

微服务插件框架

Micro Service Plugin Framework

用户手册

Version 0.1.0

Dahai Cao

2021.8.22

目录

1.引言.....	3
1.1 编写目的.....	3
1.2 研发背景.....	3
1.3 产品定义.....	3
2.Mspf 介绍.....	6
3.开发环境搭建.....	7
4.下载代码.....	7
5.项目目录结构.....	8
6.软件功能.....	10
6.1 主界面.....	10
6.2 上传插件.....	12
6.3 创建插件.....	13
6.4 删除插件.....	19
6.5 编辑插件信息.....	20
6.6 编辑插件源程序.....	20
6.7 下载插件源程序.....	31
7.实战开发指南.....	32
7.1 前端界面设计.....	32
7.2 后端接口设计.....	37
8.部署生产环境.....	38
8.1 安装 Python.....	41
8.2 安装 Pip.....	41
8.3 安装 Django.....	42
8.4 安装 uWSGI.....	43
8.5 安装 Nginx.....	44
8.6 配置 uwsgi.ini 文件.....	44
8.7 配置 Nginx 文件.....	46
8.8 操作命令.....	49
9.Mspf 的数据库设计.....	50
9.FAQ.....	54
9.1 出现无法运行 <code>python manage.py dbshell</code> 的处理方法.....	54

1. 引言

1.1 编写目的

本文档是微服务插件框架（英文名称 **Micro service plugin framework**，以下简称：**Mspf**）的用户手册。该手册用于讲解我们为什么要研发这样一个框架，以及讲解 **Mspf** 的产品定义和产品功能，用户如何使用它开发和发布自己的插件。本文档的读者群是具有开发能力的软件开发人员，开发人员具备前端开发能力，如熟悉 **HTML**，**JavaScript**，**CSS**，**DOM**，**Ajax** 等，以及 **VUE**，**Jquery**，前端插件技术等，后端开发能力如 **Python** 语言，**RESTful** 技术，**Django** 技术，数据库技术等。

1.2 研发背景

当前云计算、大数据以及 **AI** 的发展，是软件信息化革命后的又一次软件智能化升级，是一次深度革命。未来，大数据应用以及 **AI** 应用不再是完整的信息化系统，而是多个相对独立的微服务应用，将智能化数据处理结果展示给用户。因此，微服务架构将是软件领域的主流技术架构的判断。

Python 语言已经是大数据应用以及 **AI** 的主流开发语言，当前还没有一个成熟的支持 **Python** 语言的微服务插件的平台或框架。大量 **Python** 开发的应用均是独立的应用，移植性不好。

我们开发微服务框架，基于 **Django** 框架开发，每个微服务均是 **Django** 的一个 **APP**，以 **RESTful** 形式接口提供 **JSON** 数据访问服务。在可预见的未来十年，**AI** 和大数据微服务将遍布于全球互联网。而微服务框架将是支撑微服务运行的基础设施和平台。

1.3 产品定义

微服务插件框架，这个概念分为两个层面，首先是微服务，其次是插件框架。

微服务是从软件组织架构上来构建软件，插件是从软件技术框架上来构建软件，是从不同层面的构建软件。

微服务，又叫微服务架构，是一种软件的架构方式和组织方法，其中软件由通过明确定义的 **API** 进行通信的小型独立服务组成，这些独立服务是按功能最小化原则进行划分的、具有自治能力的、具有互相通信能力的软件模块。

微服务的特点

解耦：系统内的软件各模块基于功能最小化原则被解耦。解耦后的系统将可轻易地通过组合被重构、修改和扩展。

组合编排：系统解耦成微服务后，可以根据业务系统功能需求，实现灵活的组合和组装，快速搭建成符合特定需求的应用系统。

组件化：微服务可以被看成相互独立的组件，可以通过独立的技术栈实现，而组件间通信基于 **REST API** 访问和通过 **JSON** 作为数据交换格式（这是当前事实上的软件访问标准和数据交换标准），组件作为整体的替换和内部功能的升级对其他组件是透明的。

业务能力：微服务因功能最小化原则而构建，它们可专注于某种单一的功能，业务和功能边界清晰。

自治：开发者和团队独立地、并行地工作，提高开发速度。

持续交付：允许持续发布软件新版本，通过系统化的自动手段来创建、测试和批准新版本。

去中心化管理：微服务的关注于使用正确的工具来完成正确的工作。这也就是说，没有标准化的方式或者技术模式。开发者们有权选择最好的工具来解决问题。

敏捷性：微服务支持小型独立团队敏捷开发自己专注的服务，以自己熟悉的环境更迅速和快速的工作，以缩短开发周期。任何新功能都可以被快速开发或丢弃。

微服务的优势

独立开发：适合于小团队的快速独立开发出内聚性更高的独有的功能。

独立部署：基于它们所提供的服务，它们可以被独立地部署到应用中。

错误隔离：即便其中某个服务发生了故障，整个系统还可以继续工作。同时由于便于迅速定位错误原因，降低修复故障成本。

混合技术栈：可以使用不同技术栈为同一个应用构建不同的服务。队可以自

由选择最佳工具来解决他们的具体问题。

按粒度扩展：可以根据需求扩展某一个组件，不需要将所有组件全部扩展。

重用性增强：将软件划分为小型且明确定义的微服务，可用于多种组合以满足不同的功能需求，开发人员无需从头开始编写代码。

插件，英文，**Plug-in**，又称 **addin**、**add-in**、**addon** 或 **add-on**，又译外挂，是一种遵循一定规范的应用程序接口编写出来的一个或一组程序或程序包。其不能脱离指定的平台单独运行，只能运行在程序规定的系统平台下（可能同时支持多个平台）。因为插件需要调用原纯净系统提供的函数库或者数据。很多软件都有插件，插件有无数种。例如在 **IE** 中，安装相关的插件后，**WEB** 浏览器能够直接调用插件程序，用于处理特定类型的文件。

插件的基本特点：

热插拔：支持动态的安装、启动、停止、更新和卸载，而整个系统无需重启。

适配性：动态的注册、获取和监听服务，使得系统能够在微服务插件环境调整自己的功能。

透明：提供了接口来监控插件内部状态，能够通过命令行进行调试。

版本化隔离：插件可以版本化，多版本能够共存而不会影响系统功能，解决了 **JAR hell** 的问题。

快速：插件类的加载速度快。

懒加载：本框架采用了很多懒加载机制。比如服务可以被注册，但是直到被使用时才创建，用即加载。

插件设计的优势：

结构清晰、易于理解：由于各个插件之间是相互独立的，所以结构非常清晰也更容易理解。良好地体现了微服务的解耦思想。

易修改、可维护性强：由于插件与宿主程序之间具有热插拔特点，软件结构很灵活，容易修改，方便软件的升级和维护。

可移植性强、重用力度大：插件本身也是由一系列小的功能结构组成，通过接口向外部提供自己的服务，可复用程度高，易于移植。

结构容易调整：系统功能的增加或减少，只需相应的增删插件，而不影响整个体系结构，能方便的实现结构调整，具有超强的组合能力。

插件之间的耦合度较低：插件通过与宿主程序通信来实现插件与插件，插件与宿主程序间的通信，插件之间的耦合度更低。

可以在软件开发生命周期中修改应用程序：插件的结构支持在软件的开发过程中随时修改插件，也可在应用程序发行之后，通过补丁包的形式增删插件，通过这种形式达到修改应用程序的目的。

插件化思想已经在多个框架或者设计模式中得以实现，应用不同语言的插件框架也很多。基于 Java 语言开发的开放服务网关协议（Open Services Gateway initiative，简称 OSGi）的插件框架，如 Apache Felix，Equinox，均是优秀的插件框架。近年利用 Springboot 的注入原理开发的国产插件框架，也在成长。而基于 Python 语言的插件框架有 PluginBase。

微服务的软件架构思想与插件的技术框架思想的结合，是未来软件开发的一个重要方向，特别是智能手机的普及和移动互联网的兴起，促使软件发展趋向为功能单一化，操作简单化，业务边界明确。Python 语言近年的迅速普及，让软件开发的简单化成为新的趋势。一个功能只要从原来 Java 开发要数十行代码，到 Python 的数行代码。更促使软件开发团队小型化，甚至大量的个人软件开发极客，他们能够开发出大量具有专业性质的小软件服务，服务特定的行业和客户。这一趋势也孕育了微服务插件框架的市场要求。

2.Mspf 介绍

Mspf 是一个开源的能够支持面向 Python 语言的微服务插件的插件框架。Mspf 基于 Django 开发。每个微服务插件都是一个 Django 的 App。

这个插件框架具有以下特点：

（1）可直接热插拔到框架：用户可以上传和下载 Django 的 App 的 zip 文件到框架上，框架能够自动注册和加载；插件可以直接从框架删除，并自动注销；在框架上可创建一个 Django 的 App 插件，并注册到框架；

（2）框架提供 App 中的文件代码快速编辑功能，以支持快速开发 Python 微服务的前后端代码，前端代码可以是 VUE，Django 的 template，也可以是 HTML 和 DOM 等，后端是 Python 代码，提供 REST API 接口注册。

当前 Mspf 是一个微服务插件框架的初级版本，对于微服务的特点以及插件框架的特点都还没有充分支持，随着深入的产品开发，上述特征将逐渐支持。

我们欢迎任何感兴趣的 Python 工程加入团队，为开源的框架做出贡献。

3.开发环境搭建

Mspf 支持跨平台运行，支持 Windows 和 Linux，需要在平台上安装 Python 环境。

- Python 3.9

安装 Python 3.9，在 Windows 上安装和 Linux 上安装，网上有很多培训教材，此处不再赘述。

- Django-3.2.4:

命令：pip install django

- Watchdog 2.1.3:

命令：pip install watchdog

- Configparser-5.0.2:

命令：pip install configparser

- Python-socketio:

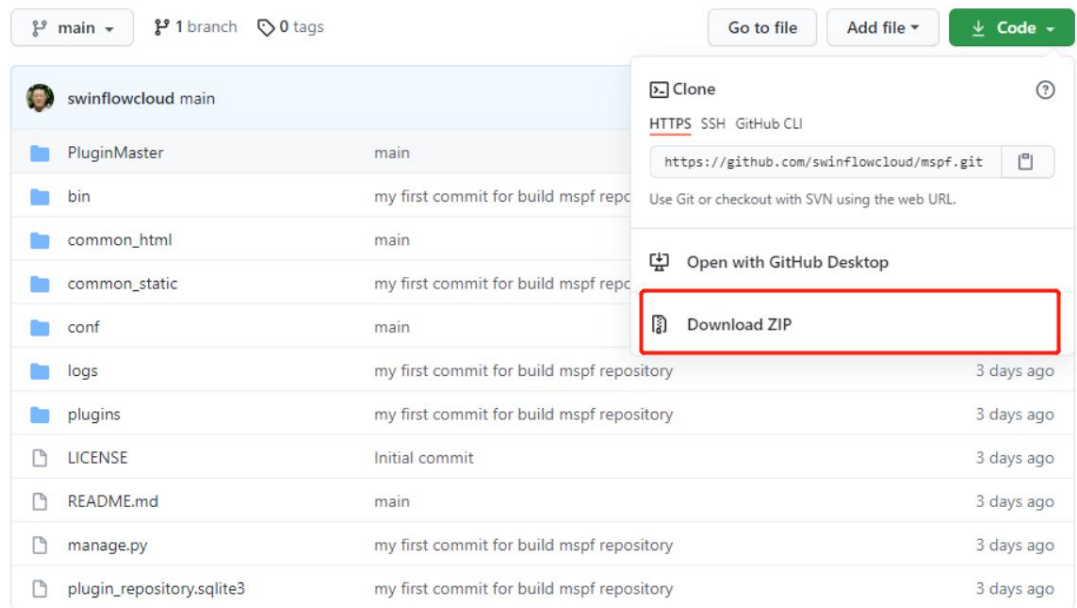
命令：pip install python-socketio

我们建立了一个长活 Demo 环境，可以随时访问：<http://47.115.33.219/>

4.下载代码

Mspf 的完整代码在 <https://github.com/swinflowcloud/mspf>，如下图所示，下

载 Download ZIP，就可以把完整代码下载下来。



在 bin 目录下面，运行 `./startup.bat`（Windows）或者 `./startup.sh`（Linux），就可以运行来整个框架。

5.项目目录结构

打开项目，通过 `tree`，来展示目录，每个目录的解释如下图所示：

命令提示符

```
D:\>
D:\>cd D:\Plugin-codes\PaaSPluginFwk
```

```
D:\Plugin-codes\PaaSPluginFwk>tree
```

卷 数据 的文件夹 PATH 列表
卷序列号为 2EA8-AAF9

```
D:
├── bin
├── common_html
├── docs
├── downloads
├── common_static
│   ├── base-plugins
│   │   ├── ace-src-noconflict
│   │   │   └── snippets
│   │   ├── bootstrap-5.0.2-dist
│   │   │   ├── css
│   │   │   └── js
│   │   ├── bootstrap-icons-1.5.0
│   │   │   └── fonts
│   │   ├── font-awesome-4.7.0
│   │   │   ├── css
│   │   │   ├── fonts
│   │   │   ├── less
│   │   │   └── scss
│   │   ├── jquery-ui-1.12.1
│   │   │   ├── external
│   │   │   │   └── jquery
│   │   │   └── images
│   │   ├── jstree-3.3.11-dist
│   │   │   └── themes
│   │   │       ├── default
│   │   │       └── default-dark
│   ├── css
│   ├── img
│   └── js
│       ├── common
│       └── plugins
├── conf
├── logs
├── PluginMaster
│   ├── migrations
│   │   └── pycache__
│   ├── static
│   │   ├── css
│   │   ├── img
│   │   └── templates
│   └── pycache__
├── plugins
│   ├── SP_00000000000000A4M0
│   │   ├── logs
│   │   ├── migrations
│   │   │   └── pycache__
│   │   ├── static
│   │   │   ├── css
│   │   │   ├── img
│   │   │   └── jsplugins
│   │   ├── templates
│   │   └── pycache__
│   └── SP_00000000000000A4NZ
│       ├── logs
│       ├── migrations
│       │   └── pycache__
│       ├── static
│       │   ├── css
│       │   ├── img
│       │   └── jsplugins
│       ├── templates
│       └── pycache__
```

公共的html代码，如html等

static目录是静态文件目录，如基础的组件如jquery，font-awesome等。该目录还包括自定义css，js，img文件目录。

PluginMaster是保存框架管理器的目录，这个目录保存着所有的管理文件，前端的和后端的脚本和程序。

plugins目录是保存所有的插件的目录，每个插件有一个目录，每个plugin有一个PluginID，并且以此PluginID来作为插件的根目录。

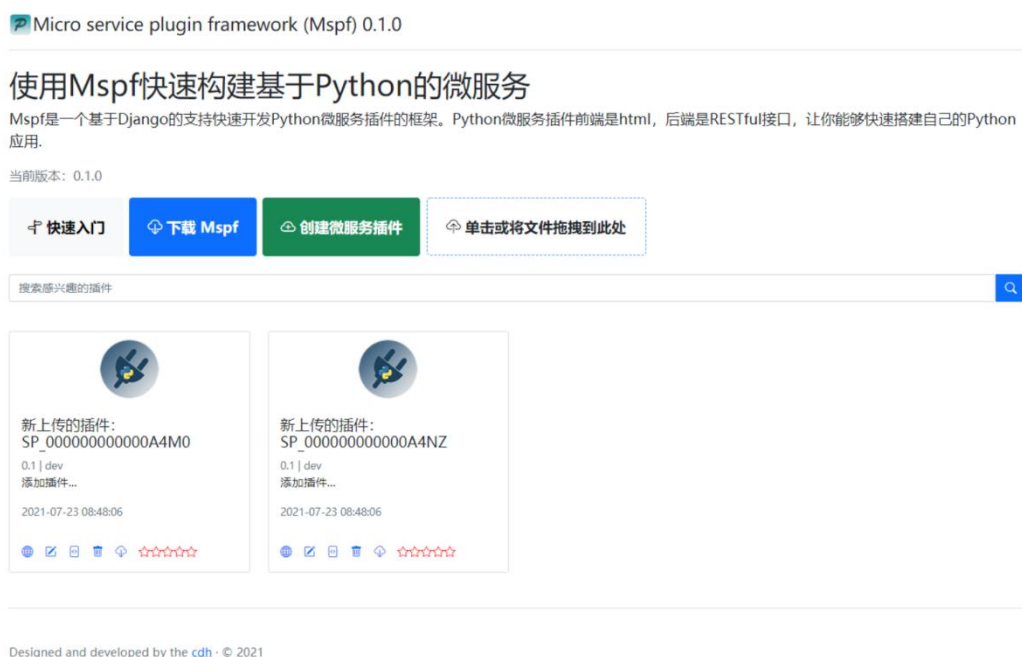
```
D:\Plugin-codes\PaaSPluginFwk>
```

项目可以通过 VSCode 打开。

6. 软件功能

6.1 主界面

Mspf 框架启动起来后，在浏览器的地址栏中输入 `http://localhost:8899/`



如上图所示，界面上有几个按钮：

快速入门按钮： 点击按钮可以下载本文件。

下载 Mspf 按钮： 点击按钮下载框架所有的代码。

创建微服务插件按钮： 点击按钮可以创建插件，参见 6.3 节。


单击或将文件拖拽到此处按钮： 将插件压缩文件拖动带此区域，直接上传并解压和注册插件。注册完成的插件，就可以直接访问了。

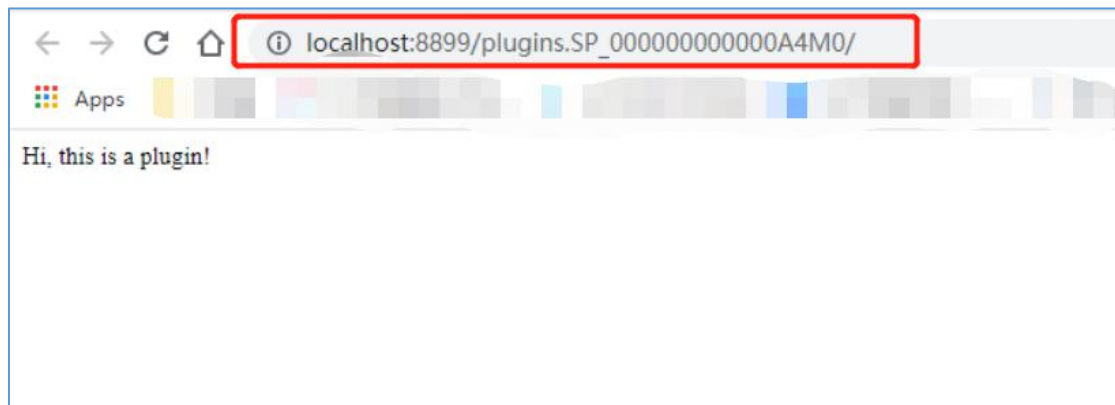
搜索框： 输入一个字符串，可以模糊匹配出符合自己需要的插件列表。

下面是插件列表，这个插件列表默认列出来所有的已经注册的插件。插件基本信息卡片如下图所示：





如上图所示，卡片最上方是图标，未来能够显示服务插件的前端页面。中间是插件的名称、版本和开发商。下面是插件的描述以及最后更新时间。

下面的一排小图标是操作按钮，第一个是预览图标 ，点击此图标是可以预览插件的主页。如下图所示：




上面的插件的访问地址是 `http://ip:port/plugins.PluginID/`，访问地址的结构，将在 6.2 节说明。

点击  图标，可以编辑插件的基本信息。详见 6.5 节。

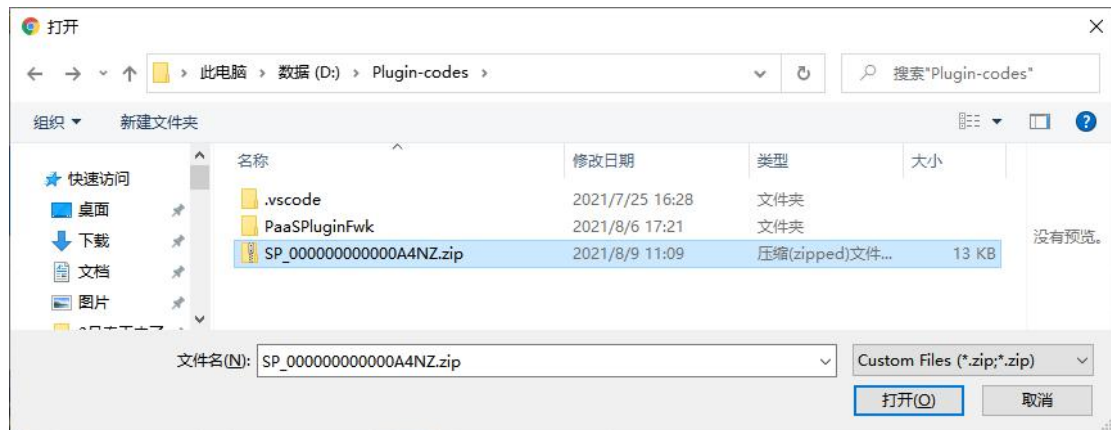
点击  图标，可以编辑插件的源代码。详见 6.6 节。

点击  图标，可以删除插件，详见 6.4 节。

点击  图标，可以下载插件的源代码，详见 6.7 节。

6.2 上传插件

点击上传单机或将文件拖拽到此处按钮，打开文件选择器。如下图所示，选择器中仅显示 zip 文件。



下面讲解下插件的文件目录结构：

```
D:\Plugin-codes\PaaSPluginFwk\plugins\SP_000000000000A4NZ>tree
卷 数据 的文件夹 PATH 列表
卷序列号为 2EA8-AAF9
D:.
|-- logs
|-- migrations
|   |-- __pycache__
|-- static
|   |-- css
|   |-- img
|   |-- jsplugins
|-- templates
|   |-- __pycache__
```

上图中，static 目录中，css 保存 CSS 文件，img 目录主要保存各种图片文件，jsplugins 目录保存 JavaScript 的脚本文件。templates 目录保存 HTML 页面，也保存 Django 的 template 文件，或者 VUE 文件。

如果要能成功上传一个插件文件包，文件包中需要包含如下图的文件：

-codes > PaaSPluginFwk > plugins > SP_000000000000A4M0				
名称	修改日期	类型	大小	
__pycache__	2021/8/6 13:04	文件夹		
logs	2021/7/23 8:48	文件夹		
migrations	2021/8/6 13:04	文件夹		
static	2021/8/6 13:04	文件夹		
templates	2021/7/23 8:48	文件夹		
__init__.py	2021/7/23 8:48	Python File	1 KB	
admin.py	2021/7/23 8:48	Python File	1 KB	
apps.py	2021/7/23 8:48	Python File	1 KB	
models.py	2021/7/23 8:48	Python File	1 KB	
plugin.conf	2021/7/23 8:48	CONF 文件	0 KB	
tests.py	2021/7/23 8:48	Python File	1 KB	
urls.py	2021/7/23 8:48	Python File	1 KB	
views.py	2021/8/5 8:18	Python File	1 KB	

__pycache__ 目录是 Django 的生成的，这里不深入讲解。其余的文件均是 Django 生成的 APP 的文件。我们将在 6.3 节，详细介绍插件中所包含文件。

在上传的插件的 apps.py 文件中，需要特别指出的是 apps.py 文件的 name 属性。一般地通过 Django 的 startapp 命令生成的 App 的 apps.py 文件中，都是直接将 app 的名字放在 class 名和 name 属性。举例来说，一个生成的 App 的名字叫 mytest，那么 apps.py 文件中的格式是：

```
from django.apps import AppConfig
# Secondly run:apps.py
# This file is used to configure current App's conf.
class MytestConfig(AppConfig):
    default_auto_field = 'django.db.models.BigAutoField'
    name = 'Mytest'
```

如果要把 mytest 这个 App 作为一个插件上传到 Mspf，那么需要修改 name 属性为：

```
name = 'plugins.Mytest'
```

然后将 App 打包后成 ZIP 文件后，就可以上传到 Mspf 框架。访问插件 App 需要在浏览器中输入：<http://ip:port/plugins.Mytest/...>

6.3 创建插件

创建插件功能是，Mspf 框架提供的在线创建一个插件的功能。点击创建微服务插件，将打开如下窗口：

创建新插件

插件名称
插件名称

开发者
开发者

版本
v0.0.1

版权所有
版权所有

描述
描述

关键词
关键词以空格分离

商标URL
商标URL, http://...

默认选项
默认选项或参数

价格 (元)
0.00

状态
未禁

取消 添加

如上图所示，输入插件名称等信息，尽管每个输入框都可以为空，但是尽量每个输入框都填写上信息。状态是未禁和已禁两种状态，这个目前初级版本未支持的。点击添加按钮，框架将创建一个插件，并在插件列表中展示出来。如下图所示，我们创建一个 **mytest** 插件：

创建新插件

插件名称

mytest

版本

0.0.12

描述

插件描述

关键词

插件

商标URL

商标URL, http://...

默认选项

默认选项或参数

价格 (元)

0.00

状态

未禁

开发者

dev

版权所有

dev

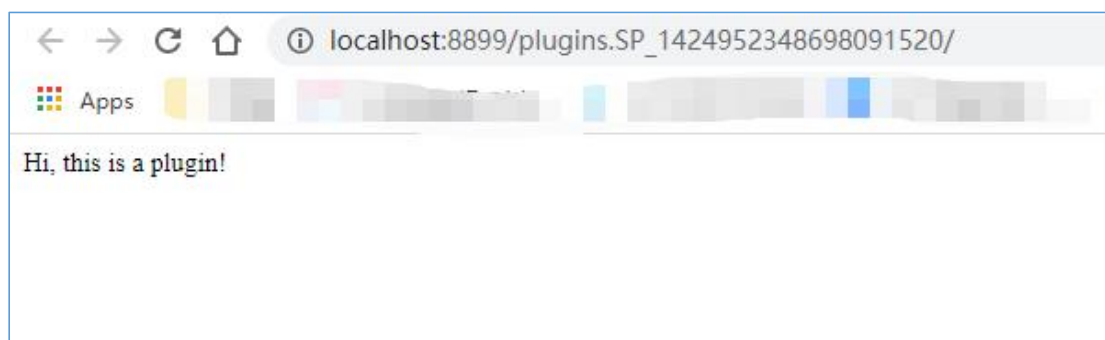
取消

添加

点击添加按钮后，如下图所示：

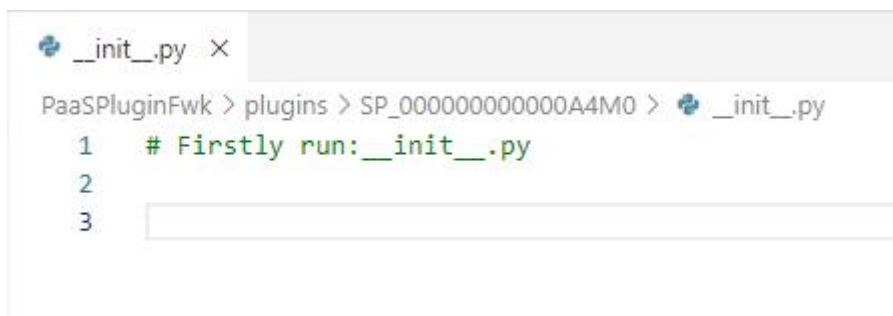


插件已经创建好，并且能够显示在插件列表中。点击预览，可以看到如下插件的页面：



这意味着我们的插件已经创建好了。新创建好的插件中包括__init__.py，admin.py，apps.py，models.py，plugin.conf，tests.py，urls.py，views.py 等文件，除 plugin.conf 之外。

一个新创建的插件中的文件如下，__init__.py 内容：



这个文件是 App 运行起来后，第一个执行的文件。

admin.py 文件的内容：

```
admin.py X
PaaSPluginFwk > plugins > SP_000000000000A4M0 > admin.py
1  from django.contrib import admin
2
3  # Fourthly run:admin.py
4  # This file is used to configure the admin site.
5  # Register your models here.
6
7  |
```

admin.py 文件是 App 启动起来后，第四个执行的文件。

apps.py 文件的内容：

```
apps.py X
PaaSPluginFwk > plugins > SP_000000000000A4M0 > apps.py > ...
1  from django.apps import AppConfig
2
3  # Secondly run:apps.py
4  # This file is used to configure current App's conf.
5
6  class SP_000000000000A4M0Config(AppConfig):
7      default_auto_field = 'django.db.models.BigAutoField'
8      name = 'plugins.SP_000000000000A4M0'
9
10 |
```

apps.py 文件是 App 执行起来后，第二个执行的文件。

models.py 文件的内容是：

```
from django.db import models
# Thirdly run:models.py
# Create your models here.
```

该文件是标准的 Django 的 App 生成的 models.py。tests.py 也是标准的 Django 的 App 生成文件。

urls.py 文件中，有两个 URL

```
from django.urls import path
from . import views
```

```

# This file is used to defined RESTFul APIs:
# path('hello/', views.hello),
that is http://localhost:port/hello,
# direct to views.hello method,
views.hello defined in views.py。
# Fifthly run:urls.py

```

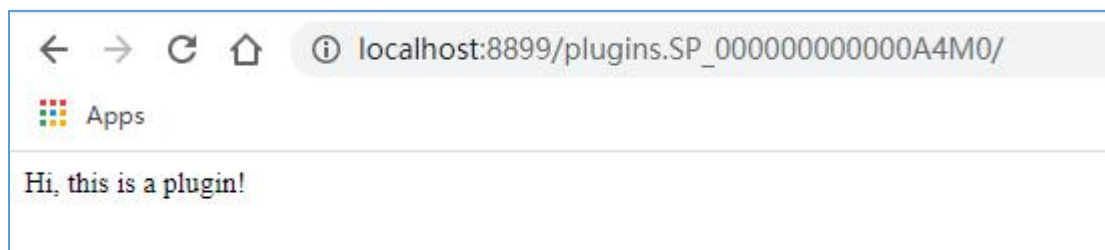
```

urlpatterns = [
    path('', views.pluginIndex),
    path('hello/', views.hello),
]

```

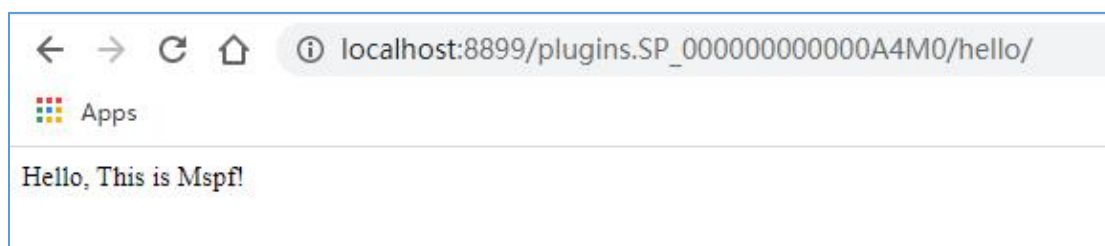
这个文件中，包含两个 URL 路径。第一个是默认路径，也就是说，在浏览器中输入 `http://ip:port/plugins.PluginID/` 就可以访问到 `views.pluginIndex` 方法，输入 `http://ip:port/plugins.PluginID/hello/`，就可以访问到 `views.hello` 方法。例如，在浏览器中输入

`http://localhost:8899/plugins.SP_000000000000A4M0/`，会出现如下图所示：



同样，输入

`http://localhost:8899/plugins.SP_000000000000A4M0/hello`，会出现如下图所示：



上述两个链接访问的 URL，都是 `views.py` 的方法，或者叫函数。具体如下：

```

from django.shortcuts import render
from django.http import HttpResponse, JsonResponse
from django.views.decorators.csrf import csrf_exempt
# When getting requests from browsers using ajax :views.py

```

```
# this file is similar to controllers in srpingmvc or sprin
gboot
# Create your views here.
def pluginIndex(request):
    return HttpResponse('Hi, this is a plugin!')
```

```
def hello(request):
    return HttpResponse('Hello, This is Mspf!')
```

这个文件中，第一个方法就是在 `urls.py` 中定义的 `path('', views.pluginIndex)`，第二个方法就是 `path('hello/', views.hello)`。

用户可以在 `urls.py` 增加新的访问 URL 定义，并相应的在 `views.py` 中添加方法即可。


如在 `urls.py` 中添加，`path('mytest/', views.test)`，在 `views.py` 中添加：

```
def test(request):
    return HttpResponse('Hello, This is mytest!')
```

可以在浏览器中试一下，
`http://localhost:8899/plugins.SP_000000000000A4M0/test/`

如果用户熟悉 Django，将能够基于这个框架开发出更复杂的应用。

6.4 删除插件


插件是可以被删除的。点击插件卡片下面的  图标，将会弹出对话框，以确认是否要删除插件。点击“是”，框架将删除插件。列表中也删除掉插件掉卡片信息。

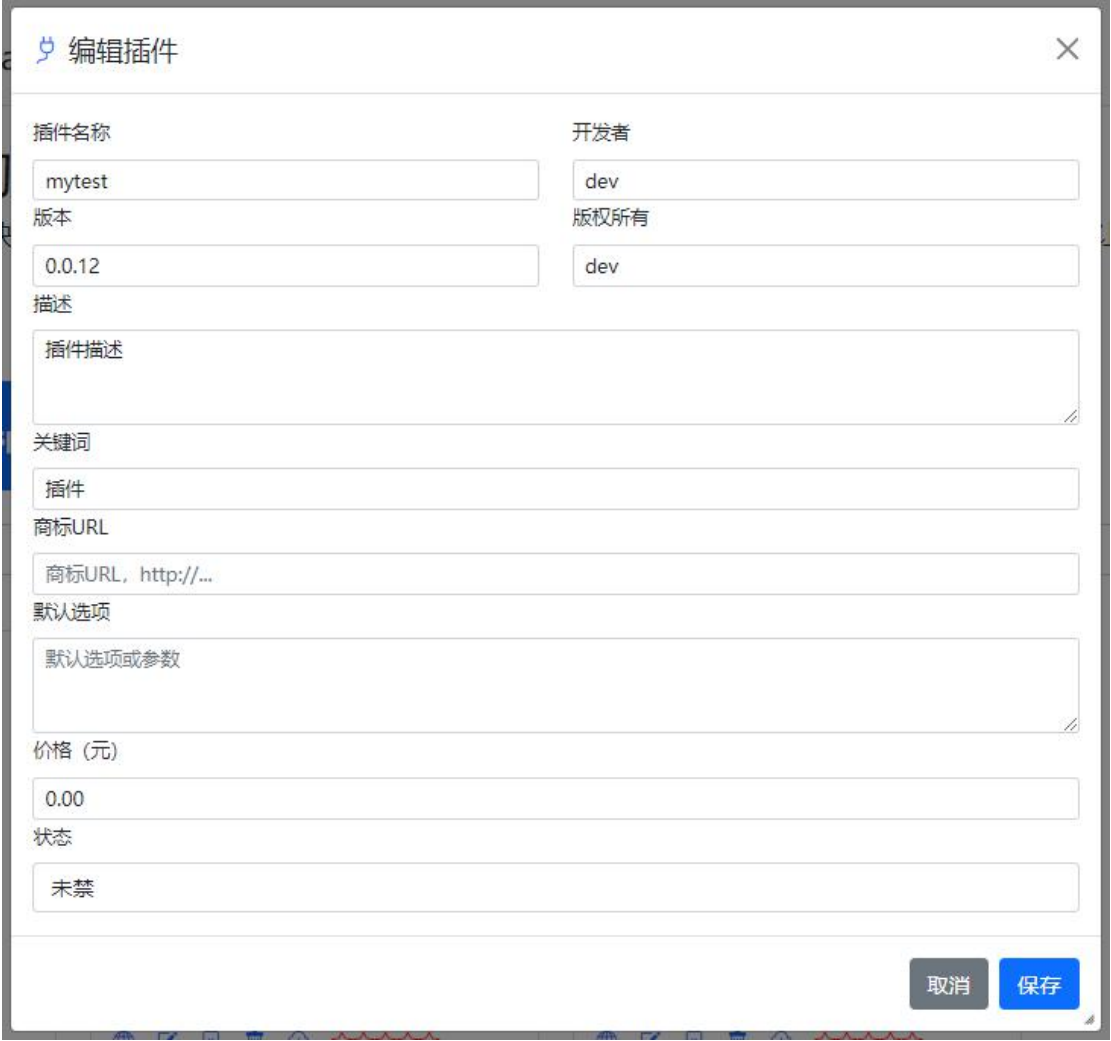


注：在初级版本中，一旦打开了插件代码编辑器，删除插件是无法控制代码

编辑框的关闭的，也就是说，删除了插件，代码编辑框应该伴随关闭。但是初级版本和没有实现。

6.5 编辑插件信息

创建插件时候输入的信息的编辑是点击插件信息卡片底部的 图标，点击后打开信息编辑对话框。如下图所示：




The image shows a dialog box titled "编辑插件" (Edit Plugin) with a close button (X) in the top right corner. The dialog contains several input fields for plugin information:

- 插件名称** (Plugin Name): mytest
- 开发者** (Developer): dev
- 版本** (Version): 0.0.12
- 版权所有** (Copyright): dev
- 描述** (Description): 插件描述
- 关键词** (Keywords): 插件
- 商标URL** (Trademark URL): 商标URL, http://...
- 默认选项** (Default Options): 默认选项或参数
- 价格 (元)** (Price): 0.00
- 状态** (Status): 未禁

At the bottom right, there are two buttons: "取消" (Cancel) and "保存" (Save).

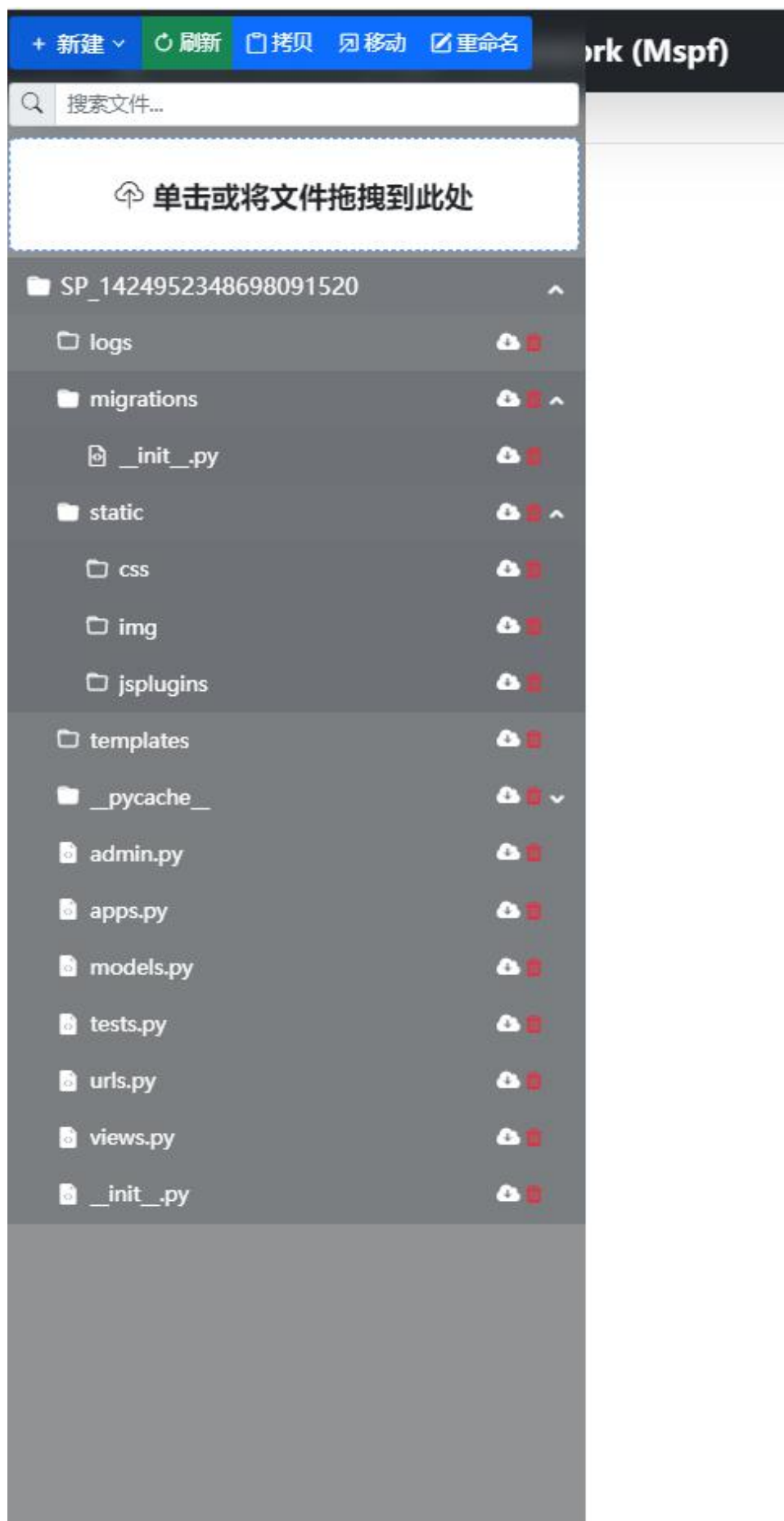
编辑完插件信息后，点击保存，插件信息将自动保存并刷新。

6.6 编辑插件源程序

插件中的文件以及源程序可以通过点击 图标来编辑，点击该图标，将打开源代码编辑器。如下图所示：



点击左上角的红框中的图标，可以看到插件下所有的文件。如下所示：



上图可以看到，最顶端有工具栏。如下图所示：



工具栏实现可以文件管理，需要注意的是，在操作文件或目录之前，需要选中一个目录，选中的这个目录作为父目录。点击新建按钮，选择新建目录选项，可以打开新目录对话框：



如上图所示，新建目录一定在一个目录之下，或者根目录，或者是根目录下边的一个目录。我们现在创建的目录就是在一个根目录下的 `static` 目录下面。输入目录名 `dddd`，点击创建。将在 `SP_1424952348698091520\\static` 下面创建一个目录 `dddd`。如下图所示：



如上图所示，鼠标移动上去，将显示目录名和最后修改时间戳。

选中 **dddd** 目录，再点击新建按钮，选择新建文件选项，打开新文件对话框，输入一个文件名称（注意一定要带扩展名）如下图所示：



输入文件名，一个新的文件将在 **dddd** 目录下创建。如下图所示：

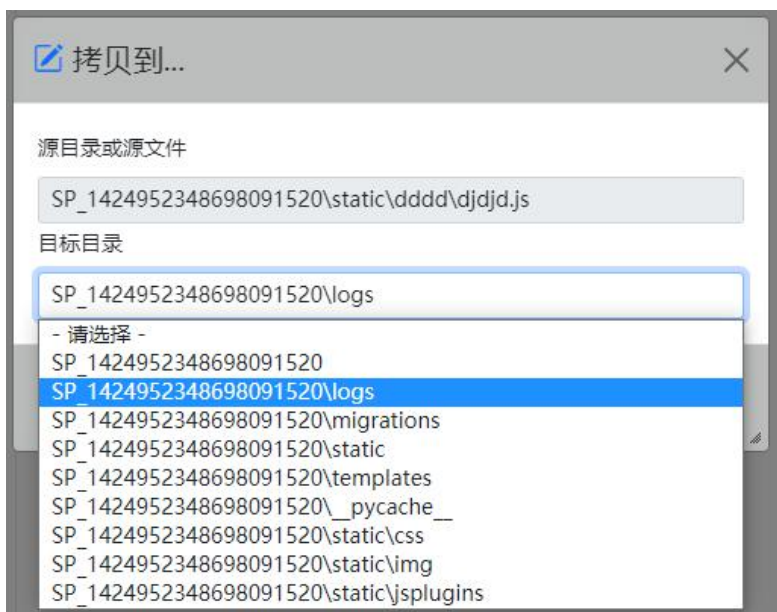


点击刷新按钮，是刷新所有插件文件目录。

拷贝功能可以拷贝文件，也可以拷贝目录，无论拷贝文件还是拷贝目录，都需要指定目标目录。点击拷贝按钮，打开拷贝对话框，如下图所示：



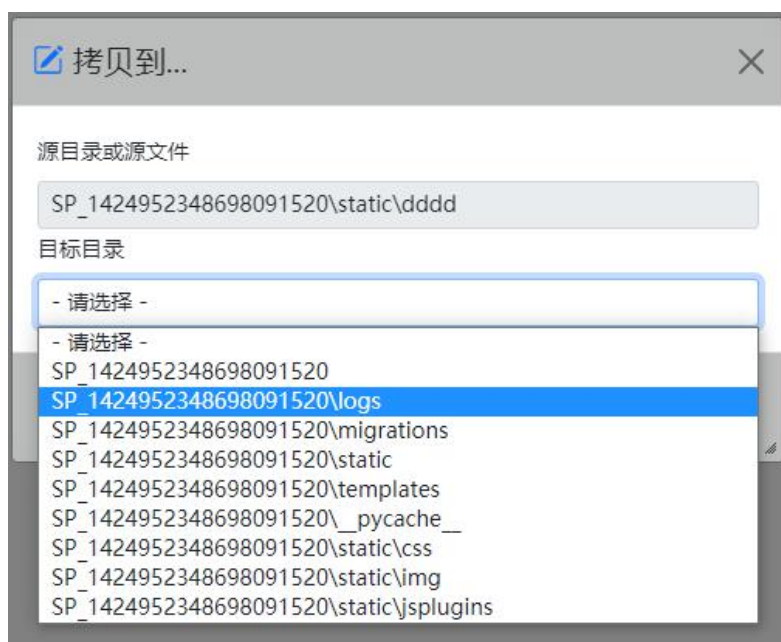
如上图所示，选择了拷贝文件 `SP_1424952348698091520\\static\\dddd\\djdd.js`，选择一个目标目录，如下图所示：



选择 SP_1424952348698091520\\logs 后，点击确定。文件将被拷贝到 SP_1424952348698091520\\logs 下面，如下图所示：



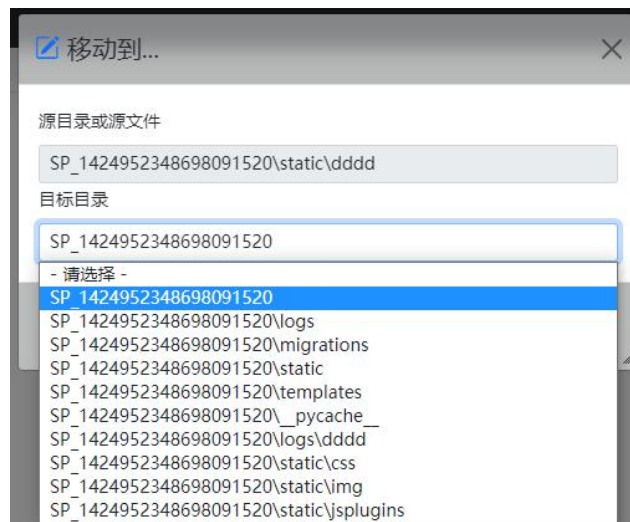
拷贝功能也可以支持拷贝目录，拷贝目录实际上拷贝目录树，即该目录下所有文件及目录都拷贝到新目录下。选择一个目录，点击拷贝按钮，选择一个目标目录，如下图所示：



点击确定按钮，源目录将被拷贝到 SP_1424952348698091520\\logs，如下图所示：



移动功能是移动文件或目录到新的目录下面，移动功能和拷贝功能类似，也是要选择目标目录。点击移动按钮，打开移动对话框，如下图所示：



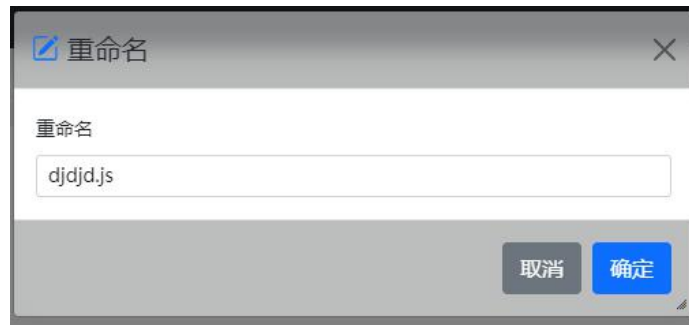
选择一个目标，目录点击确定。目录将被移动到指定目录下。如下图所示：



移动文件操作类似于拷贝文件，此处不再赘述。

重命名功能能够重命名文件或目录。以重命名文件为例，选择一个文件，点击重命名，打开重命名对话框，如下图所示：





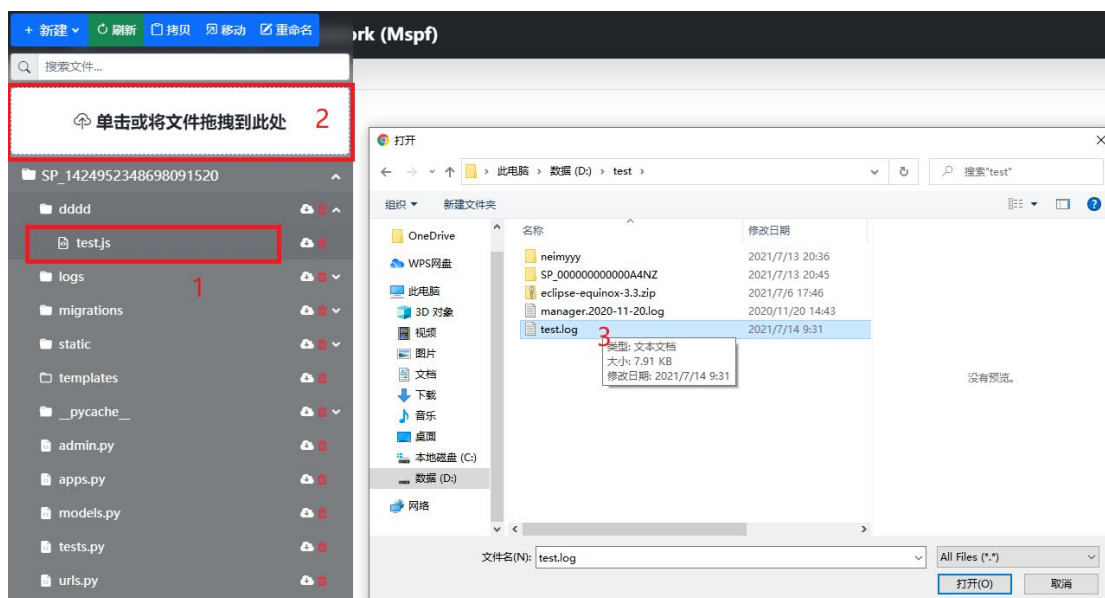
输入一个新的名称 `test.js`，点击确定，此文件的名称被更新，如下图所示：



重命名目录操作类似重命名文件名，此处不再赘述。

注：搜索文件的功能，在初级版本中未实现。待后续版本实现。


单击或将文件托拽到此处，是上传文件功能，该功能是支持将自定义的文件上传到插件空间中，上传仅支持文件上传，不支持目录上传。上传默认上传到根目录下，如果想上传到特定目录或者某文件所在目录下，选中一个上传的目标目录或者一个文件也可以，然后点击上传区域，打开一个文件选择对话框。如下图所示：




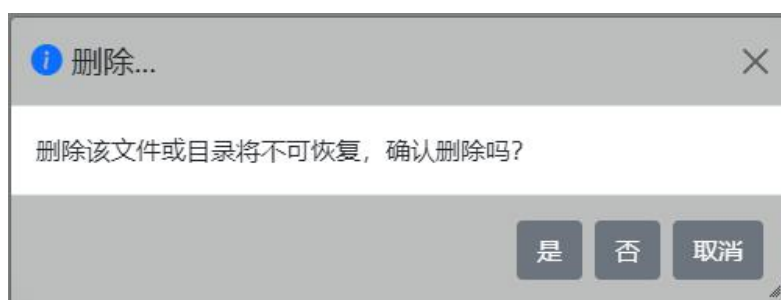
选择文件 **test.log**，点击打开。该文件将被上传到 **dddd** 目录下。如下图所示：



另一种方法是要上传的文件用鼠标拖动到此区域。也可以达到同样的效果。此处不再赘述。

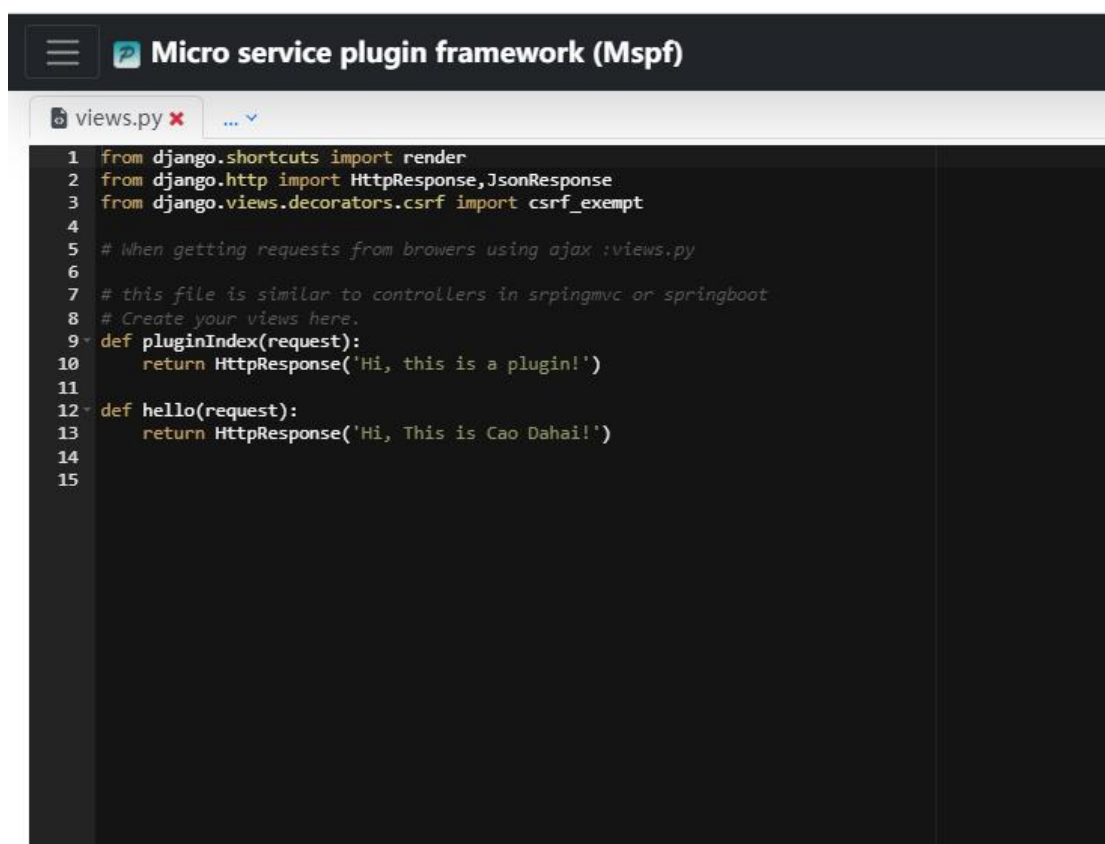
下载功能在每个目录或者文件旁边的小图标，点击这个图标，将让文件或者目录，以压缩包 ZIP 文件的形式，下载到本地。此功能很简单，不再赘述。

删除功能是点击每个目录或者文件旁边的小图标，将弹出来一个删除确认对话框，如下图所示：



点击“是”按钮，将删除该文件或目录。

点击任何文件，都可以打开一个编辑器进行编辑，如下图所示：



```
1 from django.shortcuts import render
2 from django.http import HttpResponse, JsonResponse
3 from django.views.decorators.csrf import csrf_exempt
4
5 # When getting requests from browsers using ajax :views.py
6
7 # this file is similar to controllers in srpingmvc or springboot
8 # Create your views here.
9 def pluginIndex(request):
10     return HttpResponse('Hi, this is a plugin!')
11
12 def hello(request):
13     return HttpResponse('Hi, This is Cao Dahai!')
14
15
```

编辑完文件自动保存，无需点击保存。

6.7 下载插件源程序

插件源程序可以打成 ZIP 包下载。点击下载图标，如下图所示：



插件源程序就能打包下载。此功能很简单，不再赘述。

7. 实战开发指南

用 Mspf 开发微服务插件，本质上就是开发 Django 的 App，但现在 Django 开发主要是本地开发，完成开发后，直接部署到服务器。而基于 Mspf 开发，是一种在线开发模式。Mspf 为插件创建插件空间，插件的所有文件均在插件空间中。Mspf 提供在线前端后端程序编辑器，前端编辑器设计界面和脚本，后端 Python 编辑器设计后端访问 TEST API 接口。开发完成即可完成部署和加载，是一种新的敏捷开发模式。

下面我们创建一个测试插件，并试着开发前端界面和后端接口。前端展示基于 Bootstrap 5.1.0 (<https://getbootstrap.com/>) 的 HTML 页面，后端开发一个基于 Python 的 REST API 接口，供前端页面访问。

7.1 前端界面设计

第一步：

我们创建一个插件。点击创建微服务插件。如下图所示：

创建新插件

插件名称

myplugin

开发者

dev

版本

0.1

版权所有

dev

描述

my plugin

关键词

plugin

商标URL

http://www.google.com/icon

默认选项

默认选项或参数

价格 (元)

0.00

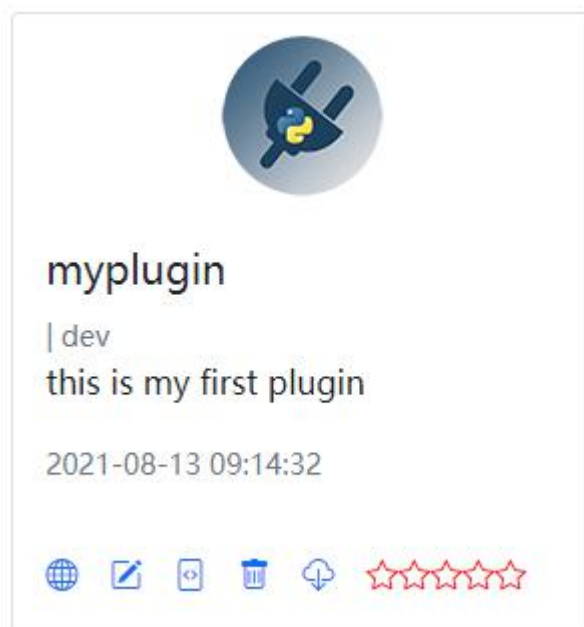
状态

未禁

取消

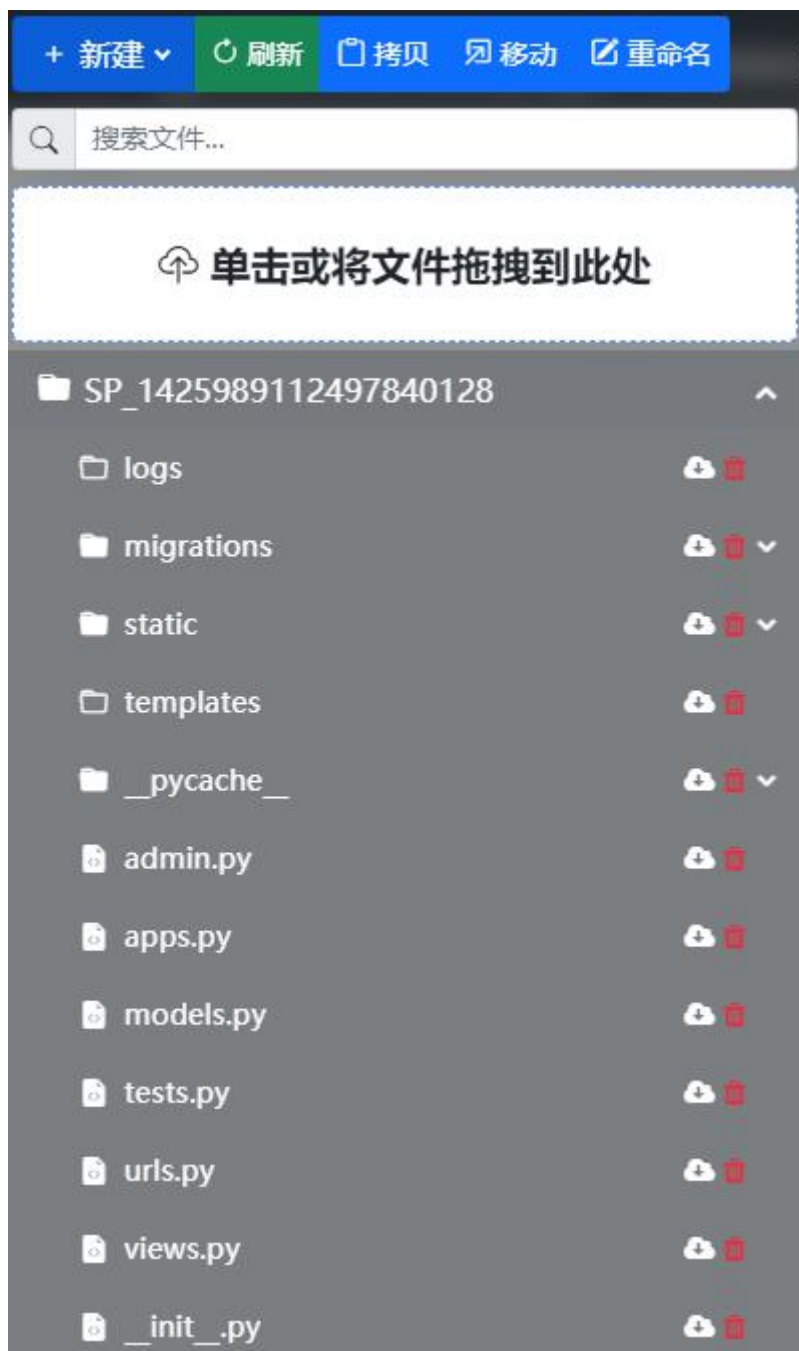
添加

点击添加按钮，新插件创建好。如下图所示：



第二步：

点击编辑代码图标，打开代码编辑界面。



第三步：

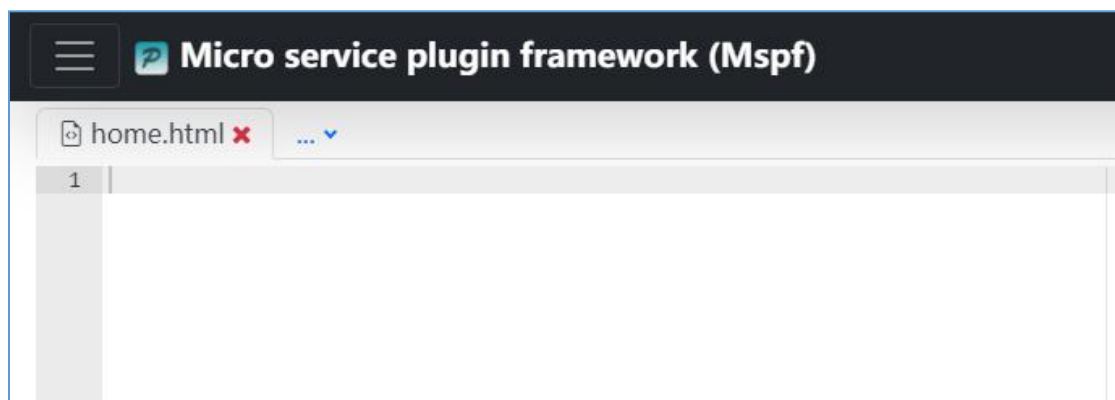
在 templates 目录下面创建一个 home.html 页面。如下图所示：



点击创建，创建一个新的空页面。

第四步：

用编辑器打开这个页面。



在该页面中，输入如下代码：

```
<!DOCTYPE html>
<html lang="zh-CN">
<head>
  <meta charset="utf-8">
  <meta http-equiv="Content-Type" content="text/html; charset=utf-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <meta http-equiv="X-UA-Compatible" content="IE=edge">
  <meta http-equiv="pragma" content="no-cache">
  <meta http-equiv="Cache-Control" content="no-store, no-cache, must-revalidate">
  <meta name="keywords" content="Mspf, Micro service, plugin framework" />
  <meta http-equiv="expires" content="0">
```

```

    <!-- Bootstrap CSS CDN -->
    <link rel="stylesheet" href="../static/base-plugins/bootstrap-5.1.0
-dist/css/bootstrap.min.css">
    <link rel="stylesheet" href="../static/base-plugins/font-awesome-4.
7.0/css/font-awesome.css">
    <link rel="stylesheet" href="../static/base-plugins/bootstrap-icons
-1.5.0/bootstrap-icons.css">

    <title>主界面</title>
</head>
<body>

    <button type="button" name="getdata" class="btn btn-primary bi bi-p
lay-fill" onclick="getdata()">获得数据</button>&nbsp;
    <div id="label"></div>

<!-- jQuery CDN - Slim version (=without AJAX) -->
<script src="../static/base-plugins/jquery-3.5.1.min.js"></script>
<!-- Popper.JS -->
<script src="../static/base-plugins/popper.min.js"></script>
<!-- Bootstrap JS -->
<script src="../static/base-plugins/bootstrap-5.1.0-dist/js/bootstrap.b
undle.js"></script>
<script>
function getdata() {
    $.ajax({
        url: "../getdata",
        data:{
            a: 1,
        },
        type: 'GET',
        dataType: 'JSON',
        complete:function(data) {
            data = data.responseJSON;
            var el = document.getElementById("label");
            el.innerHTML = data.msg;
        },
    });
}

</script>

</body>
</html>

```

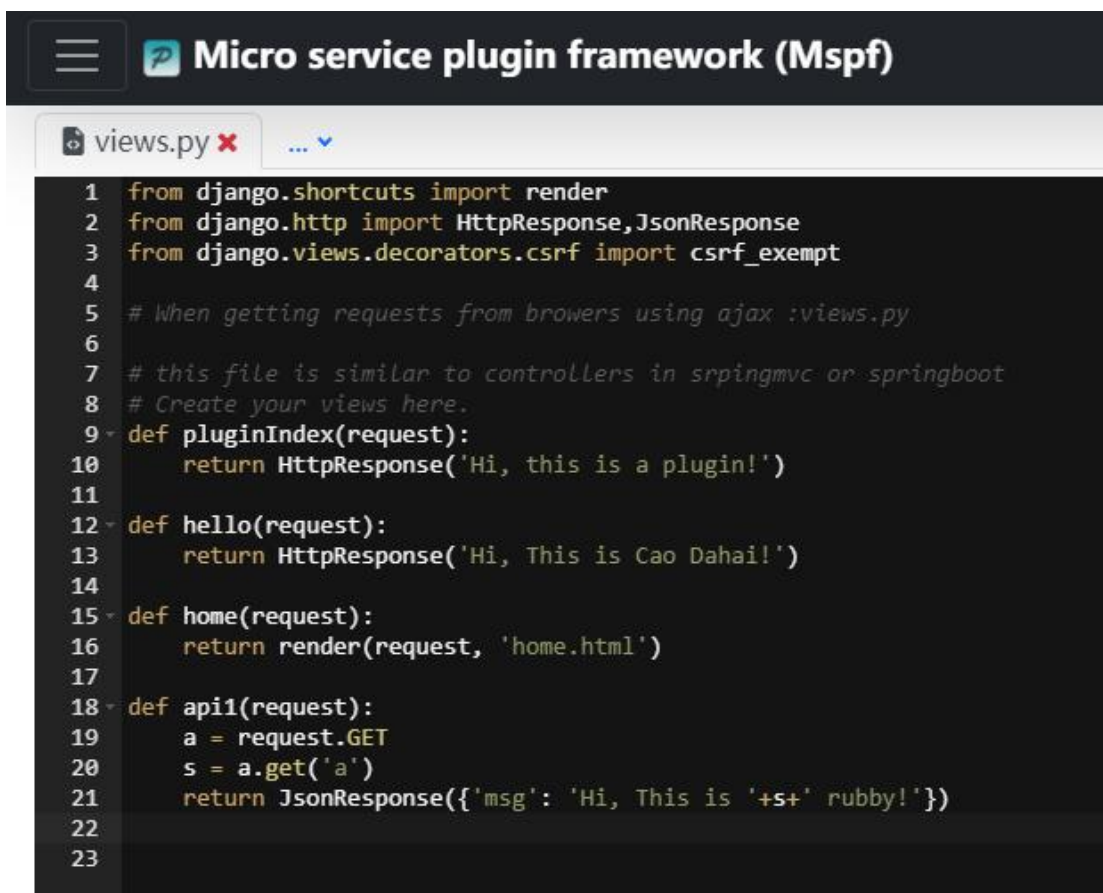
7.2 后端接口设计

第五步：

后端接口设计，打开 views.py。添加

```
def home(request):  
    return render(request, 'home.html')  
  
def api1(request):  
    a = request.GET  
    s = a.get('a')  
    return JsonResponse({'msg': 'Hi, This is '+s+' rubby!'})
```

如下图所示：



```
1 from django.shortcuts import render  
2 from django.http import HttpResponseRedirect, JsonResponse  
3 from django.views.decorators.csrf import csrf_exempt  
4  
5 # When getting requests from browsers using ajax :views.py  
6  
7 # this file is similar to controllers in srpingmvc or springboot  
8 # Create your views here.  
9 def pluginIndex(request):  
10     return HttpResponseRedirect('Hi, this is a plugin!')  
11  
12 def hello(request):  
13     return HttpResponseRedirect('Hi, This is Cao Dahail!')  
14  
15 def home(request):  
16     return render(request, 'home.html')  
17  
18 def api1(request):  
19     a = request.GET  
20     s = a.get('a')  
21     return JsonResponse({'msg': 'Hi, This is '+s+' rubby!'})  
22  
23
```

第六步：

修改 urls.py 文件，注册一个 REST API，让前端可以访问。添加如下代码到

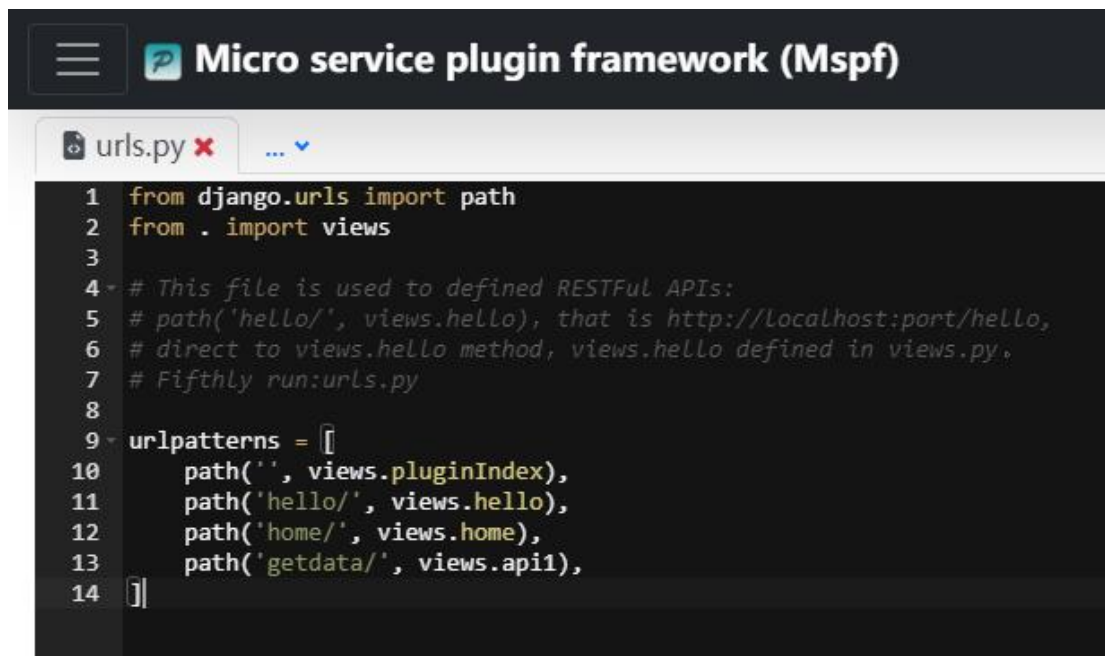
urls.py 中：

```
path('home/', views.home),
```

```
path('getdata/', views.api1),
```

注意：用户要是想注册多个插件，每个 path 的 url 均要唯一。也就是 path(param1, method)中的 param1 在所有插件中都是唯一的。

添加完后，如下图所示：

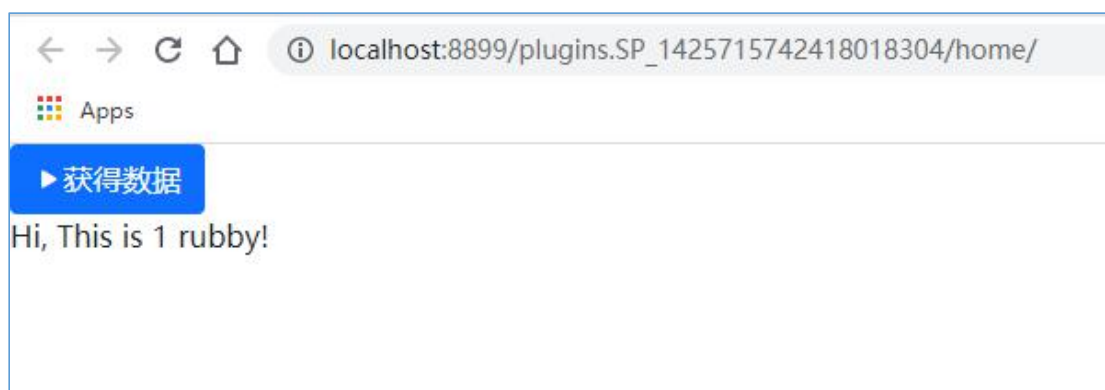


```
1 from django.urls import path
2 from . import views
3
4 # This file is used to defined RESTFul APIs:
5 # path('hello/', views.hello), that is http://localhost:port/hello,
6 # direct to views.hello method, views.hello defined in views.py.
7 # Fifthly run:urls.py
8
9 urlpatterns = [
10     path('', views.pluginIndex),
11     path('hello/', views.hello),
12     path('home/', views.home),
13     path('getdata/', views.api1),
14 ]
```

第七步：

在浏览器访问：

http://localhost:8899/plugins.SP_1425715742418018304/home/ 并点击获得数据按钮，结果如下图所示：



8.部署生产环境

微服务框架是基于 Django 开发的，Django 内置了 HTTP 服务器，通过内置服务器就可以实现访问 Django 的 APP 的前端 HTML，JS，CSS 以及后端 REST APIs。

但是 Django 自身的 HTTP 服务器无法支持大规模的请求和响应。因此, 要将 Django 的 APP 部署到生产环境, 一般采用能支持大规模 HTTP 访问的 Nginx 作为 HTTP 服务器, 用 uWSGI 作为中间件, 用 Django 作为后端应用框架, 提供 REST 服务。

这里普及一下 WSGI 知识:

WSGI: 是 Web Server Gateway Interface 缩写。WSGI 是一个描述 Web server 与 Web application 通信的协议。Server 和 Application 的规范在 PEP 3333 中有具体描述。要实现 WSGI 协议, 必须同时实现 Web server 和 Web application, 当前运行在 WSGI 协议之上的 Web 技术框架有 Bottle, Flask, Django。

uwsgi: 与 WSGI 一样是一种通信协议, 是支持 uWSGI 服务器的独占协议, 用于定义传输信息的类型 (type of information), 每一个 uwsgi packet 前 4byte 为传输信息类型的描述, 与 WSGI 协议是两种东西, 据说该协议是 fcgi 协议的 10 倍快。

uWSGI: 是一个 Web 服务器, 实现了 WSGI 协议、uwsgi 协议、http 协议等。

WSGI 协议主要包括 server 和 application 两部分:

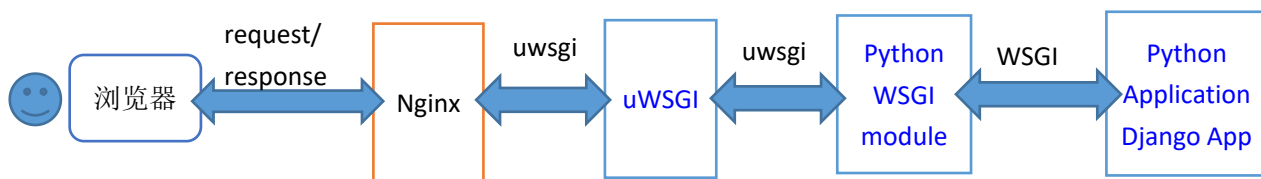
WSGI server 负责从客户端接收请求, 将 Request 转发给 Application, 将 Application 返回的 Response 返回给客户端;

WSGI application 接收由 Server 转发的 Request, 处理请求, 并将处理结果返回给 Server。application 中可以包括多个栈式的中间件 (Middle-wares), 这些中间件需要同时实现 Server 与 Application, 因此可以在 WSGI 服务器与 WSGI 应用之间起调节作用: 对服务器来说, 中间件扮演应用程序, 对应用程序来说, 中间件扮演服务器。

WSGI 协议其实是定义了一种 Server 与 Application 解耦的规范, 即可以有多个实现 WSGI Server 的服务器, 也可以有多个实现 WSGI application 的框架, 那么就可以选择任意的 Server 和 Application 组合实现自己的 Web 应用。例如 uWSGI 和 Gunicorn 都是实现了 WSGI server 协议的服务器, Django, Flask 是实现了 WSGI application 协议的 Web 框架, 可以根据项目实际情况搭配使用。

针对 WSGI 的学习资料, 网上有很多, 此处不再赘述。

HTTP 请求的访问服务链如下图所示:



(1) 首先 Nginx 是对外的服务接口，外部浏览器通过 url 访问 nginx,

(2) Nginx 接收到浏览器发送过来的 http 请求，将包进行解析，分析 url，如果是静态文件请求就直接访问用户给 Nginx 配置的静态文件目录，直接返回用户请求的静态文件，如果不是静态文件，而是一个动态的请求，那么 nginx 就将请求转发给 uWSGI，uWSGI 接收到请求之后将包进行处理，处理成 wsgi 可以接受的格式，并发给 wsgi，wsgi 根据请求调用应用程序的某个文件的某个函数，最后处理完将返回值再次交给 uWSGI，uWSGI 将返回值进行打包，打包成 uwsgi 能够接收的格式，uWSGI 接收 wsgi 发送的请求，并转发给 Nginx，Nginx 最终将返回值返回给浏览器。

(3) 要知道第一级的 Nginx 并不是必须的，uWSGI 完全可以完成整个的和浏览器交互的流程，但是要考虑到某些情况

- 安全问题，程序不能直接被浏览器访问到，而是通过 Nginx，Nginx 只开放某个接口，uWSGI 本身是内网接口，这样运维人员在 Nginx 上加上安全性的限制，可以达到保护程序的作用。
- 负载均衡问题，一个 uWSGI 很可能不够用，即使开了多个 work 也是不行，毕竟一台机器的 CPU 和内存都是有限的，有了 Nginx 做代理，一个 Nginx 可以代理多台 uWSGI 完成 uWSGI 的负载均衡。
- 静态文件问题，用 Django 或是 uWSGI 这种东西来负责静态文件的处理是很浪费的行为，而且他们本身对文件的处理也不如 Nginx 好，所以整个静态文件的处理都直接由 Nginx 完成，静态文件的访问完全不去经过 uWSGI 以及其后面的东西。

如果要进一步学习 uWSGI，建议访问以下链接：

https://uwsgi-docs-zh.readthedocs.io/zh_CN/latest/WSGIquickstart.html#wsgi

8.1 安装 Python

我们选择阿里云的 Ubuntu 服务器作为生产环境，首先通过服务器升级，完成系统的升级：

```
sudo apt-get update
```

```
sudo apt-get upgrade
```

升级到最新资源库。接下来安装 Python。由于每个插件使用的 Python 版本可能不一样，因此可以多安装几个版本。命令如下：

```
sudo apt-get install python2.7
```

```
sudo apt-get install python3.8
```

```
sudo apt-get install python3.9
```

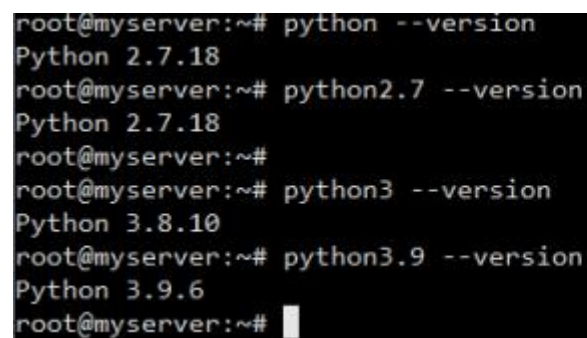
安装完成后，要想验证是否已经安装了，就输入：

```
python --version
```

```
Python3 --version
```

```
Python3.8 --version
```

```
Python3.9 --version
```



```
root@myserver:~# python --version
Python 2.7.18
root@myserver:~# python2.7 --version
Python 2.7.18
root@myserver:~#
root@myserver:~# python3 --version
Python 3.8.10
root@myserver:~# python3.9 --version
Python 3.9.6
root@myserver:~#
```

8.2 安装 Pip

安装 Pip，不同的版本安装命令是不一样的。

Python 2.7:

```
sudo apt-get install python-pip
```

Python 3.8:

```
sudo apt-get install python3-pip
```

Python 3.9:

```
sudo apt-get install python3.9-pip
```

显示版本和路径

```
pip --version    # Python2.x 版本命令
```

```
pip3 --version   # Python3.x 版本命令
```

获取帮助

```
pip --help
```

升级 pip

```
pip install -U pip
```

如果读者遇到任何问题可以上网查下教程，很多，此处不再赘述。最终结果要保证 pip 能够用来安装 Python 的相关组件。

8.3 安装 Django

python2 安装 Django:

```
sudo apt-get install python-django -y
```

python3 安装 Django:

```
sudo apt-get install python3-django -y
```

升级需要执行命令:

```
python -m pip install --upgrade django
```

测试是否安装成功:

```
root@myserver:~# python3
Python 3.8.10 (default, Jun  2 2021, 10:49:15)
[GCC 9.4.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> import django
>>> django.get_version()
'2.2.12'
>>> exit()
root@myserver:~# python3.9
Python 3.9.6 (default, Jul  3 2021, 16:40:50)
[GCC 9.3.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> import django
>>> django.get_version()
'3.2.6'
>>>
```

如果执行错误，不能直接执行 pip 命令，因为系统中自带 Python2.7 和 Python3.5 分别占用了“python”和“python3”这两个命令，如果想给 Python3.6 安装

Django（也只有 Python3.6 才能装 Django2），必须先进入 Python3.6 的“dist-packages”目录。

执行命令：

```
cd /usr/local/lib/python2.7/dist-packages/
```

```
pip install django
```

```
cd /usr/local/lib/python3.8/dist-packages/
```

```
pip3 install django
```

```
cd /usr/local/lib/python3.9/dist-packages/
```

```
pip3.9 install django
```

如果读者遇到任何问题可以上网查下教程，此处不再赘述。

8.4 安装 uWSGI

安装 uWSGI，一般安装方法是利用 pip 安装：

```
sudo pip install uwsgi
```

不过多数时候安装不是很成功，建议用以下方法安装：

```
apt-get install build-essential python-dev
```

以上命令是支持编译运行 make。

```
wget http://projects.unbit.it/downloads/uwsgi-latest.tar.gz
```

```
tar zxvf uwsgi-latest.tar.gz
```

```
cd /home/cdh/
```

```
make
```

运行了 make 命令构建之后，当前目录下，你会获得一个 uwsgi 二进制文件，如下图所示：

```
root@myserver:~# ls
uwsgi-2.0.19.1  uwsgi-2.0.19.1.tar.gz  uwsgi-latest.tar.gz
root@myserver:~# cd uwsgi-2.0.19.1
root@myserver:~/uwsgi-2.0.19.1# ls
apache2      hello.py          PaaSPluginFwk    tests            uwsgi.h
attach.py    INSTALL          PKG-INFO         uwsgi           uwsgi_main.c
bin          install.sh        plugins          uwsgibuild.lastcflags  valgrind
buildconf    lib              proto            uwsgibuild.lastprofile  vassals
contrib      LICENSE          README           uwsgibuild.log        vhosttest
CONTRIBUTORS  logo_uWSGI.png  setup.cpyext.py  uwsgiconfig.py
core         logo_uWSGI.svg  setup.py          uwsgidecorators.py
examples     Makefile         setup.pyuwsgi.py uwsgidsl.rb
ext          mongrel2-uwsgi.conf  t              uwsgi.gemspec
```

检测是否安装成功:

```
uwsgi --version
```

```
root@myserver:~/uwsgi-2.0.19.1# uwsgi --version
2.0.18-debian
root@myserver:~/uwsgi-2.0.19.1#
```

接下来, 安装 Python38 插件到 uWSGI 中:

```
apt-get install uwsgi-plugin-python
```

8.5 安装 Nginx

安装 Nginx 的命令是:

```
apt-get install nginx
```

安装完成后, 使用 Nginx 命令来启动 Nginx:

启动 nginx: `service nginx start`

重启 nginx: `service nginx restart`

停止 nginx: `service nginx stop`

查看 nginx 运行状态: `service nginx status`

8.6 配置 uwsgi.ini 文件

Mspf 微服务框架利用 uWSGI 启动来支持对其访问。为了能简化启动参数设置, 设计了 uwsgi.ini 配置文件辅助启动。ini 文件的内容如下:

```
[uwsgi]
# 项目目录
chdir=/root/uwsgi-2.0.19.1/PaaSPluginFwk
# 指定项目的 application
module=conf.wsgi:application
plugin=python38
# 进程个数
# workers=5
processes=2
threads=2
pidfile=%(chdir)/scripts/uwsgi.pid
# 指定 IP 端口
socket=127.0.0.1:3000
```

```
wsgi-file=%(chdir)/conf/wsgi.py
# 启动 uwsgi 的用户名和用户组
uid=root
gid=root
# 启用主进程
master=true
# 自动移除 unix Socket 和 pid 文件当服务停止的时候
vacuum=true
# 序列化接受的内容，如果可能的话
thunder-lock=true
# 启用线程
enable-threads=true
# 设置自中断时间
harakiri=30
# 设置缓冲
post-buffering=4096
# 设置日志目录
daemonize=%(chdir)/scripts/uwsgi.log
stats=%(chdir)/scripts/uwsgi.status
```

cddir 是项目的目录，也就是项目下载到哪里了，就设置到哪里。

注意：uWSGI 和 Nginx 之间有 3 种通信方式, unix socket, TCP socket 和 http。

Nginx 的配置必须与 uwsgi 配置保持一致。

```
# 选项 1, 使用 unix socket 与 nginx 通信
# 仅限于 uwsgi 和 nginx 在同一主机上情形
# Nginx 配置中 uwsgi_pass 应指向同一 socket 文件地址
socket=/run/uwsgi/%(project).sock

# 选项 2, 使用 TCP socket 与 nginx 通信
# Nginx 配置中 uwsgi_pass 应指向 uWSGI 服务器 IP 和端口
socket==0.0.0.0:8000 或则 socket=:8000

# 选项 3, 使用 http 协议与 nginx 通信
# Nginx 配置中 proxy_pass 应指向 uWSGI 服务器 IP 和端口
http==0.0.0.0:8000
```

在本项目中，采用第 2 个选项。

8.7 配置 Nginx 文件

配置 Nginx 文件主要是配置 Nginx 的 default 文件。该文件在 /etc/nginx/sites-enabled/目录下，文件内容如下：

```
# Default server configuration
#
upstream django {
    server 127.0.0.1:3000;
}
server {
    listen 80 default_server;
    listen [::]:80 default_server;

    # SSL configuration
    #
    # listen 443 ssl default_server;
    # listen [::]:443 ssl default_server;
    #
    # Note: You should disable gzip for SSL traffic.
    # See: https://bugs.debian.org/773332
    #
    # Read up on ssl_ciphers to ensure a secure configuration.
    # See: https://bugs.debian.org/765782
    #
    # Self signed certs generated by the ssl-cert package
    # Don't use them in a production server!
    #
    # include snippets/snakeoil.conf;

    root /var/www/html;

    # Add index.php to the list if you are using PHP
    #index index.html index.htm index.nginx-debian.html;
```

```

server_name _;

#location / {
    # First attempt to serve request as file, then
    # as directory, then fall back to displaying a 404.
    # try_files $uri $uri/ =404;
    #}

    location /static {
        alias /root/uwsgi-2.0.19.1/PaaSPluginFwk/common_static;
    }

    location / {
        # direct all params to uwsgi
        include /etc/nginx/uwsgi_params;

        # uwsgi's Ip and port
        uwsgi_pass django;

        uwsgi_param                                UWSGI_SCRIPT
/root/uwsgi-2.0.19.1/PaaSPluginFwk/conf.uwsgi;

        uwsgi_param                                UWSGI_SCRIPT
/root/uwsgi-2.0.19.1/PaaSPluginFwk;

        #index index.html index.htm;

        client_max_body_size 100m;
    }

    # pass PHP scripts to FastCGI server
    #
    #location ~ /\.php$ {
    #    include snippets/fastcgi-php.conf;
    #
    #    # With php-fpm (or other unix sockets):
    #    fastcgi_pass unix:/var/run/php/php7.4-fpm.sock;
    #    # With php-cgi (or other tcp sockets):
    #    fastcgi_pass 127.0.0.1:9000;

```

```

#}

# deny access to .htaccess files, if Apache's document root
# concurs with nginx's one
#
#location ~ /\.ht {
#    deny all;
#}
}

```

在该文件中，建立了一个变量：

```

upstream django {
    server 127.0.0.1:3000;
}

```

修改了 location / {}:

```

location /static {
    alias /root/uwsgi-2.0.19.1/PaaSPluginFwk/common_static;
}

```

static 目录在/root/uwsgi-2.0.19.1/的 PaaSPluginFwk 下面，也就是项目。

alias 是必须的参数，不能去掉。

default 还添加了一个 location /static {}:

```

location / {
    # direct all params to uwsgi
    include /etc/nginx/uwsgi_params;

    # uwsgi's lp and port
    uwsgi_pass django;

    uwsgi_param UWSGI_SCRIPT
/root/uwsgi-2.0.19.1/PaaSPluginFwk/conf.wsgi;

    uwsgi_param UWSGI_SCRIPT /root/uwsgi-2.0.19.1/PaaSPluginFwk;

    #index index.html index.htm;

    client_max_body_size 100m;
}

```


其中，/root/uwsgi-2.0.19.1/是 PaaSPluginFwk 项目所在目录。

一般地，需要设置 nginx.conf 的用户名。这个文件在下面目录下：

```
root@myserver:/etc/nginx# ls
conf.d          koi-win          nginx.conf       sites-enabled
fastcgi.conf    mime.types       proxy_params     snippets
fastcgi_params  modules-available scgi_params      uwsgi_params
koi-utf         modules-enabled  sites-available  win-utf
root@myserver:/etc/nginx#
```

文件内容的第一行一般是 user www-data;

这需要看 Nginx 的安装用户，如果是 root 用户安装，就改成 user root; 而且这个参数是用户访问权限设置，只允许有访问权限的用户能够访问。

8.8 操作命令

通过上述配置讲解，Mspf 已经能够运行在 Ubuntu 服务器了。需要几个命令来启动和停止 uwsgi 有几个命令：

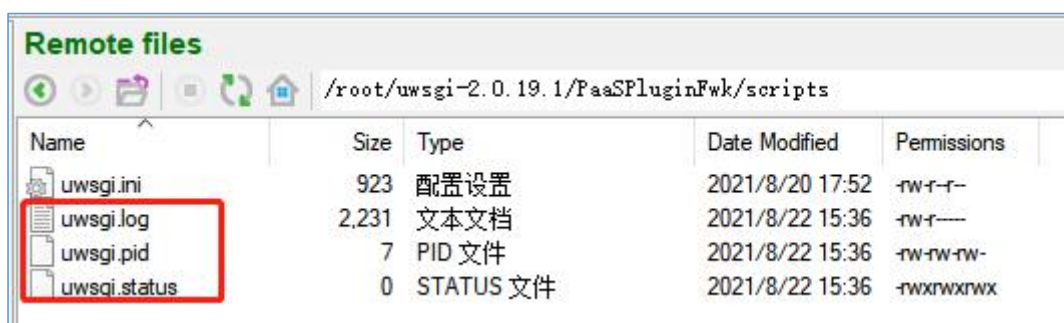
cd <uwsgi> # 转到 uwsgi 目录下

启动命令：

uwsgi --ini PaaSPluginFwk/scripts/uwsgi.ini，运行起来后如下图所示：

```
root@myserver:~/uwsgi-2.0.19.1# uwsgi --ini PaaSPluginFwk/scripts/uwsgi.ini
[uWSGI] getting INI configuration from PaaSPluginFwk/scripts/uwsgi.ini
root@myserver:~/uwsgi-2.0.19.1#
```

在/root/uwsgi-2.0.19.1/PaaSPluginFwk/scripts 目录下，会生成以下几个文件：



Name	Size	Type	Date Modified	Permissions
uwsgi.ini	923	配置设置	2021/8/20 17:52	-rw-r--r--
uwsgi.log	2,231	文本文件	2021/8/22 15:36	-rw-r-----
uwsgi.pid	7	PID 文件	2021/8/22 15:36	-rw-rw-rw-
uwsgi.status	0	STATUS 文件	2021/8/22 15:36	-rwxrwxrwx

查看是 uwsgi 的进程号，可以用以下命令：

cat /root/uwsgi-2.0.19.1/PaaSPluginFwk/scripts/uwsgi.pid

```
root@myserver:~/uwsgi-2.0.19.1# cat /root/uwsgi-2.0.19.1/PaaSPluginFwk/scripts/uwsgi.pid
448389
root@myserver:~/uwsgi-2.0.19.1#
```

查看所有的进程，可以用如下命令：

ps aux | grep uwsgi

```
root@myserver:~/uwsgi-2.0.19.1# ps aux | grep uwsgi
root      448389  0.0  3.6 379124 36820 ?        S1   15:36   0:00 uwsgi --ini PaaSPluginFwk/scripts/uwsgi.ini
root      448395  0.0  2.9 379124 29372 ?        S1   15:36   0:00 uwsgi --ini PaaSPluginFwk/scripts/uwsgi.ini
root      448396  0.0  2.9 379124 29372 ?        S1   15:36   0:00 uwsgi --ini PaaSPluginFwk/scripts/uwsgi.ini
root      448466  0.0  0.0   9032   664 pts/0    S+   15:44   0:00 grep --color=auto uwsgi
root@myserver:~/uwsgi-2.0.19.1#
```

看到进程号与上面 cat 命令的打印出来的主进程是一致的。进程数量是在 ini 文件中设置的，可以参考 8.6 节。

重启 uwsgi，用如下命令：

```
root@myserver:~/uwsgi-2.0.19.1# uwsgi --reload /root/uwsgi-2.0.19.1/PaaSPluginFwk/scripts/uwsgi.pid
root@myserver:~/uwsgi-2.0.19.1#
```

停止命令：

uwsgi --stop /root/uwsgi-2.0.19.1/PaaSPluginFwk/scripts/uwsgi.pid

```
root@myserver:~/uwsgi-2.0.19.1# uwsgi --stop /root/uwsgi-2.0.19.1/PaaSPluginFwk/scripts/uwsgi.pid
root@myserver:~/uwsgi-2.0.19.1#
```

查看执行状态，也可以用如下命令：

uwsgi --connect-and-read

/root/uwsgi-2.0.19.1/PaaSPluginFwk/scripts/uwsgi.status

```
root@myserver:~/uwsgi-2.0.19.1# uwsgi --connect-and-read /root/uwsgi-2.0.19.1/PaaSPluginFwk/scripts/uwsgi.status
{
  "version": "2.0.18-debian",
  "listen_queue": 0,
  "listen_queue_errors": 0,
  "signal_queue": 0,
  "load": 0,
  "pid": 448389,
  "uid": 0,
  "gid": 0,
  "cwd": "/root/uwsgi-2.0.19.1/PaaSPluginFwk",
  "locks": [
    {
      "user 0": 0
    },
    {
      "signal": 0
    },
    {
      "filemon": 0
    }
  ],
}
```

9.Mspf 的数据库设计

数据库，不需要用户创建，框架中已经创建好了。就是 plugin_repository.sqlite3。

这里只是讲解下数据库结构。修改 PluginMaster 中的 models.py:

```
class Plugin(models.Model):
    id = models.CharField('系统标识',max_length = 22, primary_key=True)
    name = models.CharField('插件名称',max_length = 512)
    developer = models.CharField('开发者',max_length = 512)
    pluginVersion = models.CharField('版本',max_length = 20)
    pluginDescription = models.CharField('描述',max_length = 1024)
    useCounting = models.IntegerField('引用')
    likeCounting = models.IntegerField('引用')
    copyright = models.CharField('版权所有',max_length = 512)
    keywords = models.CharField('关键词',max_length = 512)
    pluginType = models.CharField('插件类型',max_length = 32)
    logo = models.CharField('商标',max_length = 512)
    defaultOptions = models.CharField('默认参数',max_length = 512)
    createDatetime = models.DateTimeField('创建时间', auto_now_add=True)
    lastupdate = models.DateTimeField('最后更新',auto_now =True)
    license = models.CharField('许可证',max_length = 512)
    isFree = models.SmallIntegerField('免费')
    pricing = models.DecimalField('价格
',max_digits = 5, decimal_places = 2)
    banned = models.SmallIntegerField('是否封禁')
    parent = models.CharField('父节点',max_length = 22)
    currOwner = models.CharField('当前所有人',max_length = 22)
    owner = models.CharField('组织所有人',max_length = 22)
class Meta:
    db_table = 'Plugin'
```

在 Plugin-codes/PaaSPluginFwk/下面执行: python manage.py migrate

plugin service started

Operations to perform:

Apply all migrations: PluginMaster, admin, auth, contenttypes, sessions

Running migrations:

Applying PluginMaster.0001_initial... OK

Applying contenttypes.0001_initial... OK

Applying auth.0001_initial... OK

Applying admin.0001_initial... OK

Applying admin.0002_logentry_remove_auto_add... OK

Applying admin.0003_logentry_add_action_flag_choices... OK

Applying contenttypes.0002_remove_content_type_name... OK

Applying auth.0002_alter_permission_name_max_length... OK

Applying auth.0003_alter_user_email_max_length... OK

Applying auth.0004_alter_user_username_opts... OK

Applying auth.0005_alter_user_last_login_null... OK
Applying auth.0006_require_contenttypes_0002... OK
Applying auth.0007_alter_validators_add_error_messages... OK
Applying auth.0009_alter_user_last_name_max_length... OK
Applying auth.0010_alter_group_name_max_length... OK
Applying auth.0012_alter_user_first_name_max_length... OK
Applying sessions.0001_initial... OK

```
PS D:\Plugin-codes\PaaSPluginFwk> python manage.py migrate
```

```
plugin service started
```

```
Operations to perform:
```

```
  Apply all migrations: PluginMaster, admin, auth, contenttypes, sessions
```

```
Running migrations:
```

```
Applying PluginMaster.0001_initial... OK
Applying contenttypes.0001_initial... OK
Applying auth.0001_initial... OK
Applying admin.0001_initial... OK
Applying admin.0002_logentry_remove_auto_add... OK
Applying admin.0003_logentry_add_action_flag_choices... OK
Applying contenttypes.0002_remove_content_type_name... OK
Applying auth.0002_alter_permission_name_max_length... OK
Applying auth.0003_alter_user_email_max_length... OK
Applying auth.0004_alter_user_username_opts... OK
Applying auth.0005_alter_user_last_login_null... OK
Applying auth.0006_require_contenttypes_0002... OK
Applying auth.0007_alter_validators_add_error_messages... OK
Applying auth.0009_alter_user_last_name_max_length... OK
Applying auth.0010_alter_group_name_max_length... OK
Applying auth.0012_alter_user_first_name_max_length... OK
Applying sessions.0001_initial... OK
```

再执行: `python manage.py createsuperuser` 创建超级用户

```
PS D:\Plugin-codes\PaaSPluginFwk> python manage.py createsuperuser
```

```
plugin service started
```

```
用户名 (leave blank to use 'dahaicao'): admin
```

```
电子邮件地址: dhcao2003@163.com
```

```
Password:
```

```
Password (again):
```

```
密码长度太短。密码必须包含至少 8 个字符。
```

```
这个密码太常见了。
```

```
密码只包含数字。
```

```
Bypass password validation and create user anyway? [y/N]: y
```

```
Superuser created successfully.
```

再执行: `python manage.py sqlmigrate PluginMaster 0001`

输出:

```
plugin service started
```

```
BEGIN;
```

```
--
```

```
-- Create model Plugin
--
CREATE TABLE "Plugin" ("id" varchar(22) NOT NULL PRIMARY KEY, "name" varchar(512) NOT NULL, "developer" varchar(512) NOT NULL, "pluginVersion" varchar(20) NOT NULL, "pluginDescription" varchar(1024) NOT NULL, "useCounting" integer NOT NULL, "likeCounting" integer NOT NULL, "copyright" varchar(512) NOT NULL, "keywords" varchar(512) NOT NULL, "pluginType" varchar(32) NOT NULL, "logo" varchar(512) NOT NULL, "defaultOptions" varchar(512) NOT NULL, "createDatetime" datetime NOT NULL, "lastupdate" datetime NOT NULL, "license" varchar(512) NOT NULL, "isFree" smallint NOT NULL, "pricing" decimal NOT NULL, "banned" smallint NOT NULL, "parent" varchar(22) NOT NULL, "currOwner" varchar(22) NOT NULL, "owner" varchar(22) NOT NULL);
COMMIT;
```

执行效果是下图所示：

```
PS D:\Plugin-codes\PaaSPluginFwk> python manage.py sqlmigrate PluginMaster 0001
plugin service started
BEGIN;
--
-- Create model Plugin
--
CREATE TABLE "Plugin" ("id" varchar(22) NOT NULL PRIMARY KEY, "name" varchar(512) NOT NULL, "developer" varchar(512) NOT NULL, "pluginVersion" varchar(20) NOT NULL, "pluginDescription" varchar(1024) NOT NULL, "useCounting" integer NOT NULL, "likeCounting" integer NOT NULL, "copyright" varchar(512) NOT NULL, "keywords" varchar(512) NOT NULL, "pluginType" varchar(32) NOT NULL, "logo" varchar(512) NOT NULL, "defaultOptions" varchar(512) NOT NULL, "createDatetime" datetime NOT NULL, "lastupdate" datetime NOT NULL, "license" varchar(512) NOT NULL, "isFree" smallint NOT NULL, "pricing" decimal NOT NULL, "banned" smallint NOT NULL, "parent" varchar(22) NOT NULL, "currOwner" varchar(22) NOT NULL, "owner" varchar(22) NOT NULL);
COMMIT;
PS D:\Plugin-codes\PaaSPluginFwk> █
```

执行：python manage.py dbshell 可以查看对应的表：

输出：

```
plugin service started
SQLite version 3.7.14.1 2012-10-04 19:37:12
Enter ".help" for instructions
Enter SQL statements terminated with a ";"
sqlite> .tables
Plugin                                auth_user_user_permissions
auth_group                            django_admin_log
auth_group_permissions                django_content_type
auth_permission                       django_migrations
auth_user                             django_session
auth_user_groups
sqlite>
```



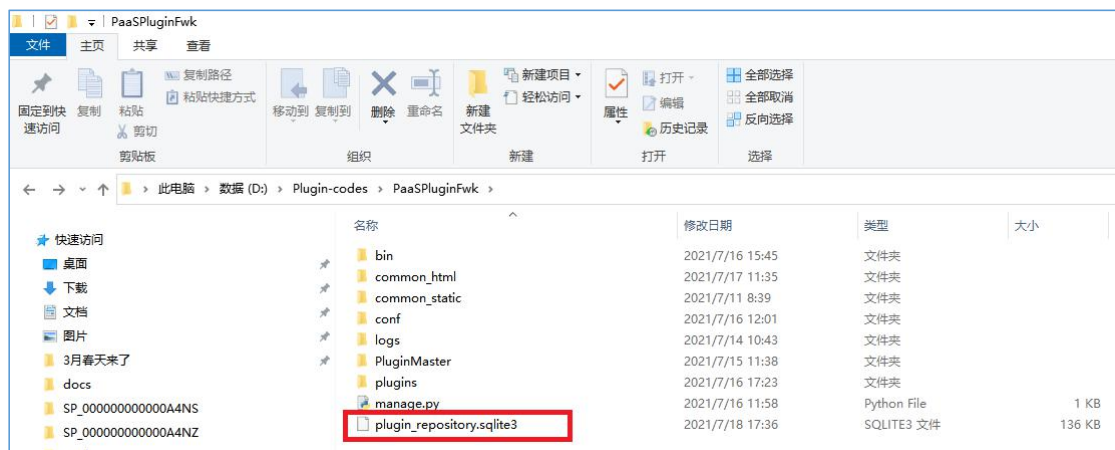
```

PS D:\Plugin-codes\PaaSPluginFwk> python manage.py dbshell
plugin service started
SQLite version 3.7.14.1 2012-10-04 19:37:12
Enter ".help" for instructions
Enter SQL statements terminated with a ";"
sqlite> .tables
Plugin                                auth_user_user_permissions
auth_group                           django_admin_log
auth_group_permissions               django_content_type
auth_permission                      django_migrations
auth_user                           django_session
auth_user_groups
sqlite>

```

可以见到 Plugin 表已经建好在数据库里面了。

我们的数据库就在：



这里，只有输出.exit 才能退出，因为这是一个 sqlite 的 shell 操作界面。

9.FAQ

9.1 出现无法运行 python manage.py dbshell 的处理方法

刚开始使用 Django 时，输入 python manage.py dbshell 命令进入 sqlite3 数据库会提示：

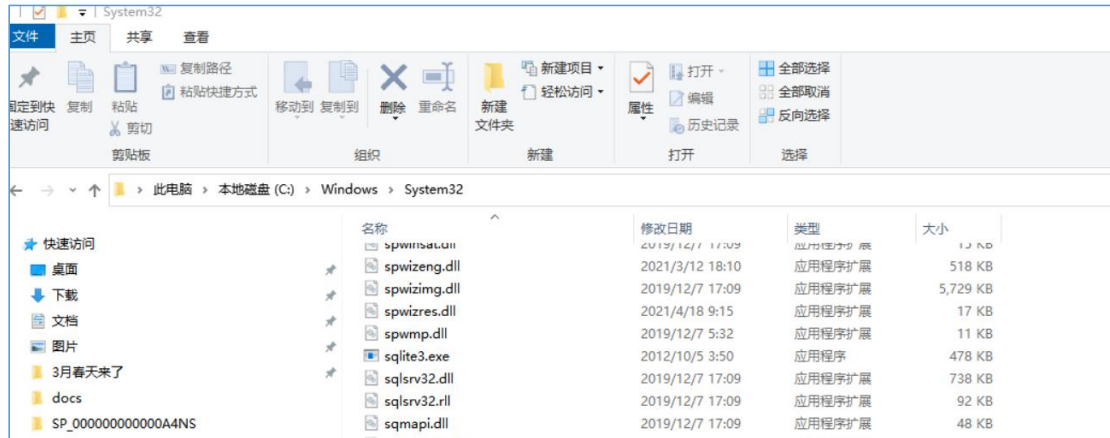
CommandError: You appear not to have the 'sqlite3' program installed or on your path.

这是因为 Django 只是集成了 sqlite3 的访问接口，并没有提供可执行文件，解决办法就是到 [sqlite](https://www.sqlite.org/) 官网下载一个 exe 文件安装到本地盘即可。

1、下载 sqlite3 Sqlite3.exe :

<http://www.sqlite.org/sqlite-shell-win32-x86-3071401.zip>

2、将 sqlite3.exe 文件放入（C:\Windwos\System32）中



3、在项目文件夹下进入 cmd dos 界面或这 pycharm 中调用中断窗口，输入 python manage.py dbshell

```
C:\Python39\python.exe: can't open file 'D:\Plugin-codes\PaaSPluginFwk\bin\manage.py': [Errno 2] No such file or directory
PS D:\Plugin-codes\PaaSPluginFwk\bin> cd ..
PS D:\Plugin-codes\PaaSPluginFwk> cd ..
PS D:\Plugin-codes> cd PaaS
PS D:\Plugin-codes\PaaSPluginFwk> python manage.py dbshell
plugin service started
CommandError: You appear not to have the 'sqlite3' program installed or on your path.
PS D:\Plugin-codes\PaaSPluginFwk> python manage.py dbshell
plugin service started
SQLite version 3.7.14.1 2012-10-04 19:37:12
Enter ".help" for instructions
Enter SQL statements terminated with a ";"
sqlite> █
```

输入.exit 退出，输入.help 打开帮助