制器。此模块中的每个函数负责检查所需的参数，并根据接收到的动态字段配置参数在特定驱动程序中调用相同的函数。

此模块还负责调用每个对象类型委托上的特定函数（如 `Ticket` 或 `Article`），例如添加一个历史记录条目或触发一个事件。

此模块位于 `$OTRS_HOME/Kernel/System/DynamicField/Backend.pm`。

动态字段驱动程序

动态字段驱动程序是动态字段的实现。每个驱动程序都必须实现后端中指定的所有强制函数（有些函数依赖于某个行为，如果动态字段没有该特定行为，则不需要实现这些函数）。

驱动程序负责知道如何从 Web 请求或配置文件（如搜索配置文件）获取自己的值。它还需要知道在编辑或显示屏幕中呈现字段的 HTML 代码，或者如何与统计模块交互，以及其它函数。

这些模块位于 `$OTRS_HOME/Kernel/System/DynamicField/Driver/*.pm`。

存在一些基本驱动程序，如 `Base.pm`、`BaseText.pm`、`BaseSelect.pm` 和 `BaseDateTime.pm`，它们为某些驱动程序实现通用（例如，`TextArea.pm` 驱动程序使用 `BaseText.pm` 也使用 `Base.pm`，然后 `TextArea` 只需要实现 `Base.pm` 和 `BaseText.pm` 缺失的函数或某些特殊情况的函数）。

以下是驱动程序继承树：

- `Base.pm`
  - `BaseText.pm`
    - `Text.pm`
    - `TextArea.pm`
  - `BaseSelect.pm`
    - `Dropdown.pm`

        * Multiselect.pm
    - BaseDateTime.pm
        * DateTime.pm
        * Date.pm
    - Checkbox.pm

对象类型委托

对象类型委托负责对链接到动态字段的对象执行特定功能。这些功能是由后端对象根据需要触发的。

这些模块位于 $OTRS_HOME/Kernel/System/DynamicField/ObjectType/*.pm。

动态字段管理模块

为了管理动态字段（添加、编辑和列出），已经开发了一系列模块。有一个特定的主模块（AdminDynamicField.pm），它显示定义的动态字段列表，并从其它模块中调用以创建新的动态字段或修改现有的动态字段。

通常，动态字段驱动程序需要自己的管理模块（管理对话框）来定义其属性。此对话框可能与其它驱动程序不同。但这不是强制性的，驱动程序可以共享管理对话框，前提是它们可以为链接到它们的所有驱动程序提供所需的信息，无论它们来自不同的类型。必须将每个驱动程序链接到管理对话框（例如，文本和文本区域驱动程序共享 AdminDynamicFieldText.pm 管理对话框，以及日期和日期/时间驱动程序共享 AdminDynamicFieldDateTime.pm 管理对话框）。

管理对话框遵循正常的 OTRS 管理模块规则和体系结构。但是为了标准化，所有动态字段的所有配置公共部分在所有管理对话框中应该具有相同的外观和感觉。

这些模块位于 $OTRS_HOME/Kernel/Modules/*.pm。

注解：每个管理对话框都需要相应的 HTML 模板文件（.tt）。

动态字段核心模块

此模块从数据库表读取动态字段信息并将其写入数据库表。

DynamicField.pm 此模块负责管理动态字段定义。它为添加、更改、删除、列出和获取动态字段提供了基本的 API。此模块位于 $OTRS_HOME/Kernel/System/DynamicField.pm。

DynamicFieldValue.pm 此模块负责将动态字段值读写到表单和数据库中。此模块被驱动程序高度使用，位于 $OTRS_HOME/Kernel/System/DynamicFieldValue.pm。

动态字段数据库表

数据库中有两个表用于存储动态字段信息：

dynamic_field 由核心模块 DynamicField.pm 使用，它存储动态字段定义。

dynamic_field_value 由核心模块 DynamicFieldValue.pm 用于保存每个动态字段和每个对象类型实例的动态字段值。

动态字段配置文件

后端模块需要一种方法来了解哪些驱动程序存在，并且因为驱动程序的数量可以轻松扩展。管理它们的最简单方法是使用系统配置，其中可以存储和扩展动态字段驱动程序和对象类型驱动程序的信息。

主管理模块还需要知道有关可用动态字段驱动程序的信息，以使用与之链接的管理对话框来创建或修改动态字段。

前端模块需要读取系统配置以了解每个屏幕的哪些动态字段是活动的，哪些是强制性的。例如：`Ticket::Frontend::AgentTicketPhone###DynamicField` 存储 新建电话工单屏幕的活动、必填和非活动的动态字段。

3.3.24 动态字段与前端模块交互

了解前端模块如何与动态字段交互并不是扩展工单或信件对象的动态字段所必需的，因为可以使用动态字段的所有屏幕都已经准备好了。但在定制开发或将动态字段扩展到其它对象的情况下，了解如何从前端模块访问动态字段框架非常有用。

下图显示了动态字段如何与其它 OTRS 框架部分交互的简单示例。

第一步是前端模块读取配置的动态字段。例如，`AgentTicketNote` 应该读取 `Ticket::Frontend::AgentTicketNote###DynamicField` 设置。此设置可用作 `DynamicField` 核心模块函数 `DynamicFieldListGet()` 的过滤器参数。该屏幕可以存储此功能的结果，以便为此特定屏幕激活动态字段列表。

接下来，该屏幕应该尝试从 Web 请求中获取值。为此，它可以使用后端对象函数 `EditFieldValueGet()`，并可以使用此值来触发 ACL。后端对象将使用每个驱动程序执行所有功能的特定操作。

要继续，该屏幕应获取每个字段的 HTML 以显示它。后端对象函数 `EditFieldRender()` 可用于执行此操作，并且可以将 ACL 限制以及 Web 请求中的值传递给此函数，以获得更好的结果。如果提交，该屏幕也可以使用后端对象函数 `EditFieldValueValidate()` 来检查必填字段。

注解：如果屏幕只显示字段值，则其它屏幕可以使用 `DisplayFieldRender()` 而不是 `EditFieldRender()`，在这种情况下，不需要进行值验证。

要获取对象 ID，需要存储动态字段的值。对于此示例，如果动态字段链接到工单对象，则屏幕应该已经具有 `TicketID`，否则如果该字段链接到信件对象，要设置该字段的值，则必须首先创建该信件。后端对象的 `ValueSet()` 可用于设置动态字段值。

总之，前端模块不需要知道每个动态字段是如何在内部工作来获取或设置它们的值或显示它们。它只需要调用后端对象模块并以通用的方式使用字段。

3.3.25 如何扩展动态字段

有很多方法可以扩展动态字段。以下部分将尝试介绍最常见的场景。

创建新的动态字段类型（用于工单或信件对象）

要创建新的动态字段类型，必须：

1. 创建一个动态字段驱动程序。这是新字段的主要模块。
2. 创建或使用现有的管理对话框以具有管理界面并设置其配置选项。

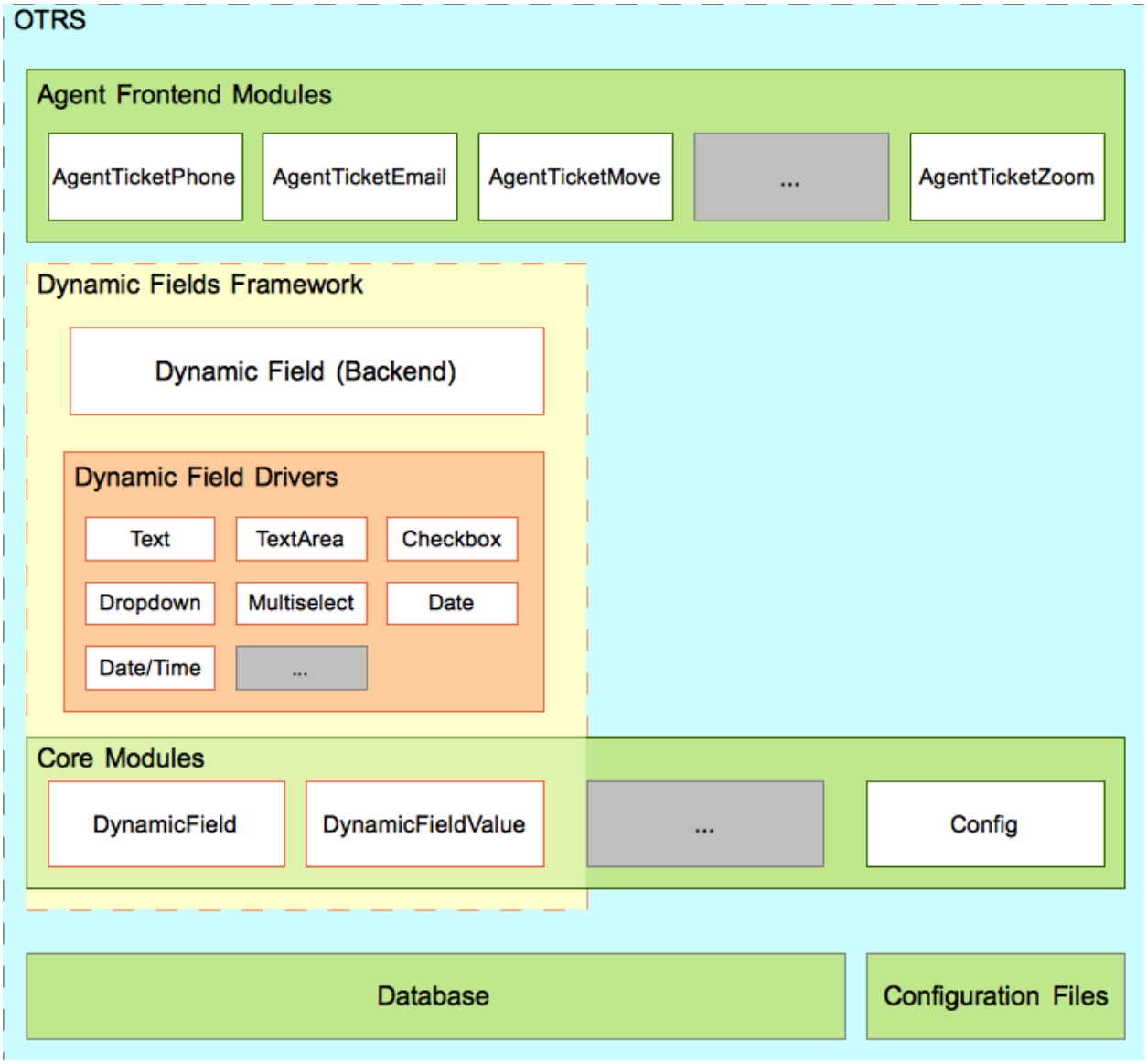
OTRS
Agent Frontend Modules
AgentTicketPhone
AgentTicketEmail
AgentTicketMove
...
AgentTicketZoom
Dynamic Fields Framework
Dynamic Field (Backend)
Dynamic Field Drivers
Text
TextArea
Checkbox
Dropdown
Multiselect
Date
Date/Time
...
Core Modules
DynamicField
DynamicFieldValue
...
Config
Database
Configuration Files

图 5: 动态字段交互

3. 创建一个配置文件来注册后端中的新字段（ 如果需要，也可以在框架中创建新的管理对话框 ），并能创建其实例 。

创建新的动态字段类型（ 用于其它对象 ）

要为其它对象创建新的动态字段类型，必须：

1. 创建一个动态字段驱动程序 。这是新字段的主要模块 。
2. 创建对象类型委托 。这是必要的，即使另一个对象在其函数中不需要任何特定的数据处理（ 例如，在设置了一个值之后 ）。所有对象类型委托都必须实现后端所需的功能 。

   查看当前对象类型委托以实现相同的函数，即使它们只是为其它对象返回一个成功的值 。
3. 创建或使用现有的管理对话框以具有管理界面并设置其配置选项 。
4. 在前端模块中实现动态字段，以便能够使用动态字段 。
5. 创建一个配置文件来注册后端中的新字段（ 如果需要，也可以在框架中创建新的管理对话框 ），并能创建其实例 。

   并进行必要的设置以在新屏幕中强制显示 、隐藏或显示动态字段 。

创建新包以使用动态字段

要创建使用现有动态字段的包，必须：

1. 在前端模块中实现动态字段，以便能够使用动态字段 。
2. 创建一个配置文件，使最终用户能够在新屏幕中显示 、隐藏或强制显示动态字段 。

扩展后端和驱动程序功能

后端对象可能没有自定义开发所需的函数，或者也可能具有所需的函数，但是返回格式不符合自定义开发的需要，或者执行新的或旧的函数需要新的行为 。

最简单的方法是扩展当前字段文件 。为此，有必要创建一个新的后端扩展文件来定义新的函数，并创建驱动程序扩展来为每个字段实现这些新函数 。这些新的驱动程序只需要实现新的功能，因为原始驱动程序负责标准功能 。所有这些新文件都不需要构造函数，因为它们将作为后端对象和驱动程序的基础进行加载 。

唯一的限制是函数的命名应该与后端和驱动程序上的不同，否则它们将被当前对象覆盖 。

将新的后端扩展放入 `DynamicField` 目录（ 例如 `/$OTRS_HOME/Kernel/System/DynamicField/NewPackageBackend.pm` 及其驱动程序位于 `/$OTRS_HOME/Kernel/System/DynamicField/Driver/NewPackage*.pm` ）。

新行为只需要扩展配置文件中的一个小型设置 。

要创建新的后端函数，需要：

1. 创建一个新的后端扩展模块以仅定义新函数 。
2. 创建动态字段驱动程序扩展以仅实现新函数 。
3. 在前端模块中实现新的动态字段功能，以便能够使用新的动态字段功能 。
4. 创建一个配置文件以注册新的后端和驱动程序扩展和行为 。

其它扩展

其它扩展可以是上述示例的组合 。

3.3.26 创建一个新动态字段

为了说明此过程，将创建一个新的动态字段 *Password* 。此新动态字段类型将为工单或信件对象显示新的密码字段 。由于与文本动态字段非常相似，我们将使用 `Base` 和 `BaseText` 驱动程序作为构建此新字段的基础 。

> **警告：** 此新密码字段实现仅用于教育目的，它不提供任何级别的安全性，不建议用于生产系统 。

要创建这个新的动态字段，我们将创建 4 个文件：

1. 一个用于注册模块的配置文件（ XML ）。
2. 一个用于设置字段选项的管理对话框模块（ Perl ）。
3. 一个管理对话框的模板模块 。
4. 一个动态字段驱动程序（ Perl ）。

文件结构：

```
$HOME (e. g. /opt/otrs/)
|
...
|--/Kernel/
|   |--/Config/
|   |   |--/Files/
|   |   |   |--/XML/
|   |   |   |   |DynamicFieldPassword.xml
...
|   |--/Modules/
|   |   |AdminDynamicFieldPassword.pm
...
|   |--/Output/
|   |   |--/HTML/
|   |   |   |--/Standard/
|   |   |   |   |AdminDynamicFieldPassword.tt
...
|   |--/System/
|   |   |--/DynamicField/
|   |   |   |--/Driver/
|   |   |   |   |Password.pm
...
```

Password 动态字段的文件

动态字段配置文件示例

配置文件用于注册后端对象的动态字段类型（ 驱动程序 ）和对象类型驱动程序 。它们还在框架中存储管理模块的标准注册 。

在本节中，将显示和解释 password 动态字段的配置文件 。

```
<?xml version="1.0" encoding="utf-8" ?>
<otrs_config version="2.0" init="Application">
```

这是配置文件的正常头文件 。

```
<ConfigItem Name="DynamicFields::Driver###Password" Required="0" Valid="1">
    <Description Translatable="1">DynamicField backend registration.</
↪Description>
    <Group>DynamicFieldPassword</Group>
    <SubGroup>DynamicFields::Backend::Registration</SubGroup>
    <Setting>
        <Hash>
            <Item Key="DisplayName" Translatable="1">Password</Item>
            <Item Key="Module">Kernel::System::DynamicField::Driver::Password
↪</Item>
            <Item Key="ConfigDialog">AdminDynamicFieldPassword</Item>
        </Hash>
    </Setting>
</ConfigItem>
```

此设置为后端模块注册 password 动态字段驱动程序，以便它可以包含在可用动态字段类型列表中 。它还在键 ConfigDialog 中指定了自己的管理对话框 。主动态字段管理模块使用此键来管理此新的动态字段类型 。

```
<ConfigItem Name="Frontend::Module###AdminDynamicFieldPassword" Required="0"␣
↪Valid="1">
    <Description Translatable="1">Frontend module registration for the agent␣
↪interface.</Description>
    <Group>DynamicFieldPassword</Group>
    <SubGroup>Frontend::Admin::ModuleRegistration</SubGroup>
    <Setting>
        <FrontendModuleReg>
            <Group>admin</Group>
            <Description>Admin</Description>
            <Title Translatable="1">Dynamic Fields Text Backend GUI</Title>
            <Loader>
                <JavaScript>Core.Agent.Admin.DynamicField.js</JavaScript>
            </Loader>
        </FrontendModuleReg>
    </Setting>
</ConfigItem>
```

这是管理界面中 password 管理对话框的标准模块注册 。

```
</otrs_config>
```

一个配置文件的标准关闭 。

动态字段管理对话框示例

管理对话框是管理（ 添加或编辑 ）动态字段的标准管理模块 。

在本节中，将显示和解释 password 动态字段的管理对话框 。

```
# --
# Copyright (C) 2001-2020 OTRS AG, https://otrs.com/
# --
# This software comes with ABSOLUTELY NO WARRANTY. For details, see
# the enclosed file COPYING for license information (GPL). If you
# did not receive this file, see https://www.gnu.org/licenses/gpl-3.0.txt.
# --

package Kernel::Modules::AdminDynamicFieldPassword;

use strict;
use warnings;

use Kernel::System::VariableCheck qw(:all);
use Kernel::System::Valid;
use Kernel::System::CheckItem;
use Kernel::System::DynamicField;
```

这是可在常见 OTRS 模块中找到的通用标头。类/包名称通过 `package` 关键字声明。

```
sub new {
    my ( $Type, %Param ) = @_;

    my $Self = {%Param};
    bless( $Self, $Type );

    for (qw(ParamObject LayoutObject LogObject ConfigObject)) {
        if ( !$Self->{$_} ) {
            $Self->{LayoutObject}->FatalError( Message => "Got no $_!" );
        }
    }

    # create additional objects
    $Self->{ValidObject} = Kernel::System::Valid->new( %{$Self} );

    $Self->{DynamicFieldObject} = Kernel::System::DynamicField->new( %{$Self}␣
↪);

    # get configured object types
    $Self->{ObjectTypeConfig} = $Self->{ConfigObject}->Get(
↪'DynamicFields::ObjectType');

    # get the fields config
    $Self->{FieldTypeConfig} = $Self->{ConfigObject}->Get(
↪'DynamicFields::Backend') || {};

    $Self->{DefaultValueMask} = '****';
    return $Self;
}
```

构造函数 `new` 创建了一个新的类实例。根据编码指南，必须在 `new` 中创建此模块中所需的其它类的对象。

```
sub Run {
```

(下页继续)

(续上页)

```
    my ( $Self, %Param ) = @_;

    if ( $Self->{Subaction} eq 'Add' ) {
        return $Self->_Add(
            %Param,
        );
    }
    elsif ( $Self->{Subaction} eq 'AddAction' ) {

        # challenge token check for write action
        $Self->{LayoutObject}->ChallengeTokenCheck();

        return $Self->_AddAction(
            %Param,
        );
    }
    if ( $Self->{Subaction} eq 'Change' ) {

        return $Self->_Change(
            %Param,
        );
    }
    elsif ( $Self->{Subaction} eq 'ChangeAction' ) {

        # challenge token check for write action
        $Self->{LayoutObject}->ChallengeTokenCheck();

        return $Self->_ChangeAction(
            %Param,
        );
    }

    return $Self->{LayoutObject}->ErrorScreen(
        Message => "Undefined subaction.",
    );
}
```

Run 是 Web 请求调用的默认函数。我们尝试使这个函数尽可能简单，并让 helper 函数完成艰苦的工作。

```
sub _Add {
    my ( $Self, %Param ) = @_;

    my %GetParam;
    for my $Needed (qw(ObjectType FieldType FieldOrder)) {
        $GetParam{$Needed} = $Self->{ParamObject}->GetParam( Param => $Needed␣
↪);
        if ( !$Needed ) {

            return $Self->{LayoutObject}->ErrorScreen(
                Message => "Need $Needed",
            );
        }
```

(下页继续)

(续上页)

```
    }

    # get the object type and field type display name
    my $ObjectTypeName = $Self->{ObjectTypeConfig}->{ $GetParam{ObjectType} }->
↪{DisplayName} || '';
    my $FieldTypeName  = $Self->{FieldTypeConfig}->{ $GetParam{FieldType} }->
↪{DisplayName}   || '';

    return $Self->_ShowScreen(
        %Param,
        %GetParam,
        Mode           => 'Add',
        ObjectTypeName => $ObjectTypeName,
        FieldTypeName  => $FieldTypeName,
    );
}
```

_Add 函数也很简单，它只是从 Web 请求获取一些参数并调用 _ShowScreen() 函数。通常，不需要修改此函数。

```
sub _AddAction {
    my ( $Self, %Param ) = @_;

    my %Errors;
    my %GetParam;

    for my $Needed (qw(Name Label FieldOrder)) {
        $GetParam{$Needed} = $Self->{ParamObject}->GetParam( Param => $Needed␣
↪);
        if ( !$GetParam{$Needed} ) {
            $Errors{ $Needed . 'ServerError' }        = 'ServerError';
            $Errors{ $Needed . 'ServerErrorMessage' } = 'This field is␣
↪required.';
        }
    }

    if ( $GetParam{Name} ) {

        # check if name is alphanumeric
        if ( $GetParam{Name} !~ m{\A ( ?: [a-zA-Z] | \d )+ \z}xms ) {

            # add server error error class
            $Errors{NameServerError} = 'ServerError';
            $Errors{NameServerErrorMessage} =
                'The field does not contain only ASCII letters and numbers.';
        }

        # check if name is duplicated
        my %DynamicFieldsList = %{
            $Self->{DynamicFieldObject}->DynamicFieldList(
                Valid      => 0,
                ResultType => 'HASH',
```

(下页继续)

(续上页)

```
                )
        };

        %DynamicFieldsList = reverse %DynamicFieldsList;

        if ( $DynamicFieldsList{ $GetParam{Name} } ) {

            # add server error error class
            $Errors{NameServerError}        = 'ServerError';
            $Errors{NameServerErrorMessage} = 'There is another field with the␣
↪same name.';
        }
    }

    if ( $GetParam{FieldOrder} ) {

        # check if field order is numeric and positive
        if ( $GetParam{FieldOrder} !~ m{\A ( ?: \d )+ \z}xms ) {

            # add server error error class
            $Errors{FieldOrderServerError}        = 'ServerError';
            $Errors{FieldOrderServerErrorMessage} = 'The field must be numeric.
↪';
        }
    }

    for my $ConfigParam (
        qw(
        ObjectType ObjectTypeName FieldType FieldTypeName DefaultValue␣
↪ValidID ShowValue
        ValueMask
        )
        )
    {
        $GetParam{$ConfigParam} = $Self->{ParamObject}->GetParam( Param =>
↪$ConfigParam );
    }

    # uncorrectable errors
    if ( !$GetParam{ValidID} ) {

        return $Self->{LayoutObject}->ErrorScreen(
            Message => "Need ValidID",
        );
    }

    # return to add screen if errors
    if (%Errors) {

        return $Self->_ShowScreen(
            %Param,
            %Errors,
```

(下页继续)

(续上页)

```
            %GetParam,
            Mode => 'Add',
        );
    }

    # set specific config
    my $FieldConfig = {
        DefaultValue => $GetParam{DefaultValue},
        ShowValue    => $GetParam{ShowValue},
        ValueMask    => $GetParam{ValueMask} || $Self->{DefaultValueMask},
    };

    # create a new field
    my $FieldID = $Self->{DynamicFieldObject}->DynamicFieldAdd(
        Name       => $GetParam{Name},
        Label      => $GetParam{Label},
        FieldOrder => $GetParam{FieldOrder},
        FieldType  => $GetParam{FieldType},
        ObjectType => $GetParam{ObjectType},
        Config     => $FieldConfig,
        ValidID    => $GetParam{ValidID},
        UserID     => $Self->{UserID},
    );

    if ( !$FieldID ) {

        return $Self->{LayoutObject}->ErrorScreen(
            Message => "Could not create the new field",
        );
    }

    return $Self->{LayoutObject}->Redirect(
        OP => "Action=AdminDynamicField",
    );
}
```

_AddAction' 函数从新的动态字段中获取配置参数，并验证动态字段名称只包含字母和数字。此函数可以验证任何其它参数。

Name、Label、FieldOrder、Validity 是所有动态字段的通用参数，它们是必需的。每个动态字段都有其特定的配置，必须至少包含 DefaultValue 参数。在这种情况下，它还有密码字段的 ShowValue 和 ValueMask 参数。

如果该字段能够存储固定的值列表，则应将它们存储在特定配置哈希内的 PossibleValues 参数中。

与其它管理模块一样，如果参数无效，则此函数将返回添加屏幕，突出显示错误的表单字段。

如果所有参数都正确，则会创建一个新的动态字段。

```
sub _Change {
    my ( $Self, %Param ) = @_;

    my %GetParam;
    for my $Needed (qw(ObjectType FieldType)) {
```

(下页继续)

(续上页)

```
        $GetParam{$Needed} = $Self->{ParamObject}->GetParam( Param => $Needed␣
↪);
        if ( !$Needed ) {

            return $Self->{LayoutObject}->ErrorScreen(
                Message => "Need $Needed",
            );
        }
    }

    # get the object type and field type display name
    my $ObjectTypeName = $Self->{ObjectTypeConfig}->{ $GetParam{ObjectType} }->
↪{DisplayName} || '';
    my $FieldTypeName  = $Self->{FieldTypeConfig}->{ $GetParam{FieldType} }->
↪{DisplayName}   || '';

    my $FieldID = $Self->{ParamObject}->GetParam( Param => 'ID' );

    if ( !$FieldID ) {

        return $Self->{LayoutObject}->ErrorScreen(
            Message => "Need ID",
        );
    }

    # get dynamic field data
    my $DynamicFieldData = $Self->{DynamicFieldObject}->DynamicFieldGet(
        ID => $FieldID,
    );

    # check for valid dynamic field configuration
    if ( !IsHashRefWithData($DynamicFieldData) ) {

        return $Self->{LayoutObject}->ErrorScreen(
            Message => "Could not get data for dynamic field $FieldID",
        );
    }

    my %Config = ();

    # extract configuration
    if ( IsHashRefWithData( $DynamicFieldData->{Config} ) ) {
        %Config = %{ $DynamicFieldData->{Config} };
    }

    return $Self->_ShowScreen(
        %Param,
        %GetParam,
        %${DynamicFieldData},
        %Config,
        ID              => $FieldID,
        Mode            => 'Change',
```

(下页继续)

(续上页)

```
        ObjectTypeName => $ObjectTypeName,
        FieldTypeName  => $FieldTypeName,
    );
}
```

_Change 函数与 _Add 函数非常相似，但由于此函数用于编辑现有字段，因此需要验证 FieldID 参数并收集当前动态字段数据 。

```
sub _ChangeAction {
    my ( $Self, %Param ) = @_;

    my %Errors;
    my %GetParam;

    for my $Needed (qw(Name Label FieldOrder)) {
        $GetParam{$Needed} = $Self->{ParamObject}->GetParam( Param => $Needed␣
↪);
        if ( !$GetParam{$Needed} ) {
            $Errors{ $Needed . 'ServerError' }        = 'ServerError';
            $Errors{ $Needed . 'ServerErrorMessage' } = 'This field is␣
↪required.';
        }
    }

    my $FieldID = $Self->{ParamObject}->GetParam( Param => 'ID' );
    if ( !$FieldID ) {

        return $Self->{LayoutObject}->ErrorScreen(
            Message => "Need ID",
        );
    }

    if ( $GetParam{Name} ) {

        # check if name is lowercase
        if ( $GetParam{Name} !~ m{\A ( ?: [a-zA-Z] | \d )+ \z}xms ) {

            # add server error error class
            $Errors{NameServerError} = 'ServerError';
            $Errors{NameServerErrorMessage} =
                'The field does not contain only ASCII letters and numbers.';
        }

        # check if name is duplicated
        my %DynamicFieldsList = %{
            $Self->{DynamicFieldObject}->DynamicFieldList(
                Valid      => 0,
                ResultType => 'HASH',
                )
        };

        %DynamicFieldsList = reverse %DynamicFieldsList;
```

(下页继续)

(续上页)

```
        if (
            $DynamicFieldsList{ $GetParam{Name} } &&
            $DynamicFieldsList{ $GetParam{Name} } ne $FieldID
            )
        {

            # add server error class
            $Errors{NameServerError}        = 'ServerError';
            $Errors{NameServerErrorMessage} = 'There is another field with the␣
↪same name.';
        }
    }

    if ( $GetParam{FieldOrder} ) {

        # check if field order is numeric and positive
        if ( $GetParam{FieldOrder} !~ m{\A ( ?: \d )+ \z}xms ) {

            # add server error error class
            $Errors{FieldOrderServerError}        = 'ServerError';
            $Errors{FieldOrderServerErrorMessage} = 'The field must be numeric.
↪';
        }
    }

    for my $ConfigParam (
        qw(
        ObjectType ObjectTypeName FieldType FieldTypeName DefaultValue␣
↪ValidID ShowValue
        ValueMask
        )
        )
    {
        $GetParam{$ConfigParam} = $Self->{ParamObject}->GetParam( Param =>
↪$ConfigParam );
    }

    # uncorrectable errors
    if ( !$GetParam{ValidID} ) {

        return $Self->{LayoutObject}->ErrorScreen(
            Message => "Need ValidID",
        );
    }

    # get dynamic field data
    my $DynamicFieldData = $Self->{DynamicFieldObject}->DynamicFieldGet(
        ID => $FieldID,
    );

    # check for valid dynamic field configuration
```

(下页继续)

(续上页)

```
    if ( !IsHashRefWithData($DynamicFieldData) ) {

        return $Self->{LayoutObject}->ErrorScreen(
            Message => "Could not get data for dynamic field $FieldID",
        );
    }

    # return to change screen if errors
    if (%Errors) {

        return $Self->_ShowScreen(
            %Param,
            %Errors,
            %GetParam,
            ID   => $FieldID,
            Mode => 'Change',
        );
    }

    # set specific config
    my $FieldConfig = {
        DefaultValue => $GetParam{DefaultValue},
        ShowValue    => $GetParam{ShowValue},
        ValueMask    => $GetParam{ValueMask},
    };

    # update dynamic field (FieldType and ObjectType cannot be changed; use␣
↪old values)
    my $UpdateSuccess = $Self->{DynamicFieldObject}->DynamicFieldUpdate(
        ID         => $FieldID,
        Name       => $GetParam{Name},
        Label      => $GetParam{Label},
        FieldOrder => $GetParam{FieldOrder},
        FieldType  => $DynamicFieldData->{FieldType},
        ObjectType => $DynamicFieldData->{ObjectType},
        Config     => $FieldConfig,
        ValidID    => $GetParam{ValidID},
        UserID     => $Self->{UserID},
    );

    if ( !$UpdateSuccess ) {

        return $Self->{LayoutObject}->ErrorScreen(
            Message => "Could not update the field $GetParam{Name}",
        );
    }

    return $Self->{LayoutObject}->Redirect(
        OP => "Action=AdminDynamicField",
    );
}
```

_ChangeAction() 与 _AddAction() 非常相似，但适用于更新现有字段而不是创建新字段 。

```
sub _ShowScreen {
    my ( $Self, %Param ) = @_;

    $Param{DisplayFieldName} = 'New';

    if ( $Param{Mode} eq 'Change' ) {
        $Param{ShowWarning}      = 'ShowWarning';
        $Param{DisplayFieldName} = $Param{Name};
    }

    # header
    my $Output = $Self->{LayoutObject}->Header();
    $Output .= $Self->{LayoutObject}->NavigationBar();

    # get all fields
    my $DynamicFieldList = $Self->{DynamicFieldObject}->DynamicFieldListGet(
        Valid => 0,
    );

    # get the list of order numbers (is already sorted).
    my @DynamicfieldOrderList;
    for my $Dynamicfield ( @{$DynamicFieldList} ) {
        push @DynamicfieldOrderList, $Dynamicfield->{FieldOrder};
    }

    # when adding we need to create an extra order number for the new field
    if ( $Param{Mode} eq 'Add' ) {

        # get the last element from the order list and add 1
        my $LastOrderNumber = $DynamicfieldOrderList[-1];
        $LastOrderNumber++;

        # add this new order number to the end of the list
        push @DynamicfieldOrderList, $LastOrderNumber;
    }

    my $DynamicFieldOrderSrtg = $Self->{LayoutObject}->BuildSelection(
        Data          => \@DynamicfieldOrderList,
        Name          => 'FieldOrder',
        SelectedValue => $Param{FieldOrder} || 1,
        PossibleNone  => 0,
        Class         => 'W50pc Validate_Number',
    );

    my %ValidList = $Self->{ValidObject}->ValidList();

    # create the Validity select
    my $ValidityStrg = $Self->{LayoutObject}->BuildSelection(
        Data         => \%ValidList,
        Name         => 'ValidID',
        SelectedID   => $Param{ValidID} || 1,
        PossibleNone => 0,
```

(下页继续)

(续上页)

```
        Translation  => 1,
        Class        => 'W50pc',
    );

    # define config field specific settings
    my $DefaultValue = ( defined $Param{DefaultValue} ? $Param{DefaultValue} :
↪'' );

    # create the Show value select
    my $ShowValueStrg = $Self->{LayoutObject}->BuildSelection(
        Data => [ 'No', 'Yes' ],
        Name => 'ShowValue',
        SelectedValue => $Param{ShowValue} || 'No',
        PossibleNone  => 0,
        Translation   => 1,
        Class         => 'W50pc',
    );

    # generate output
    $Output .= $Self->{LayoutObject}->Output(
        TemplateFile => 'AdminDynamicFieldPassword',
        Data         => {
            %Param,
            ValidityStrg          => $ValidityStrg,
            DynamicFieldOrderSrtg => $DynamicFieldOrderSrtg,
            DefaultValue          => $DefaultValue,
            ShowValueStrg         => $ShowValueStrg,
            ValueMask             => $Param{ValueMask} || $Self->
↪{DefaultValueMask},
        },
    );

    $Output .= $Self->{LayoutObject}->Footer();

    return $Output;
}

1;
```

_ShowScreen 函数用于设置和定义模板中的 HTML 元素和块，以生成管理对话框 HTML 代码 。

用于管理对话框的动态字段模块示例

模板是存储对话框的 HTML 代码的位置 。

在本节中，将显示和解释 password 动态字段的管理对话框模板 。

```
# --
# Copyright (C) 2001-2020 OTRS AG, https://otrs.com/
# --
# This software comes with ABSOLUTELY NO WARRANTY. For details, see
```

(下页继续)

(续上页)

```
# the enclosed file COPYING for license information (GPL). If you
# did not receive this file, see https://www.gnu.org/licenses/gpl-3.0.txt.
# --
```

这是可在常见 OTRS 模块中找到的通用标头 。

```
<div class="MainBox ARIARoleMain LayoutFixedSidebar SidebarFirst">
    <h1>[% Translate("Dynamic Fields") | html %] - [% Translate(Data.
↪ObjectTypeName) | html %]: [% Translate(Data.Mode) | html %] [%␣
↪Translate(Data.FieldTypeName) | html %] [% Translate("Field") | html %]</h1>

    <div class="Clear"></div>

    <div class="SidebarColumn">
        <div class="WidgetSimple">
            <div class="Header">
                <h2>[% Translate("Actions") | html %]</h2>
            </div>
            <div class="Content">
                <ul class="ActionList">
                    <li>
                        <a href="[% Env("Baselink") %]Action=AdminDynamicField
↪" class="CallForAction"><span>[% Translate("Go back to overview") | html %]
↪</span></a>
                    </li>
                </ul>
            </div>
        </div>
    </div>
```

这部分代码有主框和动作侧栏 。本部分不需要修改 。

```
<div class="ContentColumn">
    <form action="[% Env("CGIHandle") %]" method="post" class="Validate␣
↪PreventMultipleSubmits">
        <input type="hidden" name="Action" value="AdminDynamicFieldPassword" /
↪>
        <input type="hidden" name="Subaction" value="[% Data.Mode | html
↪%]Action" />
        <input type="hidden" name="ObjectType" value="[% Data.ObjectType |␣
↪html %]" />
        <input type="hidden" name="FieldType" value="[% Data.FieldType | html
↪%]" />
        <input type="hidden" name="ID" value="[% Data.ID | html %]" />
```

在这部分代码中定义了对话框的右侧部分 。请注意，`Action` 隐藏输入的值必须与管理对话框的名称匹配 。

```
<div class="WidgetSimple">
    <div class="Header">
        <h2>[% Translate("General") | html %]</h2>
    </div>
    <div class="Content">
```

(下页继续)

(续上页)

```
        <div class="LayoutGrid ColumnsWithSpacing">
            <div class="Size1of2">
                <fieldset class="TableLike">
                    <label class="Mandatory" for="Name"><span class="Marker">*
↪</span> [% Translate("Name") | html %]:</label>
                    <div class="Field">
                        <input id="Name" class="W50pc [% Data.NameServerError␣
↪| html %] [% Data.ShowWarning | html %]  Validate_Alphanumeric" type="text"␣
↪maxlength="200" value="[% Data.Name | html %]" name="Name"/>
                        <div id="NameError" class="TooltipErrorMessage"><p>[%␣
↪Translate("This field is required, and the value should be alphabetic and␣
↪numeric characters only.") | html %]</p></div>
                        <div id="NameServerError" class="TooltipErrorMessage">
↪<p>[% Translate(Data.NameServerErrorMessage) | html %]</p></div>
                        <p class="FieldExplanation">[% Translate("Must be␣
↪unique and only accept alphabetic and numeric characters.") | html %]</p>
                        <p class="Warning Hidden">[% Translate("Changing this␣
↪value will require manual changes in the system.") | html %]</p>
                    </div>
                    <div class="Clear"></div>

                    <label class="Mandatory" for="Label"><span class="Marker">
↪*</span> [% Translate("Label") | html %]:</label>
                    <div class="Field">
                        <input id="Label" class="W50pc [% Data.
↪LabelServerError | html %] Validate_Required" type="text" maxlength="200"␣
↪value="[% Data.Label | html %]" name="Label"/>
                        <div id="LabelError" class="TooltipErrorMessage"><p>[
↪% Translate("This field is required.") | html %]</p></div>
                        <div id="LabelServerError" class="TooltipErrorMessage
↪"><p>[% Translate(Data.LabelServerErrorMessage) | html %]</p></div>
                        <p class="FieldExplanation">[% Translate("This is the␣
↪name to be shown on the screens where the field is active.") | html %]</p>
                    </div>
                    <div class="Clear"></div>

                    <label class="Mandatory" for="FieldOrder"><span class=
↪"Marker">*</span> [% Translate("Field order") | html %]:</label>
                    <div class="Field">
                        [% Data.DynamicFieldOrderSrtg %]
                        <div id="FieldOrderError" class="TooltipErrorMessage">
↪<p>[% Translate("This field is required and must be numeric.") | html %]</p>
↪</div>
                        <div id="FieldOrderServerError" class=
↪"TooltipErrorMessage"><p>[% Translate(Data.FieldOrderServerErrorMessage) |␣
↪html %]</p></div>
                        <p class="FieldExplanation">[% Translate("This is the␣
↪order in which this field will be shown on the screens where is active.") |␣
↪html %]</p>
                    </div>
                    <div class="Clear"></div>
                </fieldset>
```

(下页继续)

(续上页)

```
            </div>
            <div class="Size1of2">
                <fieldset class="TableLike">
                    <label for="ValidID">[% Translate("Validity") | html %]:</
↪label>
                    <div class="Field">
                        [% Data.ValidityStrg %]
                    </div>
                    <div class="Clear"></div>

                    <div class="SpacingTop"></div>
                    <label for="FieldTypeName">[% Translate("Field type") |␣
↪html %]:</label>
                    <div class="Field">
                        <input id="FieldTypeName" readonly="readonly" class=
↪"W50pc" type="text" maxlength="200" value="[% Data.FieldTypeName | html %]"␣
↪name="FieldTypeName"/>
                        <div class="Clear"></div>
                    </div>

                    <div class="SpacingTop"></div>
                    <label for="ObjectTypeName">[% Translate("Object type") |␣
↪html %]:</label>
                    <div class="Field">
                        <input id="ObjectTypeName" readonly="readonly" class=
↪"W50pc" type="text" maxlength="200" value="[% Data.ObjectTypeName | html %]
↪" name="ObjectTypeName"/>
                        <div class="Clear"></div>
                    </div>
                </fieldset>
            </div>
        </div>
    </div>
</div>
```

第一个小部件包含动态字段的通用表单属性。为了与其它动态字段保持一致，建议保持代码的这部分不变。

```
<div class="WidgetSimple">
    <div class="Header">
        <h2>[% Translate(Data.FieldTypeName) | html %] [% Translate("Field␣
↪Settings") | html %]</h2>
    </div>
    <div class="Content">
        <fieldset class="TableLike">

            <label for="DefaultValue">[% Translate("Default value") | html %]:
↪</label>
            <div class="Field">
                <input id="DefaultValue" class="W50pc" type="text" maxlength=
↪"200" value="[% Data.DefaultValue | html %]" name="DefaultValue"/>
                <p class="FieldExplanation">[% Translate("This is the default␣
↪value for this field.") | html %]</p>
```

(下页继续)

(续上页)

```
            </div>
            <div class="Clear"></div>

            <label for="ShowValue">[% Translate("Show value") | html %]:</
↪label>
            <div class="Field">
                [% Data.ShowValueStrg %]
                <p class="FieldExplanation">
                    [% Translate("To reveal the field value in non edit␣
↪screens ( e.g. Ticket Zoom Screen )") | html %]
                </p>
            </div>
            <div class="Clear"></div>

            <label for="ValueMask">[% Translate("Hidden value mask") | html
↪%]:</label>
            <div class="Field">
                <input id="ValueMask" class="W50pc" type="text" maxlength="200
↪" value="[% Data.ValueMask | html %]" name="ValueMask"/>
                <p class="FieldExplanation">
                    [% Translate("This is the alternate value to show if Show␣
↪value is set to \"No\" ( Default: **** ).") | html %]
                </p>
            </div>
            <div class="Clear"></div>

        </fieldset>
    </div>
</div>
```

第二个小部件具有特定于动态字段的表单属性。这是可以设置新属性的地方，它可以使用 Javascript 和 AJAX 技术使最终用户更容易或更友好。

```
            <fieldset class="TableLike">
                <div class="Field SpacingTop">
                    <button type="submit" class="Primary" value="[% Translate(
↪"Save") | html %]">[% Translate("Save") | html %]</button>
                    [% Translate("or") | html %]
                    <a href="[% Env("Baselink") %]Action=AdminDynamicField">[
↪% Translate("Cancel") | html %]</a>
                </div>
                <div class="Clear"></div>
            </fieldset>
        </form>
    </div>
</div>
[% WRAPPER JSOnDocumentComplete %]
<script type="text/javascript">//<![CDATA[
$('.ShowWarning').bind('change keyup', function (Event) {
    $('p.Warning').removeClass('Hidden');
});
```

(下页继续)

(续上页)

```
Core.Agent.Admin.DynamicField.ValidationInit();
//]]></script>
[% END %]
```

文件的最后一部分包含 保存按钮和 取消链接，以及其它需要的 JavaScript 代码 。

动态字段驱动程序示例

驱动程序 即是动态字段 。它包含在 OTRS 框架中广泛使用的几个函数 。驱动程序可以从基类继承一些函数，例如 TextArea 驱动程序从 Base.pm 和 BaseText.pm 继承大多数函数，它只实现需要不同逻辑的函数或结果 。复选框字段驱动程序仅继承自 Base.pm，因为所有其它函数与任何其它基本驱动程序非常不同 。

参见:

请参阅模块 /Kernel/System/DynmicField/Backend.pm 的 Perl 在线文档(POD)，以获得每个函数的所有属性和可能的返回数据的列表 。

在本节中，将显示和解释 password 动态字段驱动程序 。这个驱动程序继承了 Base.pm 和 BaseText.pm 中的一些函数，只实现了需要不同结果的函数 。

```
# --
# Copyright (C) 2001-2020 OTRS AG, https://otrs.com/
# --
# This software comes with ABSOLUTELY NO WARRANTY. For details, see
# the enclosed file COPYING for license information (GPL). If you
# did not receive this file, see https://www.gnu.org/licenses/gpl-3.0.txt.
# --

package Kernel::System::DynamicField::Driver::Password;

use strict;
use warnings;

use Kernel::System::VariableCheck qw(:all);
use Kernel::System::DynamicFieldValue;

use base qw(Kernel::System::DynamicField::Driver::BaseText);

our @ObjectDependencies = (
    'Kernel::Config',
    'Kernel::System::DynamicFieldValue',
    'Kernel::System::Main',
);
```

这是可在常见 OTRS 模块中找到的通用标头 。类/包名称通过 package 关键字声明 。请注意，BaseText 用作基类 。

```
sub new {
    my ( $Type, %Param ) = @_;

    # allocate new hash for object
    my $Self = {};
    bless( $Self, $Type );
```

(下页继续)

(续上页)

```
    # set field behaviors
    $Self->{Behaviors} = {
        'IsACLReducible'                => 0,
        'IsNotificationEventCondition' => 1,
        'IsSortable'                    => 0,
        'IsFiltrable'                   => 0,
        'IsStatsCondition'              => 1,
        'IsCustomerInterfaceCapable'    => 1,
    };

    # get the Dynamic Field Backend custom extensions
    my $DynamicFieldDriverExtensions
        = $Kernel::OM->Get('Kernel::Config')->Get(
↪'DynamicFields::Extension::Driver::Password');

    EXTENSION:
    for my $ExtensionKey ( sort keys %{$DynamicFieldDriverExtensions} ) {

        # skip invalid extensions
        next EXTENSION if !IsHashRefWithData( $DynamicFieldDriverExtensions->{
↪$ExtensionKey} );

        # create a extension config shortcut
        my $Extension = $DynamicFieldDriverExtensions->{$ExtensionKey};

        # check if extension has a new module
        if ( $Extension->{Module} ) {

            # check if module can be loaded
            if (
                !$Kernel::OM->Get('Kernel::System::Main')->RequireBaseClass(
↪$Extension->{Module} )
                )
            {
                die "Can't load dynamic fields backend module"
                    . " $Extension->{Module}! $@";
            }
        }

        # check if extension contains more behaviors
        if ( IsHashRefWithData( $Extension->{Behaviors} ) ) {

            %{ $Self->{Behaviors} } = (
                %{ $Self->{Behaviors} },
                %{ $Extension->{Behaviors} }
            );
        }
    }

    return $Self;
}
```

构造函数 new 创建了一个新的类实例 。根据编码指南，必须在 new 中创建此模块中所需的其它类的对象 。

正确定义行为非常重要，因为在某些屏幕中可能会或可能不会使用该字段，可能不需要实现依赖于对该特定字段不活动的行为的函数 。

注解: 驱动程序仅由 BackendObject 创建，而不是直接从任何其它模块创建 。

```
sub EditFieldRender {
    my ( $Self, %Param ) = @_;

    # take config from field config
    my $FieldConfig = $Param{DynamicFieldConfig}->{Config};
    my $FieldName   = 'DynamicField_' . $Param{DynamicFieldConfig}->{Name};
    my $FieldLabel  = $Param{DynamicFieldConfig}->{Label};

    my $Value = '';

    # set the field value or default
    if ( $Param{UseDefaultValue} ) {
        $Value = ( defined $FieldConfig->{DefaultValue} ? $FieldConfig->
↪{DefaultValue} : '' );
    }
    $Value = $Param{Value} if defined $Param{Value};

    # extract the dynamic field value from the web request
    my $FieldValue = $Self->EditFieldValueGet(
        %Param,
    );

    # set values from ParamObject if present
    if ( defined $FieldValue ) {
        $Value = $FieldValue;
    }

    # check and set class if necessary
    my $FieldClass = 'DynamicFieldText W50pc';
    if ( defined $Param{Class} && $Param{Class} ne '' ) {
        $FieldClass .= ' ' . $Param{Class};
    }

    # set field as mandatory
    $FieldClass .= ' Validate_Required' if $Param{Mandatory};

    # set error css class
    $FieldClass .= ' ServerError' if $Param{ServerError};

    my $HTMLString = <<"EOF";
<input type="password" class="$FieldClass" id="$FieldName" name="$FieldName"␣
↪title="$FieldLabel" value="$Value" />
EOF

    if ( $Param{Mandatory} ) {
```

(下页继续)

(续上页)

```
        my $DivID = $FieldName . 'Error';

        # for client side validation
        $HTMLString .= <<"EOF";
    <div id="$DivID" class="TooltipErrorMessage">
        <p>
            \$Text{"This field is required."}
        </p>
    </div>
EOF
    }

    if ( $Param{ServerError} ) {

        my $ErrorMessage = $Param{ErrorMessage} || 'This field is required.';
        my $DivID = $FieldName . 'ServerError';

        # for server side validation
        $HTMLString .= <<"EOF";
    <div id="$DivID" class="TooltipErrorMessage">
        <p>
            \$Text{"$ErrorMessage"}
        </p>
    </div>
EOF
    }

    # call EditLabelRender on the common Driver
    my $LabelString = $Self->EditLabelRender(
        %Param,
        DynamicFieldConfig => $Param{DynamicFieldConfig},
        Mandatory          => $Param{Mandatory} || '0',
        FieldName          => $FieldName,
    );

    my $Data = {
        Field => $HTMLString,
        Label => $LabelString,
    };

    return $Data;
}
```

此函数负责创建字段及其标签的 HTML 呈现，并在编辑屏幕（ 如 AgentTicketPhone 、AgentTicket-Note 等 ）中使用 。

```
sub DisplayValueRender {
    my ( $Self, %Param ) = @_;

    # set HTMLOutput as default if not specified
    if ( !defined $Param{HTMLOutput} ) {
        $Param{HTMLOutput} = 1;
```

(下页继续)

(续上页)

```
    }

    my $Value;
    my $Title;

    # check if field is set to show password or not
    if (
        defined $Param{DynamicFieldConfig}->{Config}->{ShowValue}
        && $Param{DynamicFieldConfig}->{Config}->{ShowValue} eq 'Yes'
        )
    {

        # get raw Title and Value strings from field value
        $Value = defined $Param{Value} ? $Param{Value} : '';
        $Title = $Value;
    }
    else {

        # show the mask and not the value
        $Value = $Param{DynamicFieldConfig}->{Config}->{ValueMask} || '';
        $Title = 'The value of this field is hidden.'
    }

    # HTMLOutput transformations
    if ( $Param{HTMLOutput} ) {
        $Value = $Param{LayoutObject}->Ascii2Html(
            Text => $Value,
            Max => $Param{ValueMaxChars} || '',
        );

        $Title = $Param{LayoutObject}->Ascii2Html(
            Text => $Title,
            Max => $Param{TitleMaxChars} || '',
        );
    }
    else {
        if ( $Param{ValueMaxChars} && length($Value) > $Param{ValueMaxChars}␣
↪) {
            $Value = substr( $Value, 0, $Param{ValueMaxChars} ) . '...';
        }
        if ( $Param{TitleMaxChars} && length($Title) > $Param{TitleMaxChars}␣
↪) {
            $Title = substr( $Title, 0, $Param{TitleMaxChars} ) . '...';
        }
    }

    # create return structure
    my $Data = {
        Value => $Value,
        Title => $Title,
    };
```

(下页继续)

(续上页)

```
    return $Data;
}
```

DisplayValueRender() 函数返回字段值作为纯文本及其标题(两者都可以翻译)。对于此特定示例，我们检查是否应显示密码或通过动态字段中的配置参数显示预定义的掩码 。

```
sub ReadableValueRender {
    my ( $Self, %Param ) = @_;

    my $Value;
    my $Title;

    # check if field is set to show password or not
    if (
        defined $Param{DynamicFieldConfig}->{Config}->{ShowValue}
        && $Param{DynamicFieldConfig}->{Config}->{ShowValue} eq 'Yes'
        )
    {

        # get raw Title and Value strings from field value
        $Value = $Param{Value} // '';
        $Title = $Value;
    }
    else {

        # show the mask and not the value
        $Value = $Param{DynamicFieldConfig}->{Config}->{ValueMask} || '';
        $Title = 'The value of this field is hidden.'
    }

    # cut strings if needed
    if ( $Param{ValueMaxChars} && length($Value) > $Param{ValueMaxChars} ) {
        $Value = substr( $Value, 0, $Param{ValueMaxChars} ) . '...';
    }
    if ( $Param{TitleMaxChars} && length($Title) > $Param{TitleMaxChars} ) {
        $Title = substr( $Title, 0, $Param{TitleMaxChars} ) . '...';
    }

    # create return structure
    my $Data = {
        Value => $Value,
        Title => $Title,
    };

    return $Data;
}
```

此函数类似于 DisplayValueRender() 但是用于没有 LayoutObject 的地方 。

其它函数

如果新动态字段不从其它类继承，则以下是可能需要的其它函数。要查看此函数的完整代码，请直接查看文件 Kernel/System/DynamicField/Driver/Base.pm 和 Kernel/System/DynamicField/Driver/BaseText.pm。

```
sub ValueGet { ... }
```

此函数从指定对象上的字段中检索值。在这种情况下，我们返回第一个文本值，因为该字段仅存储一个文本值。

```
sub ValueSet { ... }
```

此函数用于存储动态字段值。在这种情况下，该字段仅存储一个文本类型值。其它字段可以用 ValueText、ValueDateTime 或 ValueInt 格式存储多个值。

```
sub ValueDelete { ... }
```

此函数用于删除附加到特定对象 ID 的一个字段值。例如，如果要删除对象的实例，则没有理由将该字段值存储在该特定对象实例的数据库中。

```
sub AllValuesDelete { ... }
```

此函数用于删除某个动态字段中的所有值。要删除动态字段时，此功能非常有用。

```
sub ValueValidate { ... }
```

此函数用于检查值是否与其类型一致。

```
sub SearchSQLGet { ... }
```

TicketSearch 核心模块使用此函数构建内部查询，以基于此字段作为搜索参数搜索工单。

```
sub SearchSQLOrderFieldGet { ... }
```

此函数也是 TicketSearch 模块的一个 helper。$Param{TableAlias} 应该保留，value_text 可以用 value_date 或 value_int 替换，具体取决于字段。

```
sub EditFieldValueGet { ... }
```

此函数用于 OTRS 的编辑屏幕，其目的是从自动任务配置文件或 Web 请求等模板获取字段的值。此函数在 $Param{ParamObject} 中获取 Web 请求，它是前端模块或屏幕的 ParamObject 的副本。

此函数有两种返回格式。通常只是字段名称 => 字段值对的原始值或结构。例如，日期动态字段通常将日期作为字符串返回，如果它应返回结构，则它会为散列中的日期的每个部分返回一对。

如果结果应该是结构，那么通常这用于将其值存储在模板中，如自动任务配置文件。例如，日期字段使用多个 HTML 组件来构建字段，如使用的复选框，并选择年、月、日等。

```
sub EditFieldValueValidate { ... }
```

此函数应至少提供一种方法来验证字段是否为空，如果字段为空且是必需的，则返回错误，但它也可以对其它类型的字段执行更多验证，例如，如果选择的选项有效，或者日期是否应仅在过去等。它也可以提供自定义错误消息。

```
sub SearchFieldRender { ... }
```

工单搜索对话框使用此函数，它类似于 EditFieldRander()，但通常在搜索屏幕上，必须对所有字段进行小的更改。对于此示例，我们使用 HTML 文本输入而不是密码输入。在其它字段中，下拉字段显示为多选，以便让用户一次搜索多个值。

```
sub SearchFieldValueGet { ... }
```

非常类似于 EditFieldValueGet()，但使用了不同的名称前缀，适用于搜索对话框屏幕。

```
sub SearchFieldParameterBuild { ... }
```

工单搜索对话框也使用此函数来设置正确的运算符和值，以便在此字段上进行搜索。它还返回值应在结果页面中使用的搜索属性中显示的方式。

```
sub StatsFieldParameterBuild { ... }
```

统计模块使用此函数。它包括统计信息格式的字段定义。对于具有固定值的字段，它还包括所有这些可能的值，如果它们可以被转换，请查看 BaseSelect 驱动程序代码，以获取如何实现它们的示例。

```
sub StatsSearchFieldParameterBuild { ... }
```

这个函数非常类似于 SearchFieldParameterBuild()。不同之处在于后者从搜索配置文件中获取值，并且此值直接从其参数中获取值。

此函数用于统计模块。

```
sub TemplateValueTypeGet { ... }
```

此函数用于了解如何检索存储在配置文件中的动态字段值，如标量或数组，还可以定义配置文件中字段的正确名称。

```
sub RandomValueSet { ... }
```

otrs.FillDB.pl 脚本使用此函数用一些测试和随机数据填充数据库。此函数插入的值并不真实相关。唯一的限制是该值必须与字段值类型兼容。

```
sub ObjectMatch { ... }
```

由通知模块使用。如果字段出现在 $Param{ObjectAttributes} 参数中并且它与给定值匹配，则此函数返回 1。

3.3.27 创建一个动态字段功能扩展

为了说明这个过程，Foo 函数的一个新的动态字段功能扩展将添加到后端对象和文本字段驱动程序中。

要创建此扩展，我们将创建 3 个文件：

1. 一个用于注册模块的配置文件（ XML ）。
2. 一个用于定义新函数的后端扩展（ Perl ）。
3. 一个文本字段驱动程序扩展（ Perl ），用于实现文本字段的新功能。

文件结构：

```
$HOME (e. g. /opt/otrs/)
|
...
|--/Kernel/
|    |--/Config/
|    |    |--/Files/
|    |    |    |--/XML/
|    |    |    |DynamicFieldFooExtension.xml
...
|    |--/System/
|    |    |--/DynamicField/
|    |    |    FooExtensionBackend.pm
|    |    |    |--/Driver/
|    |    |    |    |FooExtensionText.pm
...
```

动态字段 **Foo** 扩展文件

动态字段扩展文件示例

该配置文件用于注册后端和驱动程序的扩展以及每个驱动程序的新行为 。

注解: 如果使用新功能扩展一个驱动程序，则后端也需要一个该功能的扩展 。

在本节中，将显示和解释 Foo 扩展的配置文件 。

```
<?xml version="1.0" encoding="utf-8" ?>
<otrs_config version="2.0" init="Application">
```

这是配置文件的正常头文件 。

```
<ConfigItem Name="DynamicFields::Extension::Backend###100-Foo" Required="0"␣
↪Valid="1">
    <Description Translatable="1">Dynamic Fields Extension.</Description>
    <Group>DynamicFieldFooExtension</Group>
    <SubGroup>DynamicFields::Extension::Registration</SubGroup>
    <Setting>
        <Hash>
            <Item Key="Module">
↪Kernel::System::DynamicField::FooExtensionBackend</Item>
        </Hash>
    </Setting>
</ConfigItem>
```

此设置在后端对象中注册扩展名 。该模块将从 Backend 作为基类加载 。

```
<ConfigItem Name="DynamicFields::Extension::Driver::Text###100-Foo" Required="0
↪" Valid="1">
    <Description Translatable="1">Dynamic Fields Extension.</Description>
    <Group>DynamicFieldFooExtension</Group>
    <SubGroup>DynamicFields::Extension::Registration</SubGroup>
```

(下页继续)

(续上页)

```
    <Setting>
        <Hash>
            <Item Key="Module">
↪Kernel::System::DynamicField::Driver::FooExtensionText</Item>
            <Item Key="Behaviors">
                <Hash>
                    <Item Key="Foo">1</Item>
                </Hash>
            </Item>
        </Hash>
    </Setting>
</ConfigItem>
```

这是在文本动态字段驱动程序中注册扩展。该模块将作为基类加载到驱动程序中。另请注意，可以指定新行为。这些扩展的行为将被添加到驱动程序开箱即用的行为中，因此调用 `HasBehavior()` 来检查这些新行为将是完全透明的。

```
</otrs_config>
```

一个配置文件的标准关闭。

动态字段后端扩展示例

后端扩展将作为基类透明地加载到后端本身。可以在扩展中访问后端的所有已定义对象和属性。

注解: 后端扩展中定义的所有新功能都应在驱动程序扩展中实现。

在本节中，将显示和解释后端的 `Foo` 扩展。该扩展只定义函数 `Foo()`。

```
# --
# Copyright (C) 2001-2020 OTRS AG, https://otrs.com/
# --
# This software comes with ABSOLUTELY NO WARRANTY. For details, see
# the enclosed file COPYING for license information (GPL). If you
# did not receive this file, see https://www.gnu.org/licenses/gpl-3.0.txt.
# --

package Kernel::System::DynamicField::FooExtensionBackend;

use strict;
use warnings;

use Kernel::System::VariableCheck qw(:all);

=head1 NAME

Kernel::System::DynamicField::FooExtensionBackend

=head1 SYNOPSIS
```

(下页继续)

(续上页)

```
DynamicFields Extension for Backend

=head1 PUBLIC INTERFACE

=over 4

=cut
```

这是可在常见 OTRS 模块中找到的通用标头。类/包名称通过 `package` 关键字声明。

```
=item Foo()

Testing function: returns 1 if function is available on a Dynamic Field␣
↪driver.

    my $Success = $BackendObject->Foo(
        DynamicFieldConfig   => $DynamicFieldConfig,       # complete config of␣
↪the DynamicField
    );

Returns:
    $Success = 1;       # or undef

=cut

sub Foo {
    my ( $Self, %Param ) = @_;

    # check needed stuff
    for my $Needed (qw(DynamicFieldConfig)) {
        if ( !$Param{$Needed} ) {
            $Kernel::OM->Get('Kernel::System::Log')->Log(
                Priority => 'error',
                Message  => "Need $Needed!",
            );

            return;
        }
    }

    # check DynamicFieldConfig (general)
    if ( !IsHashRefWithData( $Param{DynamicFieldConfig} ) ) {
        $Kernel::OM->Get('Kernel::System::Log')->Log(
            Priority => 'error',
            Message  => "The field configuration is invalid",
        );

        return;
    }

    # check DynamicFieldConfig (internally)
    for my $Needed (qw(ID FieldType ObjectType)) {
```

(下页继续)

(续上页)

```
        if ( !$Param{DynamicFieldConfig}->{$Needed} ) {
            $Kernel::OM->Get('Kernel::System::Log')->Log(
                Priority => 'error',
                Message  => "Need $Needed in DynamicFieldConfig!",
            );

            return;
        }
    }

    # set the dynamic field specific backend
    my $DynamicFieldBackend = 'DynamicField' . $Param{DynamicFieldConfig}->
↪{FieldType} . 'Object';

    if ( !$Self->{$DynamicFieldBackend} ) {
        $Kernel::OM->Get('Kernel::System::Log')->Log(
            Priority => 'error',
            Message  => "Backend $Param{DynamicFieldConfig}->{FieldType} is␣
↪invalid!",
        );

        return;
    }

    # verify if function is available
    return if !$Self->{$DynamicFieldBackend}->can('Foo');

    # call HasBehavior on the specific backend
    return $Self->{$DynamicFieldBackend}->Foo(%Param);
}
```

`Foo()` 函数仅用于测试目的。首先，它检查动态字段配置，然后检查动态字段驱动程序（类型）是否存在且是否已加载。为了防止对未定义的驱动程序的函数调用，首先检查驱动程序是否可以执行该函数，然后在驱动程序中执行传递所有参数的函数。

注解：也可以跳过测试驱动程序是否可以执行该功能的步骤。为此，有必要在前端模块中实现一种机制，以要求在动态字段上有一个特殊的行为，并且仅在调用后端对象中的函数之后。

动态字段驱动程序扩展示例

驱动程序扩展将作为基类透明地加载到驱动程序本身中。驱动程序中定义的所有对象和属性都可以在扩展中访问。

注解：驱动程序扩展中实现的所有新函数都应该在后端扩展中定义，因为每个函数都是从后端对象调用的。

在本节中，将显示并解释文本字段驱动程序的 `Foo` 扩展。扩展仅实现 `foo()` 函数。

```
# --
# Copyright (C) 2001-2020 OTRS AG, https://otrs.com/
# --
# This software comes with ABSOLUTELY NO WARRANTY. For details, see
# the enclosed file COPYING for license information (GPL). If you
# did not receive this file, see https://www.gnu.org/licenses/gpl-3.0.txt.
# --

package Kernel::System::DynamicField::Driver::FooExtensionText;

use strict;
use warnings;

=head1 NAME

Kernel::System::DynamicField::Driver::FooExtensionText

=head1 SYNOPSIS

DynamicFields Text Driver Extension

=head1 PUBLIC INTERFACE

This module extends the public interface of L
↪<Kernel::System::DynamicField::Backend>.
Please look there for a detailed reference of the functions.

=over 4

=cut
```

这是可在常见 OTRS 模块中找到的通用标头。类/包名称通过 `package` 关键字声明。

```
sub Foo {
    my ( $Self, %Param ) = @_;
    return 1;
}
```

`foo()` 函数没有特殊的逻辑。它只用于测试，并且总是返回 1。

3.3.28 工单邮箱管理员模块

邮箱管理员模块在邮箱管理员流程中使用。邮箱管理员模块有两种：

- `PostMasterPre`：解析邮件后使用。
- `PostMasterPost`：在处理完电子邮件并保存到数据库后使用。

如果您想创建自己的邮箱管理员过滤器，只需创建自己的模块。这些模块位于 `Kernel/System/PostMaster/Filter/*.pm` 下。有关默认模块，请参阅管理员手册。你只需要两个函数：`new()` 和 `Run()`。

以下是匹配电子邮件和设置 X-OTRS-Headers 的示例模块（有关详细信息，请参阅 `doc/X-OTRS-Headers.txt`）。

Kernel/Config/Files/XML/MyModule.xml:

```
<ConfigItem Name="PostMaster::PreFilterModule###1-Example" Required="0" Valid=
↪"1">
    <Description Translatable="1">Example module to filter and manipulate␣
↪incoming messages.</Description>
    <Group>Ticket</Group>
    <SubGroup>Core::PostMaster</SubGroup>
    <Setting>
        <Hash>
            <Item Key="Module">Kernel::System::PostMaster::Filter::Example</
↪Item>
            <Item Key="Match">
                <Hash>
                    <Item Key="From">noreply@</Item>
                </Hash>
            </Item>
            <Item Key="Set">
                <Hash>
                    <Item Key="X-OTRS-Ignore">yes</Item>
                </Hash>
            </Item>
        </Hash>
    </Setting>
</ConfigItem>
```

以及 Kernel/System/PostMaster/Filter/Example.pm 中的实际过滤器代码：

```
# --
# Copyright (C) 2001-2020 OTRS AG, https://otrs.com/
# --
# This software comes with ABSOLUTELY NO WARRANTY. For details, see
# the enclosed file COPYING for license information (GPL). If you
# did not receive this file, see https://www.gnu.org/licenses/gpl-3.0.txt.
# --

package Kernel::System::PostMaster::Filter::Example;

use strict;
use warnings;

our @ObjectDependencies = (
    'Kernel::System::Log',
);

sub new {
    my ( $Type, %Param ) = @_;

    # allocate new hash for object
    my $Self = {};
    bless ($Self, $Type);

    $Self->{Debug} = $Param{Debug} || 0;
```

(下页继续)

(续上页)

```
    return $Self;
}

sub Run {
    my ( $Self, %Param ) = @_;
    # get config options
    my %Config = ();
    my %Match = ();
    my %Set = ();
    if ($Param{JobConfig} && ref($Param{JobConfig}) eq 'HASH') {
        %Config = %{$Param{JobConfig}};
        if ($Config{Match}) {
            %Match = %{$Config{Match}};
        }
        if ($Config{Set}) {
            %Set = %{$Config{Set}};
        }
        }
        # match 'Match => ???' stuff
        my $Matched = '';
        my $MatchedNot = 0;
        for (sort keys %Match) {
        if ($Param{GetParam}->{$_} && $Param{GetParam}->{$_} =~ /$Match{$_}/
↪i) {
            $Matched = $1 || '1';
            if ($Self->{Debug} > 1) {
                $Kernel::OM->Get('Kernel::System::Log')->Log(
                    Priority => 'debug',
                    Message => "'$Param{GetParam}->{$_}' =~ /$Match{$_}/i␣
↪matched!",
                );
            }
        }
        else {
            $MatchedNot = 1;
            if ($Self->{Debug} > 1) {
                $Kernel::OM->Get('Kernel::System::Log')->Log(
                    Priority => 'debug',
                    Message => "'$Param{GetParam}->{$_}' =~ /$Match{$_}/i␣
↪matched NOT!",
                );
            }
        }
        }
        # should I ignore the incoming mail?
        if ($Matched && !$MatchedNot) {
        for (keys %Set) {
            if ($Set{$_} =~ /\[\*\*\*\]/i) {
                $Set{$_} = $Matched;
            }
            $Param{GetParam}->{$_} = $Set{$_};
```

(下页继续)

(续上页)

```
            $Kernel::OM->Get('Kernel::System::Log')->Log(
                Priority => 'notice',
                Message => "Set param '$_' to '$Set{$_}' (Message-ID: $Param
↪{GetParam}->{'Message-ID'}) ",
            );
        }
    }

    return 1;
}

1;
```

下图显示了电子邮件处理流程 。

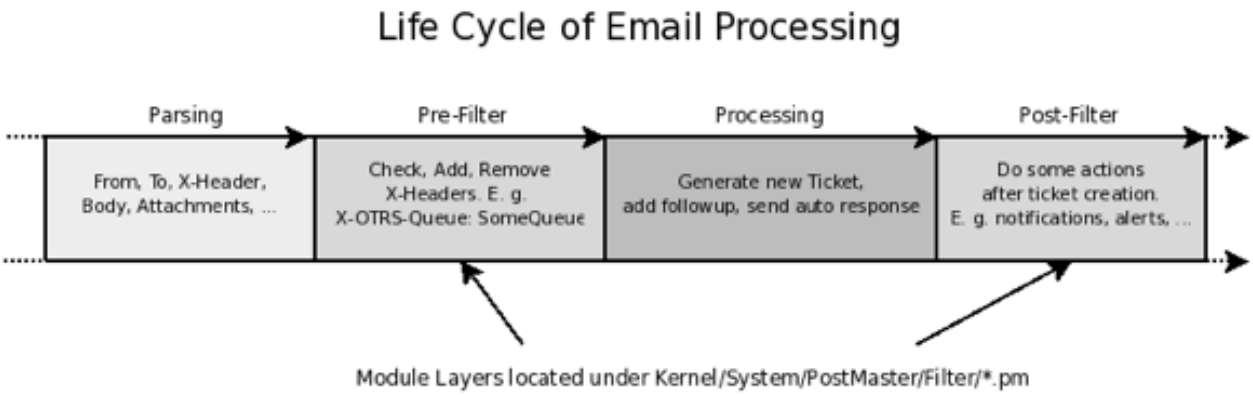

图 6: 电子邮件处理流程

3.3.29 流程管理

由于 OTRS 7 流程管理可以使用脚本模块来执行活动(脚本任务)和/或序列流操作(之前称为转换操作)。这些模块位于 `Kernel::System::ProcessManagement::Modules` 中 。

流程管理模块

流程管理模块是用 Perl 语言编写的脚本，用于执行某些操作或基于流程工单的操作(如设置动态字段或更改队列)。或者到系统的任何其它部分，比如创建新工单 。

默认情况下，模块使用一组键值对将它们用作操作的参数，例如：要更改流程工单的队列，需要队列或队列 ID 及其对应的值 。

某些脚本可能需要多个简单的键值对作为参数，或者其配置可能需要具有更加用户友好的 GUI 。在这种情况下，OTRS 提供了一些配置字段类型，如果需要也可以扩展 。

当前字段类型：

下拉选择框 显示包含预定义值的下拉列表框 。

键-值列表 显示简单键值对(文本输入)的列表 。可以添加或删除键值对 。

多语言富文本 显示与系统语言关联的富文本编辑器，还显示语言选择器，以便为选择的特定语言添加富文本字段 。

收件人 显示一个多选字段，其中预先填充了用作电子邮件收件人的服务人员，还显示一个自由输入字段，用于指定要添加到收件人列表中的外部电子邮件地址 。

富文本 显示单个富文本字段 。

创建新的流程管理模块

下面是如何创建流程管理模块的示例 。它包括一个部分，其中所有可能的字段都被定义为一个引用 。要创建新模块，只需要一种字段类型，但考虑到按照惯例，用户 ID 参数用于模拟模块中的所有操作，而不是触发操作的用户 。然后，最好始终包含键值列表字段类型以及任何其它所需字段 。

流程管理模块代码示例

以下代码实现了一个虚拟的流程管理模块，可用于脚本任务活动或序列流操作 。

```
# --
# Copyright (C) 2001-2020 OTRS AG, https://otrs.com/
# --
# This software comes with ABSOLUTELY NO WARRANTY. For details, see
# the enclosed file COPYING for license information (GPL). If you
# did not receive this file, see https://www.gnu.org/licenses/gpl-3.0.txt.
# --

package Kernel::System::ProcessManagement::Modules::Test;

use strict;
use warnings;
use utf8;

use Kernel::System::VariableCheck qw(:all);

use parent qw(Kernel::System::ProcessManagement::Modules::Base);

our @ObjectDependencies = ( );
```

这是可在常见 OTRS 模块中找到的通用标头 。类/包名称通过 `package` 关键字声明 。

在本例中我们继承自 `Base` 类，并设置了对象管理器依赖项 。

```
sub new {
    my ( $Type, %Param ) = @_;

    # Allocate new hash for object.
    my $Self = {};
    bless( $Self, $Type );

    $Self->{Config} = {
        Description => 'Description for the module.',
        ConfigSets  => [
            {
```

(下页继续)

(续上页)

```
            Type        => 'Dropdown',
            Description => 'Description for Dropdown.',
            Mandatory   => 1,
            Config      => {
                Data => {
                    Internal1 => 'Display 1',
                    Internal2 => 'Display 2',
                },
                Name         => 'Test-DropDown',
                Multiple     => 1,
                PossibleNone => 1,
                Sort         => 'AlphanumericValue',
                Translation  => 1,
            },
        },
        {
            Type        => 'KeyValueList',
            Description => 'Description of the Key/Value pairs',
            Defaults    => [
                {
                    Key       => 'Test-Param1',
                    Value     => 'Hello',
                    Mandatory => 1,
                },
                {
                    Key       => 'Test-Param2',
                    Mandatory => 1,
                },
                {
                    Key       => 'Test-Param3',
                    Value     => 'World',
                    Mandatory => 0,
                },
            ],
        },
        {
            Type        => 'MultiLanguageRichText',
            Description => "Description for Mutli-Language Rich Text.",
            Defaults    => [
                {
                    Key   => 'en_Subject',
                    Value => 'Hello',
                },
                {
                    Key   => 'en_Body',
                    Value => 'World',
                },

            ],
        },
        {
            Type        => 'Recipients',
```

(下页继续)

(续上页)

```
                Description => "Description for Recipients."
            },
            {
                Type        => 'RichText',
                Description => "Description for Rich Text.",
                Defaults    => [
                    {
                        Value     => 'Hello World',
                        Mandatory => 1,
                    },
                ],
            },
        ],
    };

    return $Self;
}
```

构造函数 `new()` 创建一个新的类实例。配置字段在这里定义，它们在 `$Self->{Config}` 中设置。

该配置有两个主要条目：

Description 这用于向管理员解释模块执行的操作和/或其配置注意事项。

ConfigSets 这只是实际配置字段的容器。

所有配置字段都需要一个定义字段类型的类型，并且它们还可以具有内部描述以用作字段小部件的标题。如果未定义，则使用默认描述。

每个字段定义其配置参数和功能。以下是 OTRS 提供的开箱即用的字段的小参考。

- `Dropdown`

  Mandatory 用于定义是否必须设置值。

  Config 保存信息以显示下拉选择框字段。

  Data 简单哈希定义下拉选择框的选项。键在内部使用，值是用户在屏幕中看到的选项。

  Name 参数的名称。

  Multiple 定义是否只能选择一个或多个值。

  PossibleNone 定义值列表是否提供空值。

  Sort 定义在渲染字段时如何对选项进行排序。可能的值是：`AlphanumericValue`、`NumericValue`、`AlphanumericKey` 和 `NumericKey`。

  Translation: 设置是否应翻译显示的值。

- `KeyValueList`

  Defaults 包含其键值对的默认配置的哈希数组。

  Key 参数的名称。

  Value 参数的默认值（可选）。

  Mandatory 不能重命名或删除必填参数（可选）。

- `MultiLanguageRichText`

Defaults 哈希数组，包含每个语言和字段部分的默认配置 。

Key 它由 en 或 es_MX 等语言组成，后跟一个 _（ 下划线字符 ），然后是 Subject 或 Body，用于字段相应的部分 。

Value 字段部分的默认值（ 可选 ）。

- Recipients

  没有为这种字段提供进一步的配置 。

- RichText

  Defaults 保存默认配置字段的哈希数组（ 仅使用第一个元素 ）。

  Value 该字段的默认值 。

  Mandatory 用于定义是否必须设置值 。

```
sub Run {
    my ( $Self, %Param ) = @_;

    # Define a common message to output in case of any error.
    my $CommonMessage = "Process: $Param{ProcessEntityID} Activity: $Param
↪{ActivityEntityID}";

    # Add SequenceFlowEntityID to common message if available.
    if ( $Param{SequenceFlowEntityID} ) {
        $CommonMessage .= " SequenceFlow: $Param{SequenceFlowEntityID}";
    }

    # Add SequenceFlowActionEntityID to common message if available.
    if ( $Param{SequenceFlowActionEntityID} ) {
        $CommonMessage .= " SequenceFlowAction: $Param
↪{SequenceFlowActionEntityID}";
    }

    # Check for missing or wrong params.
    my $Success = $Self->_CheckParams(
        %Param,
        CommonMessage => $CommonMessage,
    );
    return if !$Success;

    # Override UserID if specified as a parameter in the TA config.
    $Param{UserID} = $Self->_OverrideUserID(%Param);

    # Use ticket attributes if needed.
    $Self->_ReplaceTicketAttributes(%Param);
    $Self->_ReplaceAdditionalAttributes(%Param);

    # Get module configuration.
    my $ModuleConfig = $Param{Config};

    # Add module logic here!

    return 1;
}
```

Run 方法是模块的主要部分。首先设置可以在错误日志或任何其它目的中使用的公共消息。为了保持一致性，强烈建议如上所述使用它。

下一步是检查是否正确发送了全局参数。

按照惯例，所有模块都应该能够覆盖参数中提供的当前用户 ID（如果有的话）。此传递的用户 ID 应该在需要它的任何函数调用中使用。

用户定义的属性值可以使用 OTRS 智能标签来使用当前工单值。_ReplaceTicketAttributes 用于普通文本属性，而 _ReplaceAdditionalAttributes 用于富文本。对于更复杂的参数，可能需要自定义函数来替换此智能标签。

以下是该模块的正确逻辑。

如果一切正常，则必须返回 1。

创建新的流程管理模块配置字段

以下是如何创建流程管理模块配置字段的示例。任何流程管理模块在配置后都可以使用该字段。

流程管理模块配置字段代码示例

以下代码实现了一个简单的输入流程管理模块配置字段（test）。可以通过流程管理模块 ConfigSets 设置字段的名称及其默认值。

```
# --
# Copyright (C) 2001-2020 OTRS AG, https://otrs.com/
# --
# This software comes with ABSOLUTELY NO WARRANTY. For details, see
# the enclosed file COPYING for license information (GPL). If you
# did not receive this file, see https://www.gnu.org/licenses/gpl-3.0.txt.
# --

package Kernel::Output::HTML::ProcessManagement::ModuleConfiguration::Test;

use strict;
use warnings;

use Kernel::System::VariableCheck qw(:all);
use Kernel::Language qw(Translatable);

our @ObjectDependencies = (
    'Kernel::Output::HTML::Layout',
    'Kernel::System::Web::Request',
);
```

这是可在常见 OTRS 模块中找到的通用标头。类/包名称通过 package 关键字声明。

```
sub new {
    my ( $Type, %Param ) = @_;

    # allocate new hash for object
    my $Self = {%Param};
    bless( $Self, $Type );
```

(下页继续)

(续上页)

```
    return $Self;
}
```

构造函数 new 创建了一个新的类实例 。

每个配置字段都需要实现至少 2 个主要方法：Render 和 GetParams 。

```
sub Render {
    my ( $Self, %Param ) = @_;

    my $ConfigSet = $Param{ConfigSet} // {};
    my $EntityConfig = $Param{EntityData}->{Config}->{Config}->{ConfigTest} // {};

    my %Data;

    $Data{Description} = $ConfigSet->{Description} || 'Config Parameters (Test)
↪';
    $Data{Name} = $ConfigSet->{Config}->{Name} // 'Test';
    $Data{Value} = $EntityConfig->{ $ConfigSet->{Config}->{Name} } // $ConfigSet->
↪{Defaults}->[0]->{Value} // '';

    return {
        Success => 1,
        HTML    => $Kernel::OM->Get('Kernel::Output::HTML::Layout')->Output(
            TemplateFile => 'ProcessManagement/ModuleConfiguration/Test',
            Data         => \%Data,
        ),
    };
}
```

Render 方法负责为字段创建所需的 HTML 。

在此示例中，它首先本地化一些参数，以便更容易地读取和维护 。

以下行设置要显示的数据 。如果定义了字段小部件标题 Description 则从 ConfigSet 收集，否则它使用默认文本 。类似于字段 Name，对于 Value，它首先检查活动或序列流操作是否已经存储了值，如果没有，它会尝试使用 ConfigSet 中的默认值，否则使用空 。

最后，它返回一个结构，其中包含填充了所收集数据的模板中的 HTML 代码 。

```
sub GetParams {
    my ( $Self, %Param ) = @_;

    my %GetParams;
    my $Config = $Param{ConfigSet}->{Config} // 'Test';

    $GetParams{ $Config->{Name} } = $Kernel::OM->Get(
↪'Kernel::System::Web::Request')->GetParam( Param => $Config->{Name} );

    return \%GetParams;
}
```

对于这个例子，GetParams 方法非常简单 。它从 ConfigSet 获取字段的名称或使用默认值，并从 Web 请求中获取值 。

流程管理模块配置字段模板示例

以下代码实现了测试流程管理模块配置字段的基本 HTML 模板 。

```
# --
# Copyright (C) 2001-2020 OTRS AG, https://otrs.com/
# --
# This software comes with ABSOLUTELY NO WARRANTY. For details, see
# the enclosed file COPYING for license information (GPL). If you
# did not receive this file, see https://www.gnu.org/licenses/gpl-3.0.txt.
# --
```

这是可在常见 OTRS 模块中找到的通用标头 。

```
<div id="TestConfig" class="WidgetSimple Expanded">
    <div class="Header">
        <div class="WidgetAction Toggle">
            <a href="#" title="[% Translate("Show or hide the content") |␣
↪html %]"><i class="fa fa-caret-right"></i><i class="fa fa-caret-down"></i></
↪a>
        </div>
        <h2 for="Config">[% Translate(Data.Description) | html %]</h2>
    </div>
    <div class="Content" id="TestParams">
        <fieldset class="TableLike">
            <label for="[% Data.Name | html %]">[% Data.Name | html %]</label>
            <div class="Field">
                <input type="text" value="[% Data.Value | html %]" name="[%␣
↪Data.Name | html %]" id="[% Data.Name | html %]" />
            </div>
        </fieldset>
    </div>
</div>
```

模板显示一个简单的文本输入元素及其关联的标签 。

CHAPTER 4

如何发布你的 OTRS 扩展

4.1 软件包管理

OPM（ OTRS 包管理器 ）是一种通过 HTTP 、FTP 或文件上载为 OTRS 框架分发软件包的机制 。

例如，OTRS 项目通过我们的 FTP 服务器上的在线软件仓库提供 OTRS 模块，如 OTRS 软件包中的日历 、文件管理器或 Web 邮件 。可以通过管理员界面管理（ 安装 、升级和卸载 ）软件包 。

4.1.1 软件包分发

如果您想创建一个 OPM 在线软件仓库，只需通过激活系统配置设置 `Package::RepositoryList` 并在那里添加新位置，告诉 OTRS 框架该位置 。然后，您将在软件包管理器中有一个新的选择选项 。

在您的存储库中，为您的 OPM 软件包创建索引文件 。OTRS 只读取此索引文件来获取可用的软件包 。

```
shell> bin/otrs.Console.pl Dev::Package::RepositoryIndex /path/to/repository/␣
↪> /path/to/repository/otrs.xml
```

4.1.2 软件包命令

您可以通过管理员界面或 `bin/otrs.Console.pl` 来使用以下 OPM 命令来管理 OPM 软件包的管理任务 。

安装

安装 OPM 软件包 。

- Web: http://localhost/otrs/index.pl?Action=AdminPackageManager
- CMD: `bin/otrs.Console.pl Admin::Package::Install /path/to/package.opm`

卸载

卸载 OPM 软件包 。

- Web: http://localhost/otrs/index.pl?Action=AdminPackageManager
- CMD: `bin/otrs.Console.pl Admin::Package::Uninstall /path/to/package.opm`

升级

升级 OPM 软件包 。

- Web: http://localhost/otrs/index.pl?Action=AdminPackageManager
- CMD: `bin/otrs.Console.pl Admin::Package::Upgrade /path/to/package.opm`

列表

列出所有的 OPM 软件包 。

- Web: http://localhost/otrs/index.pl?Action=AdminPackageManager
- CMD: `bin/otrs.Console.pl Admin::Package::List`

4.2 创建软件包

如果你想创建一个 OPM 包(`.opm`)，你需要创建一个规范文件(`.sopm`)，包含包的属性 。

4.2.1 包规范文件

OPM 包是基于 XML 的 。您可以通过文本或 XML 编辑器创建/编辑该 `.sopm` 文件 。它包含元数据 、文件列表和数据库选项 。

<Name> * 包名称 。

```
<Name>Calendar</Name>
```

<Version> * 包版本 。

```
<Version>1.2.3</Version>
```

<Framework> * 目标框架版本(7.0.x 表示 7.0.1 或 7.0.2 之类)。

```
<Framework>7.0.x</Framework>
```

也可以多次使用 。

```
<Framework>5.0.x</Framework>
<Framework>6.0.x</Framework>
<Framework>7.0.x</Framework>
```

<Vendor> * 包的提供商 。

```
<Vendor>OTRS AG</Vendor>
```

<URL> * 提供商的 URL 。

```
<URL>https://otrs.com/</URL>
```

<License> * 包的许可证 。

```
<License>GNU GENERAL PUBLIC LICENSE Version 3, 29 June 2007</License>
```

<ChangeLog> 包更改日志 。

```
<ChangeLog Version="1.1.2" Date="2018-11-15 18:45:21">Added some feature.
↪</ChangeLog>
<ChangeLog Version="1.1.1" Date="2018-11-15 16:17:51">New package.</
↪ChangeLog>
```

<Description> * 不同语言的包描述 。

```
<Description Lang="en">A web calendar.</Description>
<Description Lang="de">Ein Web Kalender.</Description>
```

包的操作 安装后包的可能操作 。如果未在包中定义其中一个操作，则将被视为可能 。

```
<PackageIsVisible>1</PackageIsVisible>
<PackageIsDownloadable>0</PackageIsDownloadable>
<PackageIsRemovable>1</PackageIsRemovable>
```

一个特殊的包操作是 PackageAllowDirectUpdate 。只有在该包上定义并设置为 true 时，才能将该包从较低的主要版本(早于最后一个版本)升级到最新版本 。(例如，将 OTRS 5 的包更新到 OTRS 7)。

```
<PackageAllowDirectUpdate>1</PackageAllowDirectUpdate>
```

<BuildHost> 这将由 OPM 自动填写 。

```
<BuildHost>?</BuildHost>
```

<BuildDate> 这将由 OPM 自动填写 。

```
<BuildDate>?</BuildDate>
```

<PackageRequired> 必须事先安装的软件包 。如果使用 PackageRequired，则必须指定所需包的版本 。

```
<PackageRequired Version="1.0.3">SomeOtherPackage</PackageRequired>
<PackageRequired Version="5.3.2">SomeotherPackage2</PackageRequired>
```

<ModuleRequired> 必须事先安装的 Perl 模块 。

```
<ModuleRequired Version="1.03">Encode</ModuleRequired>
<ModuleRequired Version="5.32">MIME::Tools</ModuleRequired>
```

<OS> 所需操作系统 。

```
<OS>linux</OS>
<OS>darwin</OS>
<OS>mswin32</OS>
```

<Filelist> 这是包中包含的文件列表 。

```
<Filelist>
    <File Permission="644" Location="Kernel/Config/Files/Calendar.pm"/>
    <File Permission="644" Location="Kernel/System/CalendarEvent.pm"/>
    <File Permission="644" Location="Kernel/Modules/AgentCalendar.pm"/>
    <File Permission="644" Location="Kernel/Language/de_AgentCalendar.pm"/
 ↪>
</Filelist>
```

<DatabaseInstall> 安装软件包时必须创建的数据库条目 。

```
<DatabaseInstall>
    <TableCreate Name="calendar_event">
    <Column Name="id" Required="true" PrimaryKey="true" AutoIncrement=
 ↪"true" Type="BIGINT"/>
    <Column Name="title" Required="true" Size="250" Type="VARCHAR"/>
    <Column Name="content" Required="false" Size="250" Type="VARCHAR"/>
    <Column Name="start_time" Required="true" Type="DATE"/>
    <Column Name="end_time" Required="true" Type="DATE"/>
    <Column Name="owner_id" Required="true" Type="INTEGER"/>
    <Column Name="event_status" Required="true" Size="50" Type="VARCHAR"/>
    </TableCreate>
</DatabaseInstall>
```

你还可以选择 `<DatabaseInstall Type="post">` 或 `<DatabaseInstall Type="pre">` 来分别定义执行时间(`post` 是默认的)。有关更多信息，请参阅软件包生命周期 。

<DatabaseUpgrade> 有关升级软件包时必须执行哪些操作的信息 。

例如，如果已安装的软件包版本低于 1.3.4(例如 1.2.6)，则将执行定义的操作：

```
<DatabaseUpgrade>
    <TableCreate Name="calendar_event_involved" Version="1.3.4">
        <Column Name="event_id" Required="true" Type="BIGINT"/>
        <Column Name="user_id" Required="true" Type="INTEGER"/>
    </TableCreate>
</DatabaseUpgrade>
```

你还可以选择 `<DatabaseUpgrade Type="post">` 或 `<DatabaseUpgrade Type="pre">` 来分别定义执行时间(`post` 是默认的)。有关更多信息，请参阅软件包生命周期 。

<DatabaseReinstall> 有关安装软件包时必须执行哪些操作的信息 。

```
<DatabaseReinstall></DatabaseReinstall>
```

你还可以选择 `<DatabaseReinstall Type="post">` 或 `<DatabaseReinstall Type="pre">` 来分别定义执行时间(`post` 是默认的)。有关更多信息，请参阅软件包生命周期 。

<DatabaseUninstall> 卸载软件包时要执行的操作 。

```
<DatabaseUninstall>
    <TableDrop Name="calendar_event" />
</DatabaseUninstall>
```

你还可以选择 <DatabaseUninstall Type="post"> 或 <DatabaseUninstall Type="pre"> 来分别定义执行时间(post 是默认的)。有关更多信息，请参阅软件包生命周期 。

<IntroInstall> 在安装对话框中显示安装前或安装后的介绍 。

```
<IntroInstall Type="post" Lang="en" Title="Some Title"><![CDATA[
    Some information formatted in HTML.
]]></IntroInstall>
```

您还可以使用 Format 属性来定义在显示介绍时是否要使用 html (默认值)或 plain 来自动使用 <pre> </pre> 标签(保留内容的换行符和空格)。

<IntroUninstall> 在卸载对话框中显示卸载前或卸载后的介绍 。

```
<IntroUninstall Type="post" Lang="en" Title="Some Title"><![CDATA[
    Some information formatted in HTML.
]]></IntroUninstall>
```

您还可以使用 Format 属性来定义在显示介绍时是否要使用 html (默认值)或 plain 来自动使用 <pre> </pre> 标签(保留内容的换行符和空格)。

<IntroReinstall> 在重新安装对话框中显示重新安装前或重新安装后的介绍 。

```
<IntroReinstall Type="post" Lang="en" Title="Some Title"><![CDATA[
    Some information formatted in HTML.
]]></IntroReinstall>
```

您还可以使用 Format 属性来定义在显示介绍时是否要使用 html (默认值)或 plain 来自动使用 <pre> </pre> 标签(保留内容的换行符和空格)。

<IntroUpgrade> 在升级对话框中显示升级前或升级后的介绍 。

```
<IntroUpgrade Type="post" Lang="en" Title="Some Title"><![CDATA[
    Some information formatted in HTML.
]]></IntroUpgrade>
```

您还可以使用 Format 属性来定义在显示介绍时是否要使用 html (默认值)或 plain 来自动使用 <pre> </pre> 标签(保留内容的换行符和空格)。

<CodeInstall> 安装包后要执行的 Perl 代码 。

```
<CodeInstall><![CDATA[
# log example
$Kernel::OM->Get('Kernel::System::Log')->Log(
    Priority => 'notice',
    Message => "Some Message!",
);
# database example
$Kernel::OM->Get('Kernel::System::DB')->Do(SQL => "SOME SQL");
]]></CodeInstall>
```

你还可以选择 <CodeInstall Type="post"> 或 <CodeInstall Type="pre"> 来分别定义执行时间(post 是默认的)。有关更多信息，请参阅软件包生命周期 。

<CodeUninstall> 卸载软件包后要执行的 Perl 代码 。在卸载软件包之前或之后的时间 。

```
<CodeUninstall><![CDATA[
# Some Perl code.
]]></CodeUninstall>
```

你还可以选择 <CodeUninstall Type="post"> 或 <CodeUninstall Type="pre"> 来分别定义执行时间(post 是默认的)。有关更多信息，请参阅软件包生命周期 。

<CodeReinstall> 重新安装软件包后要执行的 Perl 代码 。

```
<CodeReinstall><![CDATA[
# Some Perl code.
]]></CodeReinstall>
```

你还可以选择 <CodeReinstall Type="post"> 或 <CodeReinstall Type="pre"> 来分别定义执行时间(post 是默认的)。有关更多信息，请参阅软件包生命周期 。

<CodeUpgrade> 升级软件包后要执行的 Perl 代码(以 version 标签为准)。

例如，如果已安装的软件包版本低于 1.3.4(例如 1.2.6)，则将执行定义的操作：

```
<CodeUpgrade Version="1.3.4"><![CDATA[
# Some Perl code.
]]></CodeUpgrade>
```

你还可以选择 <CodeUpgrade Type="post"> 或 <CodeUpgrade Type="pre"> 来分别定义执行时间(post 是默认的)。有关更多信息，请参阅软件包生命周期 。

<PackageMerge> 此标记表示包已合并到另一个包中 。在这种情况下，需要从文件系统和包数据库中删除原始包，但必须保留所有数据 。

我们假设 PackageOne 被合并到 PackageTwo 中 。那么 PackageTwo.sopm 应该包含这个：

```
<PackageMerge Name="MergeOne" TargetVersion="2.0.0"></PackageMerge>
```

如果 PackageOne 也包含数据库结构，我们需要确保它是在包的最新可用版本，以便在合并包后在数据库中具有一致状态 。TargetVersion 属性的作用：它表示在创建 PackageTwo 时 PackageOne 的最后一个已知版本 。这主要是为了在用户的系统中发现了一个 PackageOne 版本比 TargetVersion 中指定的版本 更新时停止升级过程，因为这可能会导致问题 。

另外，可以为 PackageOne 添加所需的数据库和代码升级标签，以确保在 合并之前将其正确升级到 TargetVersion - 以避免不一致问题 。这看起来是这样的：

```
<PackageMerge Name="MergeOne" TargetVersion="2.0.0">
    <DatabaseUpgrade Type="merge">
        <TableCreate Name="merge_package">
            <Column Name="id" Required="true" PrimaryKey="true"␣
↪AutoIncrement="true" Type="INTEGER"/>
            <Column Name="description" Required="true" Size="200" Type=
↪"VARCHAR"/>
        </TableCreate>
    </DatabaseUpgrade>
</PackageMerge>
```

正如您所看到的，在这种情况下需要设置属性 Type="merge" 。只有在可以进行包合并时才会执行这些部分 。

软件包条件 IfPackage 和 IfNotPackage 属性可以添加到常规 Database* 和 Code* 部分。如果它们存在，则只有在本地软件包存储库中存在或不存在另一个软件包时才会执行该部分。

```
<DatabaseInstall IfPackage="AnyPackage">
    # ...
</DatabaseInstall>
```

or(或)

```
<CodeUpgrade IfNotPackage="OtherPackage">
    # ...
</CodeUpgrade>
```

这些属性也可以在 PackageMerge 标签内的 Database* 和 Code* 部分设置。

4.2.2 .sopm 示例

这是一个示例规范文件，包含一些上述标签。

```
<?xml version="1.0" encoding="utf-8" ?>
<otrs_package version="1.0">
    <Name>Calendar</Name>
    <Version>0.0.1</Version>
    <Framework>7.0.x</Framework>
    <Vendor>OTRS AG</Vendor>
    <URL>https://otrs.com/</URL>
    <License>GNU GENERAL PUBLIC LICENSE Version 3, 29 June 2007</License>
    <ChangeLog Version="1.1.2" Date="2018-11-15 18:45:21">Added some feature.
↪</ChangeLog>
    <ChangeLog Version="1.1.1" Date="2018-11-15 16:17:51">New package.</
↪ChangeLog>
    <Description Lang="en">A web calendar.</Description>
    <Description Lang="de">Ein Web Kalender.</Description>
    <IntroInstall Type="post" Lang="en" Title="Thank you!">Thank you for␣
↪choosing the Calendar module.</IntroInstall>
    <IntroInstall Type="post" Lang="de" Title="Vielen Dank!">Vielen Dank fuer␣
↪die Auswahl des Kalender Modules.</IntroInstall>
    <BuildDate>?</BuildDate>
    <BuildHost>?</BuildHost>
    <Filelist>
        <File Permission="644" Location="Kernel/Config/Files/Calendar.pm"></
↪File>
        <File Permission="644" Location="Kernel/System/CalendarEvent.pm"></
↪File>
        <File Permission="644" Location="Kernel/Modules/AgentCalendar.pm"></
↪File>
        <File Permission="644" Location="Kernel/Language/de_AgentCalendar.pm">
↪</File>
        <File Permission="644" Location="Kernel/Output/HTML/Standard/
↪AgentCalendar.tt"></File>
        <File Permission="644" Location="Kernel/Output/HTML/
↪NotificationCalendar.pm"></File>
        <File Permission="644" Location="var/httpd/htdocs/images/Standard/
↪calendar.png"></File>
```

(下页继续)

(续上页)

```
    </Filelist>
    <DatabaseInstall>
        <TableCreate Name="calendar_event">
            <Column Name="id" Required="true" PrimaryKey="true" AutoIncrement=
↪"true" Type="BIGINT"/>
            <Column Name="title" Required="true" Size="250" Type="VARCHAR"/>
            <Column Name="content" Required="false" Size="250" Type="VARCHAR"/
↪>
            <Column Name="start_time" Required="true" Type="DATE"/>
            <Column Name="end_time" Required="true" Type="DATE"/>
            <Column Name="owner_id" Required="true" Type="INTEGER"/>
            <Column Name="event_status" Required="true" Size="50" Type=
↪"VARCHAR"/>
        </TableCreate>
    </DatabaseInstall>
    <DatabaseUninstall>
        <TableDrop Name="calendar_event"/>
    </DatabaseUninstall>
</otrs_package>
```

4.2.3 构建软件包

若要从 opm 规范文件构建.opm 包 。

```
shell> bin/otrs.Console.pl Dev::Package::Build /path/to/example.sopm /tmp
Building package...
Done.
shell>
```

4.2.4 软件包生命周期

下图显示了软件包安装 、升级和卸载的生命周期如何在后端逐步进行 。

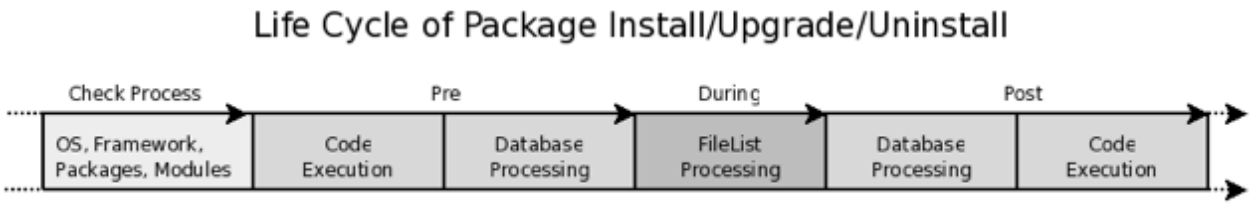

图 1: 软件包生命周期

4.3 软件包移植

对于每个新的次要或主要版本的 OTRS，您需要移植软件包并确保它们仍然可以使用 OTRS API 。

本节列出了将软件包从 OTRS 6 移植到 7 时需要检查的更改 。

4.3.1 会话始终需要 Cookie

从 OTRS 7 开始，会话总是要求启用 cookie 。因此，`SessionIDCookie` 值已从 `LayoutObject` 、模板机制 和 JavaScript 中删除 。不再需要将会话变量附加到 URL 或 HTTP 请求负载 。

4.3.2 `groups` 表被重命名

由于 MySQL 8 的变化，`groups` 表必须重命名为 `groups_table` 。如果直接在任何 SQL 语句中使用此表，则需要对其进行调整 。有关更多信息，请参阅 bug#13866 。

4.3.3 新的外部人员界面

现有的客户(`customer.pl`)和公共(`public.pl`)界面被新的 REST 后端(`Kernel/WebApp`)和现代的基于 Vue.js 的前端应用程序所取代 。这意味着必须移植和/或重写所有相关代码 。

对于不提供 HTML 应用程序的公共前端模块，有一种特殊情况 。这些可以很容易地移植到新的 REST 后端(另请参阅 REST API 文档)。例如，参见 `Kernel/WebApp/Controller/API/Public/Package/Repository.pm` 。此示例还显示了端点如何同时支持新的 REST 类 URL 和基于 `/otrs/customer.pl?Action=MyAction` 路由的旧 URL 。

对于从旧系统链接的客户界面中的一些重要 URL，可能需要提供重定向控制器以确保旧 URL 继续工作 。

4.3.4 流程管理中的更改

迁移脚本 `scripts/DBUpdate-to-7.pl` 将升级数据库中已有的所有流程 。只有使用 OTRS 7 提供的任何新功能才需要手动操作 。

新的活动类型

由于 OTRS 7 能够管理更多活动类型，因此所有现有活动现在都成为 * 用户任务 * 活动 。要更新 YAML 文件上的任务定义，请添加 `Type: UserTask`，其缩写与 `Name` 或 `ID` 相同 。

重命名流程管理组件

`Transition`(转换) 变为 **`SequenceFlow`**(序列流) 此流程组件已重命名为与 BPMN 更加一致 。相应地重命名了文件 、类和方法 。需要根据新约定更新自定义文件 。

`TransitionAction`(转换操作) 变为 **`SequenceFlowAction`**(序列流操作) 此流程组件在 BPMN 中不存在，但也必须重命名为与其他更改一致 。相应地重命名了文件 、类和方法 。需要根据新约定更新自定义文件 。

`TransitionActionModules`(转换操作模块) 变为 **`ProcessManagementModules`**(流程管理模块) 这些流程组件不仅用于 序列流操作，还用于 脚本任务活动，并且已经从 `Kernel/System/ProcessManagement/TransitionAction` 转移到 `Kernel/System/ProcessManagement/Modules` 。

新的流程管理模块可以提供更多的字段类型和选项，以向流程设计者提供参数 。请按照流程管理 文档中的说明进行操作，以了解有关此新功能以及如何改进现有模块的更多信息 。

需要更新任何已发布的流程定义以使用新的名称方案 。

- 用 `SequenceFlow` 替换 `Transition` 。

- 用 SequenceFlows 替换 Transitions 。
- 用 SequenceFlowAction 替换 TransitionAction 。
- 用 SequenceFlowActions 替换 TransitionActions 。
- 从所有 SequenceFlowAction 上的 Module: 中删除 Kernel::System::ProcessManagement::TransitionAction 。例如：Module: Kernel::System::ProcessManagement::TransitionAction::TicketLockSet 应该变成 Module: TicketLockSet 。

4.3.5 LayoutObject 中的变化

Kernel/Output/HTML/Layout.pm 中有一些变化，这些变化是使用 Mojolicious 实时 Web 框架正确渲染内容所必需的 。

屏幕中未显示表格或空白的表格

请确保检查每个产生表格式输出的屏幕（ 例如 Kernel/Modules/AgentTicketStatusView.pm ）。如果列表（ 如工单列表 ）是空的甚至根本没有显示，检查在创建页面的输出时是否使用了参数 Output => 1 。

传统前端模块中的编码问题

如果你遇到像（ 日耳曼语系中的 ）元音变音这样的破碎字符的问题，可能是因为要显示的内容是由 Print() 方法渲染的 。要解决此问题，请将代码从使用 Print() 方法切换到从前端模块返回完整响应的常规方法 。

CHAPTER 5

文档

从 OTRS 7 开始，文档以 *reStructuredText* 格式提供，取代了旧的 *DocBook* 格式 。在 OTRS 文档页面 上可以获得各种输出格式，如 HTML 、EPUB 和 PDF 。

文档是用英语编写的，并翻译成多种语言 。

5.1 文档基础架构

OTRS 使用 Sphinx 生成输出 。HTML 输出是使用 Read the Docs 主题生成的 。

注解: 所有输出都在构建服务器上自定义 。如果要在本地计算机中设置开发人员环境以进行测试和编写文档，则输出可能会有所不同 。

5.2 reStructuredText 入门

文档格式名称是 **reStructuredText**（ 一个单词，这是正确的拼写 ）。这是一种使用纯文本和小型内联标记的易于阅读的文档格式 。

这个简短的教程将指导您创建或更新文档 。提供有关如何使用 *reStructuredText* 格式的全功能教程超出了本文档的范围，可在互联网上找到许多教程（ 例如 Sphinx reStructuredText primer 和 reStructuredText 用户文档 ）和 在线编辑器 。

以下示例显示了最常用的文档元素 。

5.2.1 标题

要在文档中使用标题，必须在标题下加上特殊字符 。下划线必须从标题的第一个字母开始，并以标题的最后一个字母结束 。特殊字符的层次结构如下：= 、- 、~ 、^ 、. 。

以下示例显示了标题的用法：

```
Chapter title
=============

This is the heading 1 title. It has numbering like 1.

Section title
-------------

This is the heading 2 title. It has numbering like 1.1.

Subsection title
~~~~~~~~~~~~~~~~

This is the heading 3 title. It has numbering like 1.1.1.

Subsubsection title
^^^^^^^^^^^^^^^^^^^

This is the heading 4 title. It has numbering like 1.1.1.1.

Subsubsubsection title
......................

This is the heading 5 title. It has numbering like 1.1.1.1.1. Please don't␣
↪use this level of heading.
```

5.2.2 段落

要写段落，您必须从行首开始句子 。要创建新段落，只需在段落之间留一行空白 。示例：

```
Lorem ipsum dolor sit amet, consectetur adipiscing elit. Sed dictum imperdiet␣
↪enim. Curabitur
nisi diam, lobortis facilisis quam ut, porttitor consequat lectus. Nam␣
↪elementum, ipsum id
feugiat vestibulum, dolor ante dictum quam, ac bibendum ipsum felis in orci.

Vestibulum maximus egestas orci, eget consequat nibh imperdiet eget.␣
↪Suspendisse sagittis tempus
sapien, sit amet tincidunt tortor efficitur et. Etiam ac lacus sem. Sed ut␣
↪magna imperdiet,
viverra quam vitae, consequat mauris.
```

5.2.3 内联标记

标准的内联标记非常简单：

- 一个星号：*text* 表示 重点（ 斜体 ）。
- 两个星号：**text** 表示 强调（ 粗体 ）。
- 反引号：“text“ 表示 常量文本（ 示例代码 ）。

如果正在运行的文本中出现星号或反引号，可能与内联标记分隔符混淆，则必须使用反斜杠进行转义，例如* 。

5.2.4 列表项目

要创建无序列表，请使用星号(*)或短划线(-)开始一行 。要创建有序列表，请使用数字或井号(#)开始一行 。如果需要嵌套列表，请在列表项之间留一个空行，并使用 3 个空格的缩进 。例：

```
* This is a bulleted list.
* It has two items, the second
  item uses two lines.

1. This is a numbered list.
2. It has two items too.

#. This is a numbered list.
#. It has two items too.
```

嵌套列表的例子：

```
- this is
- a list

  - with a nested list
  - and some subitems

- and here the parent list continues
```

5.2.5 文字块

文字块是应逐字显示的文本 。要创建文字块，请执行以下操作：

1. 在新行中输入 2 冒号(::) 。
2. 留一个空行 。
3. 用 3 个空格的缩进写出文本 。

对代码片段 、终端输出 、配置文件等使用文本块 。示例：

```
::

   $Self->{DatabaseHost} = '127.0.0.1';
   $Self->{Database} = 'otrs';
   $Self->{DatabaseUser} = 'otrs';
```

如果代码片段的语言是已知的，则可以将其指定为语法突出显示：

```
.. code-block:: perl

   $Self->{DatabaseHost} = '127.0.0.1';
   $Self->{Database} = 'otrs';
   $Self->{DatabaseUser} = 'otrs';
```

```
.. code-block:: xml

   <Setting Name="FAQ::Agent::StateTypes" Required="1" Valid="1">
       <Description Translatable="1">List of state types which can be used in␣
↪the agent interface.</Description>
       <Navigation>Core::FAQ</Navigation>
       <Value>
           <Array>
               <Item>internal</Item>
               <Item>external</Item>
               <Item>public</Item>
           </Array>
       </Value>
   </Setting>
```

5.2.6 表格

要创建表格，必须绘制表格 。示例：

```
+------------------------+------------+----------+----------+
| Header row, column 1   | Header 2   | Header 3 | Header 4 |
| (header rows optional) |            |          |          |
+========================+============+==========+==========+
| body row 1, column 1   | column 2   | column 3 | column 4 |
+------------------------+------------+----------+----------+
| body row 2             | ...        | ...      |          |
+------------------------+------------+----------+----------+
```

5.2.7 超链接

超链接可以用于内联或引用 。对于内联使用，请使用反引号和两个尾随下划线字符封装链接的文本和 URL 。

```
Visit `OTRS website <https://otrs.com>`__ for more information.
```

上面的链接将显示为：OTRS 网站 。

要创建引用的链接，您必须分隔文本和链接 。例：

```
The documentations are available in the `OTRS documentation portal`_.

.. _OTRS documentation portal: https://doc.otrs.com/
```

5.2.8 图片

要将图片插入文档：

1. 将图片放在 images 文件夹中 。
2. 创建对图片的引用：

```
.. figure:: images/admin-general-catalog-management-class.png
   :alt: Admin General Catalog

   Admin General Catalog
```

5.2.9 彩色的框

这些框具有特殊含义，默认情况下会突出显示 。

警告框：

```
.. warning::

   This is a warning box.
```

警告: 这是一个警告框 。

备注框：

```
.. note::

   This is a note box.
```

注解: 这是一个备注框 。

参见框：

```
.. seealso::

   This is a see also box.
```

参见:

这是一个参见框 。

5.3 样式指南

这部分文档仅用于视觉风格和措辞 。

5.3.1 编写内容

有一个互联网俚语 **TL; DR**，这意味着 太长，不看（ 更多信息请参阅 Wikipedia 维基百科 ）。很多人不喜欢阅读长篇文章，所以请尽量缩短文档 。使用分步教程而不是编写文本墙 。

例如，这是一个 错误的编写内容示例：

```
The agents are able to change the interface language of OTRS. To change the
interface language, click on your avatar on the top left corner, then select
Personal Settings menu item. A new screen will be displayed. On this screen
click on the User Profile, and then find a widget named Language. Select the
desired language in the drop-down menu. Please make sure to click on the
Save button next to the language widget.
```

同样的内容采用 建议的 、人类可理解的格式：

```
To change the interface language of OTRS:

1. Click on your avatar on the top left corner.
2. Select *Personal Settings*.
3. Click the *User Profile* in the new screen.
4. Choose a language from the drop-down menu of the *Language* widget.
5. Click the *Save* button next to the widget.
```

后者更容易翻译，因为语言文件中将包含 6 个短句 。如果在其中一个句子中更改了内容，则只需要再次检查和翻译已更改的句子 。第一个错误示例只将一个大字符串放入语言文件中，如果在源字符串中进行某些更改，则翻译程序需要检查并重新翻译整个字符串 。

5.3.2 屏幕截图

不要使用电脑的原始分辨率 。通常它是全高清或更高，因此在某些输出中创建具有此分辨率的屏幕截图将变得不可读，因为在 PDF 的情况下所有图像必须缩小到 A4 纸的宽度 。OTRS 使用响应式设计，因此 1025 像素是最小的，OTRS 假设它是一个大的显示器 。请将此值用作屏幕截图的宽度 。

参见:

可以在所有 Web 浏览器中使用称为 移动模式或 响应式设计的功能设置分辨率 。查看浏览器用户手册以了解该功能的用法，并将屏幕宽度设置为 1025 像素 。

这是一个 错误的截图示例，因为它具有全高清分辨率 。由于自动收缩，屏幕截图上的文字难以阅读：

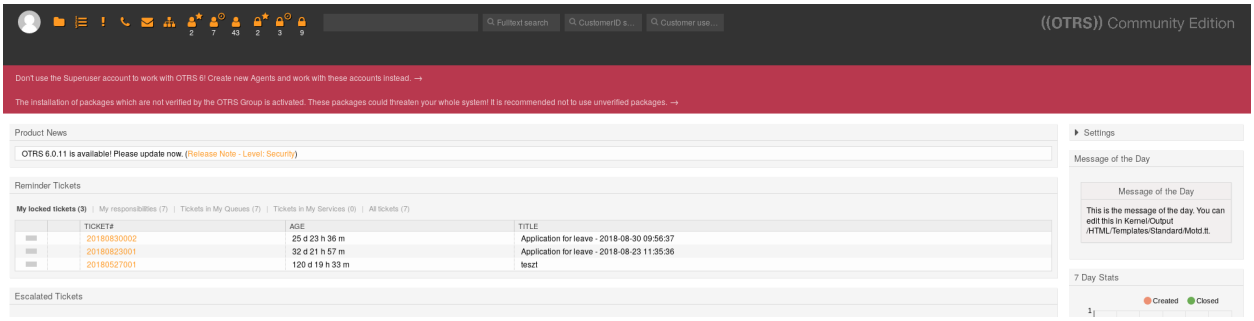

图 1: 服务人员仪表板（ 1920 像素宽 ）

与 建议的分辨率相同的屏幕截图 。文本更容易阅读：

如果屏幕截图具有良好的像素分辨率，但具有高 DPI，则也是错误的 。例如，这个截图是 错误的，因为它上面的文本比文档中的其它文本大得多：

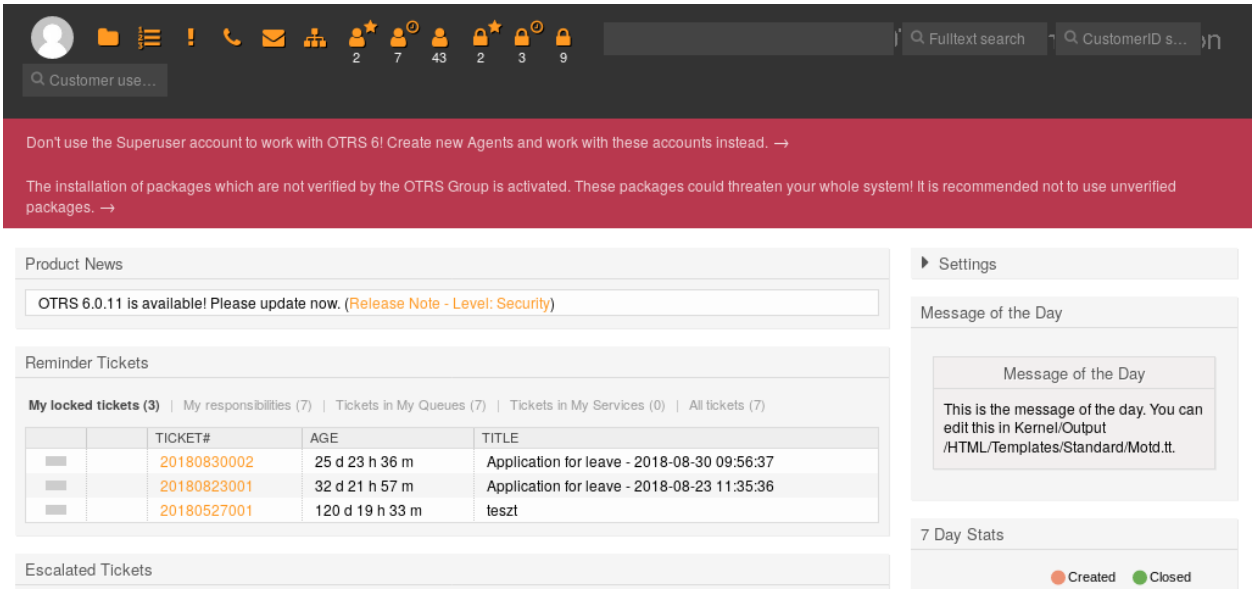

图 2: 服务人员仪表板（ 1025 像素宽 ）

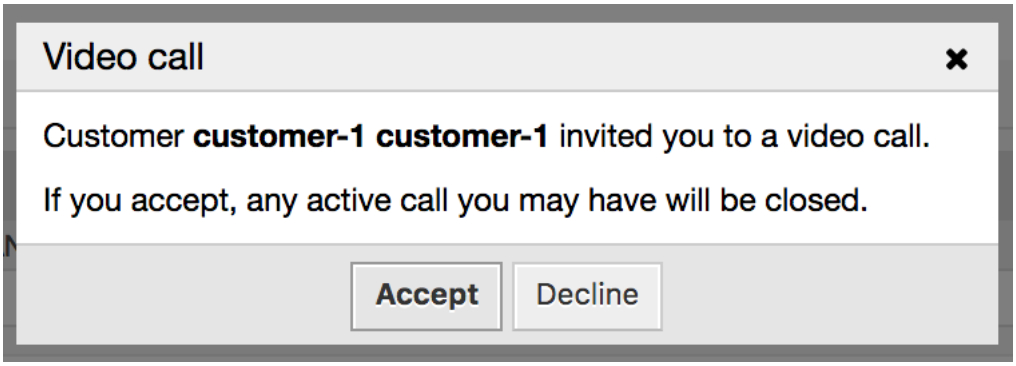

图 3: 视频邀请对话框（ 宽度为 756 像素但具有高 DPI ）

使用 Firefox 创建屏幕截图

如果只需要部分屏幕截图，则需要裁剪屏幕截图 。OTRS 的管理员界面由左侧边栏和主内容列组成 。要使用 Firefox 创建屏幕截图：

1. 右键单击浏览器中的元素，然后选择 查看元素 。
2. 如果未选中，则选择 DOM 中的元素 。
3. 右键单击节点并选择 节点截图 。

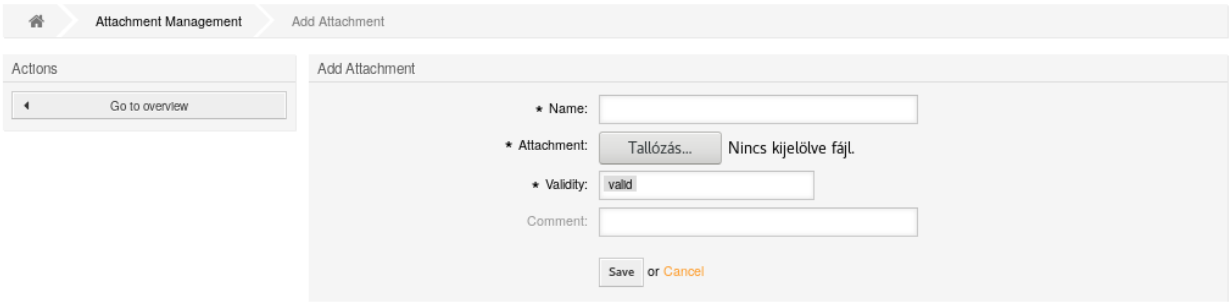

图 4: 主要内容的示例屏幕截图

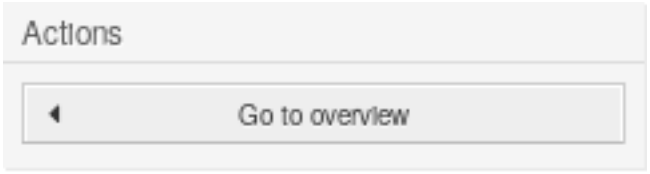

图 5: 左侧边栏的示例屏幕截图

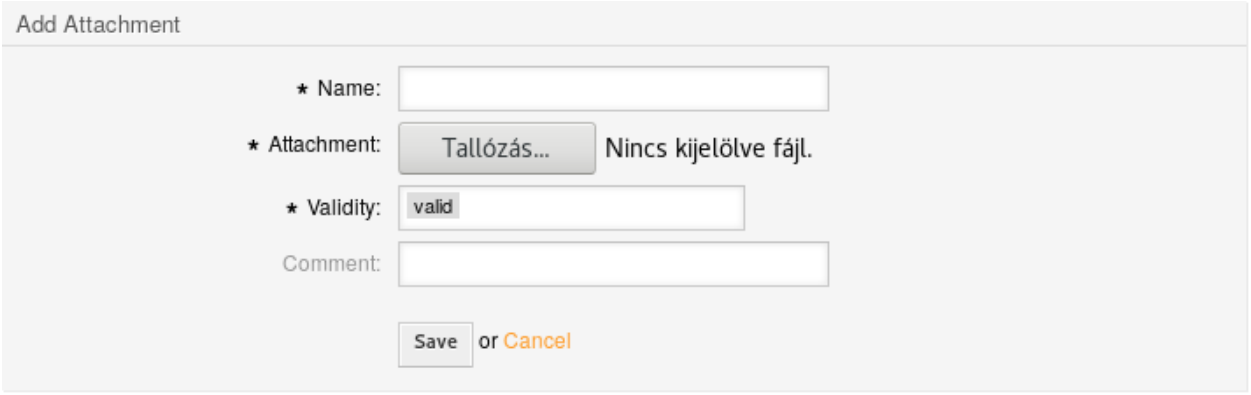

图 6: 主内容列的示例屏幕截图

5.3.3 文档中的大写

对于标题总是必须使用句子大小写，这意味着，标题总是大写：

- 名词(man, bus, book)。
- 形容词(angry, lovely, small)。
- 动词(run, eat, sleep)。
- 副词(slowly, quickly, quietly)。
- 代词(he, she, it)。
- 从属连词(as, because, that)。

在标题中不大写：

- 冠词：a, an, the 。
- 并列连词：and 、but 、or 、for 、nor 等 。
- 介词(少于五个字母)：on 、at 、to 、from 、by 等 。

在普通句子中，不要将任何单词大写，只有名称和标题的引用必须大写 。这是 错误的示例：

```
An Agent is a user, who handles Tickets in the Ticket Zoom screen.
```

建议的句子有适当的大写字母 。此外，*Ticket Zoom* 是屏幕的名称，因此需要强调：

```
An agent is a user, who handles tickets in the *Ticket Zoom* screen.
```

5.3.4 按钮和屏幕名称

在内容句子中，应强调所有按钮和屏幕，并应使用大写字母或句子大小写 。不要使用撇号或引号来强调 。

这句话是 错误的，因为用撇号来强调：

```
If you click the 'Save and Finish' button, you will be redirected to the
↪'Ticket Zoom' screen.
```

建议的方法是用星号来强调：

```
If you click the *Save and Finish* button, you will be redirected to the␣
↪*Ticket Zoom* screen.
```

5.3.5 措词

不要在句子中使用变量名 。这句话是 错误的，因为变量名对某些人来说毫无意义：

```
Add a new widget to AgentTicketZoom.
```

没有变量名的同一个句子，这是 建议的：

```
Add a new widget to the *Ticket Zoom* screen of the agent interface.
```

5.3.6 变量名

变量名应始终标记为 `literal( 原封不动显示文字 )` 内容 。这对翻译人员非常有用，因为他们可以确切地知道，字符串不得翻译 。如果字符串未标记为文字内容，则通常应将其翻译 。例如：

```
The ``ObjectManager`` object has an ``Init()`` function. Additional␣
↪configuration can be set in ``Kernel::Config::Config.pm`` file.
```

5.4 翻译文档

OTRS 图形界面、公共扩展模块和文档的所有翻译都通过 Weblate 进行管理。

从 OTRS 7 开始，文档以 *reStructuredText* 格式提供，取代了旧的 *DocBook* 格式。翻译文档时请注意不要破坏结构。

这里有些例子。

重点 强调文本在两个星号之间。文本应该翻译。这通常用于屏幕名称、标题、按钮和标签。请检查用户界面翻译以查找和使用文档中的相同措辞。

示例原句：

```
Use the *Ticket Zoom* screen to see the ticket details.
```

匈牙利语翻译示例：

```
Használja a *Jegynagyítás* képernyőt a jegy részleteinek megtekintéséhez.
```

强调 强文本位于两个双星号之间。应翻译该文本。这通常用于重要信息。

示例原句：

```
**Don't continue** the update if you get an error message.
```

匈牙利语翻译示例：

```
**Ne folytassa** a frissítést, ha hibaüzenetet kap.
```

常量文本 常量文本介于两个双反引号之间。这通常用于变量名、配置名和文件路径，不能翻译，否则会破坏结构。

示例原句：

```
Activate ``group`` in system configuration ``ExamplePermission###100``.
```

匈牙利语翻译示例：

```
Aktiválja a ``group`` értéket az ``ExamplePermission###100``␣
↪rendszerbeállításban.
```

内部链接 内部链接指向页面的其它页面或标题。:doc:`page-name` 用于引用页面，:ref:`Heading Title` 用于引用标题。有一个自定义标签 :sysconfig:`System Configuration Name` 用于引用系统配置。不得翻译 页面名称、标题和 系统配置名称文本，否则将破坏结构。

示例原句：

```
See page :doc:`queues` to add a queue, especially section :ref:`Queue␣
↪Settings`.
```

匈牙利语翻译示例：

```
Nézze meg a :doc:`queues` oldalt, különösen a :ref:`Queue Settings`␣
↪szakaszt.
```

外部链接 外部链接由可见文本和表单中的 URL 按 *‘visible text <https://example.com>‘__* 格式组成。应翻译 *visible text* 。

示例原句：

```
See `OTRS website <https://otrs.com/>`__ for more information.
```

匈牙利语翻译示例：

```
Nézze meg az `OTRS weboldalát <https://otrs.com/>`__ a további␣
↪információkért.
```

CHAPTER 6

为 OTRS 做出贡献

本章将介绍如何为 OTRS 框架做出贡献，以便其它用户能够从您的工作中获益 。

6.1 发送贡献

可以在 GitHub 上找到 ((OTRS)) 社区版和其它公共模块的源代码 。从那里，您可以访问所有可用存储库的列表 。它还描述了当前活动的分支以及贡献应该去哪里（ 稳定分支与开发分支 ）。

强烈建议您在发送贡献之前使用有用的工具 中所述的 OTRS 代码质量检查程序 OTRSCodePolicy 。如果您的代码没有通过这个工具进行验证，那么它很可能不会被接受 。

将您的贡献发送给 OTRS 开发人员团队的最简单方法是在 Github 中创建一个 *pull request* 。请查看关于 GitHub，特别是关于 forking a repository and sending pull requests 的说明 。

基础的工作流应看起来像这样：

1. 如果您还没有帐户，请在 GitHub 注册 。
2. 克隆您要贡献的仓库（ Fork ），并检出要进行更改的分支 。
3. 根据当前分支为您的修订/特色功能/贡献创建一个新的开发分支 。
4. 完成更改并提交更改后，将分支推送到 GitHub 。
5. 创建一个”pull request”，OTRS 开发团队将会收到通知，并检查您的”pull request”，要么合并更改，要么给您一些关于可能的改进的反馈 。

这可能听起来很复杂，但一旦你设置了这个工作流程，你会看到做出贡献非常简单 。

6.2 翻译

翻译主要由 OTRS 用户提供和维护，因此需要 您的帮助 。

OTRS 图形界面 、公共扩展模块和文档的所有翻译都通过 Weblate 进行管理 。

为翻译做出贡献：

1. 在 Weblate 注册一个免费的翻译帐户 。
2. 选择翻译组件和要翻译的语言 。
3. 开始更新您的翻译 。无需其它软件或文件 。

注解：如果仪表板中未列出您的语言，则可以请求一门语言 。经批准后，您就可以开始翻译 。

从 OTRS 7 开始，文档以 *reStructuredText* 格式提供，取代了旧的 *DocBook* 格式 。翻译文档时请注意不要破坏结构 。

参见：

您可以在文档 一章中找到一些示例 。

OTRS 开发人员会不时将翻译下载到 OTRS 源代码库中，您无需在任何地方提交 。

6.3 代码样式指南

为了保持 OTRS 项目的一致发展，我们为不同的编程语言制定了样式指南 。

6.3.1 Perl

空格

TAB：我们用 4 个空格 。大括号示例：

```
if ($Condition) {
    Foo();
}
else {
    Bar();
}

while ($Condition == 1) {
    Foo();
}
```

每行宽度

除非出于特殊原因，否则每行通常不应超过 120 个字符 。

空格和圆括号

为了获得更多可读性，我们在关键字和左括号之间使用空格 。

```
if ()...
for ()...
```

如果只有一个变量，则括号内不用空格包含变量 。

```
if ($Condition) { ... }

# instead of

if ( $Condition ) { ... }
```

如果条件不只是一个变量，我们在括号和条件之间使用空格 。关键字(例如 `if`)和左括号之间仍然存在空格 。

```
if ( $Condition && $ABC ) { ... }
```

请注意，对于 Perl 内置函数，我们不使用括号：

```
chomp $Variable;
```

源码文件头和字符集

将以下头部信息附加到每个源文件 。源文件以 UTF-8 字符集保存 。

```
# --
# Copyright (C) 2001-2020 OTRS AG, https://otrs.com/
# --
# This software comes with ABSOLUTELY NO WARRANTY. For details, see
# the enclosed file COPYING for license information (GPL). If you
# did not receive this file, see https://www.gnu.org/licenses/gpl-3.0.txt.
# --
```

可执行文件(`*.pl`)有一个特殊的文件头 。

```
#!/usr/bin/perl
# --
# Copyright (C) 2001-2020 OTRS AG, https://otrs.com/
# --
# This program is free software: you can redistribute it and/or modify
# it under the terms of the GNU General Public License as published by
# the Free Software Foundation, either version 3 of the License, or
# (at your option) any later version.
#
# This program is distributed in the hope that it will be useful,
# but WITHOUT ANY WARRANTY; without even the implied warranty of
# MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
# GNU General Public License for more details.
#
# You should have received a copy of the GNU General Public License
# along with this program. If not, see https://www.gnu.org/licenses/gpl-3.0.
↪txt.
# --
```

条件

条件可能非常复杂，可能存在 链式条件(与逻辑 *or* 或 *and* 操作链接)。在编写 OTRS 时，您必须了解几种情况 。

Perl 最佳实践说，高优先级运算符(`&&` 和 `||`)不应与低优先级运算符(`and`` 和 ``or`)混淆 。为避免混淆，我们总是使用高优先级运算符 。

```
if ( $Condition1 && $Condition2 ) { ... }

# instead of

if ( $Condition and $Condition2 ) { ... }
```

这意味着您必须了解陷阱 。有时您需要使用括号来明确您想要的内容 。

如果条件很长(行总长度超过 120 个字符)，则必须将它分成几行 。条件的开始是一个新的行(不在 `if` 的行中)。

```
if (
    $Condition1
    && $Condition2
    )
{ ... }

# instead of

if ( $Condition1
    && $Condition2
    )
{ ... }
```

另请注意，右括号本身在一行中，左括号也在一个新行中，并且与 `if` 具有相同的缩进 。运算符处于新行的开头！随后的示例显示了如何执行此操作 。

```
if (
    $XMLHash[0]->{otrs_stats}[1]{StatType}[1]{Content}
    && $XMLHash[0]->{otrs_stats}[1]{StatType}[1]{Content} eq 'static'
    )
{ ... }

if ( $TemplateName eq 'AgentTicketCustomer' ) {
    ...
}

if (
    ( $Param{Section} eq 'Xaxis' || $Param{Section} eq 'All' )
    && $StatData{StatType} eq 'dynamic'
    )
{ ... }

if (
    $Self->{TimeObject}->TimeStamp2SystemTime( String => $Cell->{TimeStop} )
    > $Self->{TimeObject}->TimeStamp2SystemTime(
        String => $ValueSeries{$Row}{$TimeStop}
    )
    || $Self->{TimeObject}->TimeStamp2SystemTime( String => $Cell->{TimeStart}
↪ )
    < $Self->{TimeObject}->TimeStamp2SystemTime(
```

(下页继续)

(续上页)

```
            String => $ValueSeries{$Row}{$TimeStart}
        )
        )
{ ... }
```

后缀 if

通常我们使用 后缀 *"if"* 语句来减少级别数 。但我们不将它用于多行语句，只有在函数中包含 return 语句或结束循环或进行下一次迭代时才允许使用它 。

这样是对的：

```
next ITEM if !$ItemId;
```

这样是错的：

```
return $Self->{LogObject}->Log(
    Priority => 'error',
    Message  => 'ItemID needed!',
) if !$ItemId;
```

这样更容易维护：

```
if( !$ItemId ) {
    $Self->{LogObject}->Log( ... );
    return;
}
```

这样是对的：

```
for my $Needed ( 1 .. 10 ) {
    next if $Needed == 5;
    last  if $Needed == 9;
}
```

这样是错的：

```
my $Var = 1 if $Something == 'Yes';
```

使用某些 Perl 内置函数的限制

Perl 的一些内置子程序不适合在每个地方使用：

- 不要在 .pm 文件中使用 die 和 exit 。
- 不要在已发布的文件中使用 Dumper 函数 。
- 不要在 .pm 文件中使用 print 。
- 不要使用 require ，而是使用 Main::Require() 。
- 使用 DateTimeObject 的函数代替 time() 、localtime() 等内置函数 。

正则表达式

对于 源代码中的正则表达式，我们总是使用带有花括号的 m// 运算符作为分隔符 。我们默认使用修饰符 x 、m 和 s 。x 修饰符允许您注释正则表达式并使用空格来可视地分隔逻辑组 。

```
$Date =~ m{ \A \d{4} - \d{2} - \d{2} \z }xms
$Date =~ m{
    \A      # beginning of the string
    \d{4} - # year
    \d{2} - # month
    [^\n]   # everything but newline
    #..
}xms;
```

由于空格不再具有特殊含义，您必须使用单个字符类来匹配单个空格([ ])。如果你想匹配任何空格，你可以使用 \s 。

在正则表达式中，点(.)包括换行符(而在没有 s 修饰符的正则表达式中，点表示“除了换行符之外的所有内容”)。如果你想匹配除换行之外的任何东西，你必须使用否定的单个字符类([^\n])。

```
$Text =~ m{
    Test
    [ ]     # there must be a space between 'Test' and 'Regex'
    Regex
}xms;
```

上述约定的适用于所有情况，一个例外是正则表达式不是在代码中静态写入，而是由用户 以一种或另一种形式提供(例如通过系统配置或邮箱管理员过滤器配置)。任何对这种正则表达式的评估都必须在没有任何修饰符的情况下完成(例如 $Variable =~ m{$Regex})，以匹配(大部分是没有经验的)用户的期望，并且也是向后兼容的 。

如果修饰符对于用户提供的正则表达式是严格必需的，则始终可以使用嵌入式修饰符(例如 (?:(?i)SmAlL oR lArGe))。有关详细信息，请参阅 perlretut 。

鼓励使用 r 修饰符，例如如果您需要将字符串的一部分提取到另一个变量中 。此修饰符使匹配的变量保持不变，而是将替换结果作为返回值提供 。

这样使用：

```
my $NewText = $Text =~ s{
    \A
    Prefix
    (
        Text
    )
}
{NewPrefix$1Postfix}xmsr;
```

而不是这样使用：

```
my $NewText = $Text;
$NewText =~ s{
    \A
    Prefix
    (
        Text
```

(下页继续)

(续上页)

```
    )
}
{NewPrefix$1Postfix}xms;
```

如果你想匹配 字符串的开头和结尾，你通常应该使用 \A 和 \z 而不是更通用的 ^ 和 $，除非你真的需要在多行字符串中匹配 行的开头或结尾 。

```
$Text =~ m{
    \A      # beginning of the string
    Content # some string
    \z      # end of the string
}xms;

$MultilineText =~ m{
    \A                      # beginning of the string
    .*
    (?: \n Content $ )+      # one or more lines containing the same string
    .*
    \z                      # end of the string
}xms;
```

还鼓励使用命名捕获组，尤其是对于多个匹配 。命名捕获组更易于阅读/理解，在匹配多个捕获组时防止混淆，并允许扩展而不会意外地引入错误 。

这样使用：

```
$Contact =~ s{
    \A
    [ ]*
    (?'TrimmedContact'
        (?'FirstName' \w+ )
        [ ]+
        (?'LastName' \w+ )
    )
    [ ]+
    (?'Email' [^ ]+ )
    [ ]*
    \z
}
{$+{TrimmedContact}}xms;
my $FormattedContact = "$+{LastName}, $+{FirstName} ($+{Email})";
```

而不是这样使用：

```
$Contact =~ s{
    \A
    [ ]*
    (
        ( \w+ )
        [ ]+
        ( \w+ )
    )
    [ ]+
```

(下页继续)

(续上页)

```
    ( [^ ]+ )
    [ ]*
    \z
}
{$1}xms;
my $FormattedContact = "$3, $2 ($4)";
```

命名

名称和注释都用英语。变量、对象和方法必须是描述性名词或名词短语，第一个字母设置为大写（驼峰式命名法）。

名称应尽可能具有描述性。读者应该能够在不深入挖掘代码的情况下说出名称的含义。例如。使用 `$ConfigItemID` 而不是 `$ID`。例如：`@TicketIDs`、`$Output`、`StateSet()` 等。

变量声明

如果有多个变量且属于一起，则可以在一行中声明它们：

```
my ( $Minute, $Hour, $Year );
```

否则，将其分成几行：

```
my $Minute;
my $ID;
```

不要在声明中设置为 `undef` 或 `''`，因为这可能会隐藏代码中的错误。

```
my $Variable = undef;

# is the same as

my $Variable;
```

如果要连接字符串，可以将变量设置为 `''`：

```
my $SqlStatement = '';
for my $Part (@Parts) {
    $SqlStatement .= $Part;
}
```

否则你会收到 *uninitialized*（未初始化）警告。

处理参数

为了获取传递给子程序的参数，OTRS 通常使用散列 `%Param`（不是 `%Params`）。这导致代码更易读，因为每次在子程序代码中使用 `%Param` 时我们都知道它是传递给子程序的参数哈希。

除了某些例外情况，应使用常规参数列表。所以我们想避免这样的事情：

```
sub TestSub {
    my ( $Self, $Param1, $Param2 ) = @_;
}
```

我们想要这样使用：

```
sub TestSub {
    my ( $Self, %Param ) = @_;
}
```

这有几个好处：

- 当要传递新参数时，我们不必更改子程序中的代码 。
- 使用命名参数调用函数更具可读性 。

多个命名参数

如果函数调用需要多个命名参数，请将它们拆分为多行 。

这样使用：

```
$Self->{LogObject}->Log(
    Priority => 'error',
    Message  => "Need $Needed!",
);
```

而不是这样使用：

```
$Self->{LogObject}->Log( Priority => 'error', Message  => "Need $Needed!", );
```

return 语句

子程序必须有一个 return 语句 。显式的 return 语句优于隐式方式（ 子程序中最后一个语句的结果 ），因为这澄清了子程序返回的内容 。

```
sub TestSub {
    ...
    return; # return undef, but not the result of the last statement
}
```

显式返回值

显式返回值意味着您不应该有一个 return 语句，后面跟一个子程序调用 。

```
return $Self->{DBObject}->Do( ... );
```

以下示例更好，因为它明确说明了返回的内容 。上面的例子，读者不知道返回值是什么，因为他可能不知道 Do() 返回什么 。

```
return if !$Self->{DBObject}->Do( ... );
return 1;
```

如果将子程序的结果分配给变量，则一个 良好的变量名称可以表明返回的内容：

```
my $SuccessfulInsert = $Self->{DBObject}->Do( ... );
return $SuccessfulInsert;
```

use 语句

use strict 和 use warnings 必须是模块中前两个 *use* 语句 。

这样是对的：

```
package Kernel::System::ITSMConfigItem::History;

use strict;
use warnings;

use Kernel::System::User;
use Kernel::System::DateTime;
```

这样是错的：

```
package Kernel::System::ITSMConfigItem::History;

use Kernel::System::User;
use Kernel::System::DateTime;

use strict;
use warnings;
```

对象及其分配

在 OTRS 中有许多对象可用 。但是你不应该使用每个文件中的每个对象来保持前端/后端的分离 。

- 不要在核心模块 (Kernel/System) 中使用 LayoutObject 。
- 不要在核心模块 (Kernel/System) 中使用 ParamObject 。
- 不要在核心模块 (Kernel/System) 中使用 DBObject 。

后端模块文档记录

NAME 部分 本部分应包括模块名称，“ - “ 作为分隔符和模块目的的简要说明 。

```
=head1 NAME

Kernel::System::MyModule - Functions to read from and write to files
```

SYNOPSIS 部分 本部分应提供常用模块功能的简短用法示例 。

使用此部分是可选的 。

```
=head1 SYNOPSIS

my $Object = $Kernel::OM->Get('Kernel::System::MyModule');

Read data

    my $FileContent = $Object->Read(
        File => '/tmp/testfile',
    );

Write data

    $Object->Write(
        Content => 'my file content',
        File    => '/tmp/testfile',
    );
```

DESCRIPTION 部分 如果认为有必要，本部分应该提供有关模块的更多深入信息（ 而不是有个很长的 NAME 部分 ）。

使用此部分是可选的 。

```
=head1 DESCRIPTION

This module does not only handle files.

It is also able to:
- brew coffee
- turn lead into gold
- bring world peace
```

PUBLIC INTERFACE 部分 本部分标记了作为 API 一部分的所有函数的开头，因此意味着由其它模块使用 。

```
=head1 PUBLIC INTERFACE
```

PRIVATE FUNCTIONS 部分 本部分标志着私有函数的开始 。

以下函数不是 API 的一部分，仅在模块中使用，因此不被认为是稳定的 。

只要存在一个或多个私有函数，建议使用此部分 。

```
=head1 PRIVATE FUNCTIONS
```

记录子程序

应始终记录子程序文档 。该文档包含有关子程序的作用 、示例子程序调用以及子程序返回的内容的一般说明 。它应该按此顺序排列 。示例文档如下所示：

```
=head2 LastTimeObjectChanged()

Calculates the last time the object was changed. It returns a hash reference␣
↪with
    information about the object and the time.
```

(下页继续)

(续上页)

```
    my $Info = $Object->LastTimeObjectChanged(
        Param => 'Value',
    );

This returns something like:

    my $Info = {
        ConfigItemID    => 1234,
        HistoryType      => 'foo',
        LastTimeChanged => '08.10.2009',
    };

=cut
```

您可以复制并粘贴 `Data::Dumper` 输出作为返回值 。

Perl 中的代码注释

通常，您应该尝试将代码编写为可读和自我解释 。不要写注释来解释明显代码的作用，这是不必要的重复 。好的注释应该解释 为什么有一些代码 、可能的副作用以及任何可能特殊或异常复杂的代码 。

请遵守以下准则：

如果可能的话，使代码具有可读性，以便不需要注释 。 编写非常易读和自我解释的代码总是比较好，例如使用精确的变量和函数名称 。

不要说代码所说的(不要重复)。 不要在注释中重复(明显的)代码 。

```
# WRONG:

# get config object
my $ConfigObject = $Kernel::OM->Get('Kernel::Config');
```

记录代码 为什么在那里，而不是它的工作方式 。 通常，代码注释应该解释代码的 目的，而不是它的详细工作原理 。特殊复杂代码可能有例外，但在这种情况下，重构以使其更具可读性可能是值得称道的 。

记录陷阱 。 在开发过程中，所有不清楚 、棘手或困扰您的事情都应该记录下来 。

使用整行句子样式注释来记录算法段落 。 始终使用完整句子(大写第一个字母和最后一个句点)。句子的后续行应缩进 。

```
# Check if object name is provided.
if ( !$_[1] ) {
    $_[0]->_DieWithError(
        Error => "Error: Missing parameter (object name)",
    );
}

# Record the object we are about to retrieve to potentially build better␣
↪error messages.
# Needs to be a statement-modifying 'if', otherwise 'local' is local
#   to the scope of the 'if'-block.
local $CurrentObject = $_[1] if !$CurrentObject;
```

使用简短的行尾注释添加详细信息 。 这些可以是一个完整的句子(大写首字母和句点)，也可以只是一个短语(小写首字母和无句点)。

```
$BuildMode = oct $Param{Mode};    # *from* octal, not *to* octal

# or

$BuildMode = oct $Param{Mode};    # Convert *from* octal, not *to* octal.
```

SQL 语句的声明

如果没有机会更改 SQL 语句，则应在 Prepare 函数中使用它 。原因是，SQL 语句和绑定参数彼此更接近 。

SQL 语句应该写成一个缩进良好的字符串，没有连接，如下所示：

```
return if !$Self->{DBObject}->Prepare(
    SQL => '
        SELECT art.id
        FROM article art, article_sender_type ast
        WHERE art.ticket_id = ?
            AND art.article_sender_type_id = ast.id
            AND ast.name = ?
        ORDER BY art.id',
    Bind => [ \$Param{TicketID}, \$Param{SenderType} ],
);
```

这样很容易阅读和修改，我们支持的 DBMS 可以很好地处理空白 。对于自动生成的 SQL 代码(如 TicketSearch)，此缩进不是必需的 。

出错时返回

无论何时使用数据库函数，都应该处理错误 。如果出现任何错误，请从子程序返回：

```
return if !$Self->{DBObject}->Prepare( ... );
```

使用 Limit

如果你希望只返回一行，使用 Limit => 1 。

```
$Self->{DBObject}->Prepare(
    SQL   => 'SELECT id FROM users WHERE username = ?',
    Bind  => [ \$Username ],
    Limit => 1,
);
```

使用 while 循环

总是使用 while 循环，即使你期望返回一行，因为一些数据库不释放语句句柄，这可能导致奇怪的错误 。

6.3.2 JavaScript

在所有的浏览器中所有的 JavaScript 都会加载（ 模板文件中没有浏览器入侵 ）。代码负责决定是否必须跳过或仅在某些浏览器中执行其自身的某些部分 。

目录结构

`var/httpd/htdocs/js/` 文件夹内的目录结构：

```
* js
    * thirdparty                # thirdparty libs always have the version␣
↪number inside the directory
        * ckeditor-3.0.1
        * jquery-1.3.2
    * Core.Agent.*              # stuff specific to the agent interface
    * Core.Customer.*           # customer interface
    * Core.*                    # common API
```

第三方代码

每个第三方模块都有自己的子目录：模块名称-版本号（ 例如，ckeditor-4.7.0 、jquery-3.2.1 ）。其中，文件名不应包含版本号或后缀（ 错误：`jquery/jquery-3.2.1.min.js` ，正确：`jquery-3.2.1/jquery.js` ）。

JavaScript 变量

变量名应该使用骆驼拼写法，就像在 Perl 中一样 。

保存 jQuery 对象的变量应以 `$` 开头，例如：`$Tooltip` 。

函数

函数名应该使用骆驼拼写法，就像在 Perl 中一样 。

命名空间

JavaScript 中的代码注释

Perl 中的代码注释 也适用于 javascript 。

- 单行注释用 `//` 完成 。
- 使用 `/ * ... */` 进行更长的注释 。
- 如果你注释掉 JavaScript 代码的一部分，只使用 `//` ，因为 `/* ... */` 会导致代码中的正则表达式出现问题 。

事件处理

总是使用 `$.on()` 代替 jQuery 的事件简写方法以获得更好的可读性(错误：`$SomeObject.click(...)` ，正确：`$SomeObject.on('click', ...)`)。

如果你 `$.on()` 事件，请确保事先 `$.off()` 它们，以确保如果代码再次执行，事件不会绑定两次 。

确保在命名空间中使用 `$.on()` ，比如 `$.on('click.<Name>')` 。

6.3.3 HTML

使用 HTML 5 表示法 。不要将自闭合标签用于非空元素(例如 `div` 、`span` 等)。

使用正确的缩进 。包含其它非空子元素的元素不应与子元素处于同一级别 。

不要出于布局原因使用 HTML 元素(例如，使用 `br` 元素为其它元素的顶部或底部添加空间)。请改用适当的 CSS 类 。

不要使用内联 CSS 。所有的 CSS 应该使用预定义的类或者(如果需要的话)使用 JavaScript 来添加(例如用于显示/隐藏元素)。

不要在模板中使用 JavaScript 。对于某个前端模块或适当的全局库，所有需要的 JavaScript 都应该是适当库的一部分 。如果需要将 JavaScript 数据传递到前端，请使用 `$LayoutObject->AddJSData()` 。

6.3.4 CSS

最小分辨率为 1024x768 像素 。

布局是流动的，这意味着如果屏幕变宽，空间也会被使用 。

应该以 px(像素)指定绝对大小的测量值，以在许多平台和浏览器上保持一致的外观 。

文档是使用 CSSDOC 制作的(请参阅 CSS 文件的示例)。所有的逻辑块应该有一个 CSSDOC 注释 。

CSS 体系结构

我们遵循 面向对象的 CSS 方法 。本质上，这意味着布局是通过组合不同的通用构建块来实现特定的设计 。

只要有可能，就不应使用模块特定的设计 。因此，如果可以避免的话，我们也不使用 `body` 元素上的 ID 。

CSS 样式

所有定义在选择器的同一行中都有一个 `{` ，所有规则都按每个规则一行定义，定义以一个单行的 `}` 结束 。

请参阅以下示例：

```
#Selector {
    width: 10px;
    height: 20px;
    padding: 4px;
}
```

- 在 `:` 和规则值之间，有一个空格 。
- 每条规则都有 4 个空格的缩进 。
- 如果指定了多个选择器，请用逗号分隔它们，并且一行一个：

```
#Selector1,
#Selector2,
#Selector3 {
    width: 10px;
}
```

- 如果规则是可组合的，则将它们组合起来（ 例如将 background-position 、background-image 等组合成 background ）。
- 规则应该在定义内按逻辑顺序排列（ 所有颜色特定规则在一起 、所有定位规则在一起等 ）。
- 所有 ID 和名称都以驼峰式命名法编写：

```
<div class="NavigationBar" id="AdminMenu"></div>
```

6.4 用户界面设计

6.4.1 字母大写样式

本节讨论如何将英语用户界面的不同部分大写 。有关详细信息，您可能需要查看 这个有用的页面 。

标题（ h1-h6 ）和标题（ 名称，如 队列视图 ）设置为 标题样式大小写，这意味着所有首字母都将大写（ 少数例外情况，如 *this* 、*and* 、*or* 等 ）。

例子：

- *Action List*
- 管理客户-组的关联

按钮 、标签 、选项卡 、菜单项等其它结构元素以 句子样式大写形式设置（ 只有短语的第一个字母大写 ），但不添加最后一个点以完成句子形式的短语 。

例子：

- 名
- 选择队列刷新时间
- 打印此工单

描述性文本和提示内容被写为完整句子 。

例如：

- 此值为必填项 。

翻译时必须检查标题大写样式是否也适合目标语言，它可能需要改为句子大写样式或别的 。

6.5 无障碍环境指南

本文试图解释关于无障碍环境问题的基础知识，并给予指导方针，以便为 OTRS 做出贡献 。

6.5.1 无障碍环境基础

什么是无障碍环境？

无障碍环境是一个通用术语，用于描述尽可能多的人可以访问产品 、设备 、服务或环境的程度 。无障碍环境可被视为 * 访问 * 的能力以及某些系统或实体的可能益处 。可访问性通常用于关注残疾人及其访问实体的权利，通常通过使用辅助技术 。

在 Web 开发的上下文中，无障碍环境集中于使受损害的人能够完全访问 Web 界面 。例如，这群人可能包括部分视力障碍或完全失明的人 。虽然前者仍然可以部分使用 GUI，但是后者必须完全依赖辅助技术，例如向他们读屏幕的软件（ 屏幕阅读软件 ）。

为什么它对 **OTRS** 很重要？

使受损用户访问 OTRS 系统本身是一个有效的目标 。它体现了尊重 。

此外，在属于 OTRS 目标市场的公共部门（ 政府机构 ）和大型公司，实现无障碍环境标准变得越来越重要 。

如果我没有受损，我如何成功地处理无障碍环境问题？

这很简单，假装失明 。

- 不要使用鼠标 。
- 不要看屏幕 。

然后尝试在屏幕阅读器和键盘的帮助下使用 OTRS 。这应该可以让您了解盲人的感受 。

好的，但我没有屏幕阅读软件！

虽然诸如 JAWS（ 可能是最著名的 ）的商业屏幕阅读软件可能非常昂贵，但是有一些可以安装和使用的开源屏幕阅读软件：

- NVDA，用于 Windows 的屏幕阅读器 。
- ORCA，用于 Gnome/Linux 的屏幕阅读器 。

现在你再没有借口了 。;)

6.5.2 无障碍环境标准

本部分仅供参考，您无需研究标准本身就能够处理 OTRS 中的可访问性问题 。我们将尝试在本文档中提取相关指南 。

网页内容无障碍环境指南（ **WCAG** ）

这个 W3C 标准为如何创建可访问的网页提供了通用指南 。

- WCAG 2.0
- 如何满足 WCAG 2.0
- 理解 WCAG 2.0

WCAG 具有不同级别的可访问性支持 。我们目前计划支持 A 级，因为 AA 和 AAA 处理与 OTRS 无关的事项 。

可访问的互联网应用（ WAI-ARIA ）1.0

本标准涉及从静态内容转移到动态 Web 应用程序引起的特殊问题 。它处理诸如用户如何从 AJAX 请求产生的用户界面变化得到通知之类的问题 。

- WAI-ARIA 1.0

6.5.3 实施指南

提供非文本内容的替代方案

目标：呈现给用户的所有非文本内容都有一个文本替代方案，用于同等目的（ WCAG 1.1.1 ）

理解屏幕阅读器只能向用户呈现文本信息和可用元数据是非常重要的 。举个例子，每当屏幕阅读器看到 `<a href="#" class="CloseLink"></a>` 时，它只能读取 * 链接 给用户，而不是这个链接的目标 。稍微改进一下就可以访问：*““* 。在这种情况下，用户会听到 * 链接关闭这个小部件，瞧！

始终以最 口语的方式表述文本非常重要 。想象一下，这是你拥有的唯一信息，它会帮到你吗？你只听到它就能理解它的目的吗？

使用 OTRS 时请遵从这些规则：

- 规则：尽可能使用口头文本并用真实 、可理解和精确的句子表达 。关闭这个小部件比 关闭好得多，因为后者是冗余的 。
- 规则：链接始终必须具有屏幕阅读器能说的文本内容（ `<a href="#">` 删除此条目 `</a>` ）或 `title` 属性（ `<a href="#" title=" 关闭这个小部件"></a>` ）。
- 规则：图片必须始终有一个可以读给用户的替代文本（ `<img src="house.png" alt=" 一座房子的图片" />` ）。

使导航变得容易

目标：允许用户轻松浏览当前页面和整个应用程序 。

`title` 标签是用户在打开网页时从屏幕阅读器听到的第一项 。对于 OTRS，页面上总是只有一个 `h1` 元素，表示当前页面（ 它包含来自 `title` 的部分信息 ）。该导航信息有助于用户了解它们的位置以及当前页面的用途 。

- 规则：始终为页面提供一个精确的标题，允许用户了解它们当前的位置 。

屏幕阅读器可以使用 HTML 的内置文档结构（ 标题 `h1` 到 `h6` ）来确定文档的结构，并允许用户从一节跳到另一节 。但是，这不足以反映动态 Web 应用程序的结构 。这就是为什么 ARIA 定义了几个 地标角色，这些角色可以赋予元素以指示其导航意义 。

为了保持 HTML 文档的有效性，`role` 属性（ ARIA 地标角色 ）不是直接插入到源代码中，而是通过以后将被 `OTRS.UIAccessibility` 中的 JavaScript 函数使用的类插入，在节点上设置相应的 `role` 属性 。

- 规则：使用 WAI-ARIA 地标角色为屏幕阅读器组织内容 。
  - Banner: `<div class="ARIARoleBanner"></div>` 将 变 成 `<div class="ARIARoleBanner" role="banner"></div>`
  - 导 航: `<div class="ARIARoleNavigation"></div>` 将 变 成 `<div class="ARIARoleNavigation" role="navigation"></div>`
  - 搜 索 功 能: `<div class="ARIARoleSearch"></div>` 将 变 成 `<div class="ARIARoleSearch" role="search"></div>`

  - 主应用区域: `<div class="ARIARoleMain"></div>` 将变成 `<div class="ARIARoleMain" role="main"></div>`
  - 页脚: `<div class="ARIARoleContentinfo"></div>` 将变成 `<div class="ARIARoleContentinfo" role="contentinfo"></div>`

对于 `<form>` 元素内部的导航，受损用户必须知道每个输入元素的用途。这可以通过使用标准 HTML `<label>` 元素来实现，这些元素在标签和表单元素之间创建链接。

当输入元素获得焦点时，屏幕阅读器通常会读取连接的标签，以便用户可以听到其确切用途。正常用户的额外好处是，他们可以通过点击标签让此输入元素将获得焦点（例如，对复选框特别有用）。

- 规则：为所有的表单元素（`input` 文本输入框、`select` 选择框、`textarea` 多行文本框）字段提供 `<label>` 元素。

  例如：`<label for="date">日期:</label><input type="text" name="date" id="date"/>`

使交互成为可能

目标：允许用户仅使用键盘执行所有交互。

虽然在技术上可以在任意 HTML 元素上创建与 JavaScript 的交互，但这必须限于用户可以使用键盘与之交互的元素。具体来说，他们需要能够获取元素的焦点并与元素进行交互。例如，切换小部件的按钮不应该通过使用带有附加的 JavaScript `onclick` 事件监听器的 `span` 元素来实现，而应该是（或包含）`a` 标签使屏幕阅读器清楚这个元素可以引起交互。

- 规则：对于交互，总是使用可以获得焦点的元素，例如 `a`、`input`、`select` 和 `button`。
- 规则：确保用户始终能够识别交互的性质（请参阅有关非文本内容和表单元素标签的规则）。

目标：让用户了解动态变化。

无障碍环境问题的一个特殊领域是用户界面中的动态更改，可以是 JavaScript，也可以是 AJAX 调用。如果没有特殊的预防措施，屏幕阅读器不会告诉用户有关变化。这是一个困难的话题，这里还不能完全解释。

- 规则：始终使用验证框架 `OTRS.Validate` 进行表单验证。

  这将确保屏幕阅读器读取错误提醒。这样，盲人用户 a）知道有错误的项目，b）得到描述错误的文本。

- 规则：使用函数 `OTRS.UI.Accessibility.AudibleAlert()` 通知用户其它重要的用户界面变化。
- 规则：使用 `OTRS.UI.Dialog` 框架创建模态对话框。这些已经针对无障碍环境进行了优化。

通用屏幕阅读软件优化

目标：帮助屏幕阅读器完成工作。

- 规则：每个页面必须标识自己的主语言，以便屏幕阅读器可以选择正确的语音合成引擎。

  例如：`<html lang="fr">...</html>`

6.6 单元测试

OTRS 提供了一个测试套件，可用于开发和运行所有系统相关代码的单元测试。

6.6.1 创建一个测试文件

测试文件存储在 scripts/test/*.t 下的 .t 文件中。例如，让我们看一下 *calendar*（ 日历 ）类的文件 scripts/test/calendar.t 。

每个测试文件都应该在开始时实例化单元测试 helper 对象，以便它可以从它提供的一些内置方法中受益：

```
# --
# Copyright (C) 2001-2020 OTRS AG, https://otrs.com/
# --
# This software comes with ABSOLUTELY NO WARRANTY. For details, see
# the enclosed file COPYING for license information (GPL). If you
# did not receive this file, see https://www.gnu.org/licenses/gpl-3.0.txt.
# --

use strict;
use warnings;
use utf8;

use vars (qw($Self));

$Kernel::OM->ObjectParamAdd(
    'Kernel::System::UnitTest::Helper' => {
        RestoreDatabase => 1,
    },
);
my $Helper = $Kernel::OM->Get('Kernel::System::UnitTest::Helper');
```

通过向 helper 构造函数提供 RestoreDatabase 参数，在单元测试期间执行的任何数据库语句都将在结束时回滚，确保没有进行永久性更改。

像任何其它测试套件一样，OTRS 提供了可用于测试条件的断言方法。例如，下面就是我们如何创建测试用户并测试它确实已经创建的：

```
my $UserLogin = $Helper->TestUserCreate();
my $UserID = $UserObject->UserLookup( UserLogin => $UserLogin );

$Self->True(
    $UserID,
    "Test user $UserID created"
);
```

有关断言方法的完整列表，请参阅下面的 API 部分。

在单元测试中创建随机数据总是一个很好的做法，这可以帮助将它与以前添加的数据区分开来。使用 API 中的随机方法获取字符串并将其包含在你的参数中：

```
my $RandomID = $Helper->GetRandomID();

# create test group
my $GroupName = 'test-calendar-group-' . $RandomID;
my $GroupID   = $GroupObject->GroupAdd(
    Name    => $GroupName,
    ValidID => 1,
    UserID  => 1,
```

(下页继续)

(续上页)

```
);

$Self->True(
    $GroupID,
    "Test group $GroupID created"
);
```

优秀的开发人员使他们的单元测试易于维护。考虑将所有的测试用例放在一个数组中，然后用一些代码对它们进行迭代。这将为以后扩展测试提供一个简单的方法：

```
#
# Tests for CalendarCreate()
#
my @Tests = (
    {
        Name     => 'CalendarCreate - No params',
        Config   => {},
        Success => 0,
    },
    {
        Name    => 'CalendarCreate - All required parameters',
        Config => {
            CalendarName => "Calendar-$RandomID",
            Color        => '#3A87AD',
            GroupID      => $GroupID,
            UserID       => $UserID,
        },
        Success => 1,
    },
    {
        Name    => 'CalendarCreate - Same name',
        Config => {
            CalendarName => "Calendar-$RandomID",
            Color        => '#3A87AD',
            GroupID      => $GroupID,
            UserID       => $UserID,
        },
        Success => 0,
    },
);

for my $Test (@Tests) {

    # make the call
    my %Calendar = $CalendarObject->CalendarCreate(
        %{ $Test->{Config} },
    );

    # check data
    if ( $Test->{Success} ) {
        for my $Key (qw(CalendarID GroupID CalendarName Color CreateTime␣
↪CreateBy ChangeTime ChangeBy ValidID)) {
```

(下页继续)

(续上页)

```
            $Self->True(
                $Calendar{$Key},
                "$Test->{Name} - $Key exists",
            );
        }

        KEY:
        for my $Key ( sort keys %{ $Test->{Config} } ) {
            next KEY if $Key eq 'UserID';

            $Self->IsDeeply(
                $Test->{Config}->{$Key},
                $Calendar{$Key},
                "$Test->{Name} - Data for $Key",
            );
        }
    }
    else {
        $Self->False(
            $Calendar{CalendarID},
            "$Test->{Name} - No success",
        );
    }
}
```

6.6.2 测试的先决条件

为了能够运行单元测试，您需要安装所有可选的环境依赖项（ Perl 模块和其它模块 ），除了那些用于不同数据库后端的依赖项以外 。运行 `bin/otrs.checkenvironment.pl` 验证模块安装 。

您还需要在本地 OTRS 的 `Config.pm` 文件中配置的 FQDN 上运行 OTRS Web 前端的实例 。此 OTRS 实例必须使用为单元测试配置的相同数据库 。

6.6.3 测试

若要运行你的测试，只需执行 `bin/otrs.Console.pl Dev::UnitTest::Run --test Calendar` 即可使用 `scripts/test/Calendar.t` 。

```
shell:/opt/otrs> bin/otrs.Console.pl Dev::UnitTest::Run --test Calendar
+-------------------------------------------------------------------+
/opt/otrs/scripts/test/Calendar.t:
+-------------------------------------------------------------------+
.....................................................................
 ↪..................
=====================================================================
yourhost ran tests in 2s for OTRS 6.0.x git
All 97 tests passed.
shell:/opt/otrs>
```

您甚至可以一次运行多个测试，只需为命令提供额外的 `--test` 参数：

```
shell:/opt/otrs> bin/otrs.Console.pl Dev::UnitTest::Run --test Calendar --
↪test Appointment
+-------------------------------------------------------------------+
/opt/otrs/scripts/test/Calendar.t:
+-------------------------------------------------------------------+
.....................................................................
↪..................
+-------------------------------------------------------------------+
/opt/otrs/scripts/test/Calendar/Appointment.t:
+-------------------------------------------------------------------+
.....................................................................
↪.................................
====================================================================
yourhost ran tests in 5s for OTRS 6.0.x git
All 212 tests passed.
shell:/opt/otrs>
```

如果你不带任何参数执行 `bin/otrs.Console.pl Dev::UnitTest::Run`，它将运行系统中找到的所有测试。请注意，这可能需要一些时间才能完成。

还提供 `--verbose` 参数，以便查看有关成功测试的消息。测试过程中遇到的任何错误都将显示出来，只要它们在测试中被实际引发，而不管是否有这个开关。

6.6.4 单元测试 API

OTRS 为上一个示例中使用的单元测试提供 API。在这里，我们将列出最重要的函数，请参阅 Kernel::System::UnitTest 的在线 API 参考。

True() 这个函数测试给定的标量值在 Perl 中是否为真值。

```
$Self->True(
    1,
    'Scalar 1 is always evaluated as true'
);
```

False() 这个函数测试给定的标量值在 Perl 中是否为假值。

```
$Self->False(
    0,
    'Scalar 0 is always evaluated as false'
);
```

Is() 这个函数测试给定的标量变量是否相等。

```
$Self->Is(
    $A,
    $B,
    'Test Name',
);
```

IsNot() 这个函数测试给定的标量变量是否不相等。

```
$Self->IsNot(
    $A,
    $B,
    'Test Name'
);
```

IsDeeply() 这个函数比较复杂的数据结构是否相等。$A 和 $B 必须是引用。

```
$Self->IsDeeply(
    $A,
    $B,
    'Test Name'
);
```

IsNotDeeply() 这个函数比较复杂的数据结构是否不相等。$A 和 $B 必须是引用。

```
$Self->IsNotDeeply(
    $A,
    $B,
    'Test Name'
);
```

除此之外，单元测试 helper 对象还为常见的测试条件提供了一些有用的方法。如需完整参考，请参阅 Kernel::System::UnitTest::Helper 的在线 API 参考。

GetRandomID() 这个函数创建一个随机 ID，可以在测试中用作唯一标识符。这可以保证在一个测试内，这个函数永远不会返回一个重复。

> 注解：请注意，这些数字并不是真正的随机数，只能用于创建测试数据。

```
my $RandomID = $Helper->GetRandomID();
# $RandomID = 'test6326004144100003';
```

TestUserCreate() 这个函数创建一个可以在测试中使用的测试用户。它将在析构函数中自动设置为无效。它返回新用户的登录名，密码与用户名相同。

```
my $TestUserLogin = $Helper->TestUserCreate(
    Groups   => ['admin', 'users'],          # optional, list of groups␣
↪to add this user to (rw rights)
    Language => 'de',                        # optional, defaults to 'en'␣
↪if not set
);
```

FixedTimeSet() 只要此对象存在，这个函数就可以覆盖系统时间。你可以传递要使用的可选时间参数，如果没有参数，则会使用当前系统时间。

> 注解：对 Kernel::System::Time 和 Kernel::System::DateTime 方法的所有调用都将使用给定的时间。

```
$HelperObject->FixedTimeSet(366475757);           # with Timestamp
$HelperObject->FixedTimeSet($DateTimeObject);     # with previously created␣
↪DateTime object
$HelperObject->FixedTimeSet();                    # set to current date and␣
↪time
```

FixedTimeUnset() 这个函数恢复常规的系统时间行为 。

FixedTimeAddSeconds() 这个函数为固定时间增加几秒钟，这个时间先前由 FixedTimeSet() 设置 。您可以通过一个负值返回之前的时间 。

ConfigSettingChange() 这个函数临时将系统范围内的配置设置更改为另一个值，既在 ConfigObject 的当前实例中，也在磁盘上的系统配置中 。当 Helper 对象被销毁时，这将被重置 。

注解: 请注意，这个函数在集群环境中无法正常工作 。

```
$Helper->ConfigSettingChange(
    Valid => 1,             # (optional) enable or disable setting
    Key   => 'MySetting',   # setting name
    Value => { ... } ,      # setting value
);
```

CustomCodeActivate() 这个函数将在系统中临时包含自定义代码 。例如，可以使用它从另一个类重新定义子程序 。此更改将在测试的其余部分持续 。当 Helper 对象被销毁时，所有代码都将被删除 。

注解: 请注意，这个函数在集群环境中无法正常工作 。

```
$Helper->CustomCodeActivate(
    Code => q^
use Kernel::System::WebUserAgent;
package Kernel::System::WebUserAgent;
use strict;
use warnings;
{
    no warnings 'redefine';
    sub Request {
        my $JSONString = '{"Results":{},"ErrorMessage":"","Success":1}';
        return (
            Content => \$JSONString,
            Status  => '200 OK',
        );
    }
}
1;^,
    Identifier => 'News',    # (optional) Code identifier to include in file␣
↪name
);
```

ProvideTestDatabase() 这个函数将为测试提供临时数据库 。请首先在 Kernel/Config.pm 中定义测试数据库设置，如:

```
$Self->{TestDatabase} = {
    DatabaseDSN  => 'DBI:mysql:database=otrs_test;host=127.0.0.1;',
    DatabaseUser => 'otrs_test',
    DatabasePw   => 'otrs_test',
};
```

方法调用将在测试期间覆盖全局数据库配置，即临时数据库将接收通过系统 `DBObject` 发送的所有调用 。

当 `Helper` 对象被销毁时，所有数据库内容都将自动删除 。

如果未配置测试数据库，此方法将返回 `undef` 。如果它已配置，但无法读取或执行提供的 XML，则此方法将调用 `die()` 以中断测试并显示错误 。

```
$Helper->ProvideTestDatabase(
    DatabaseXMLString => $XML,       # (optional) OTRS database XML schema␣
↪to execute
                                     # or
    DatabaseXMLFiles => [            # (optional) List of XML files to load␣
↪and execute
        '/opt/otrs/scripts/database/otrs-schema.xml',
        '/opt/otrs/scripts/database/otrs-initial_insert.xml',
    ],
);
```

CHAPTER 7

其它资源

otrs.com 含有源代码、文档和新闻的 OTRS 网站为 www.otrs.com。在这里，您还可以找到来自 OTRS 创始人–OTRS 集团关于专业服务和 OTRS 管理员培训研讨会的信息。

在线 **API** 库 The OTRS developer API documentation is available for Perl and JavaScript.

开发者邮件列表 The OTRS developer mailing list is available at https://lists.otrs.org/.