Django REST Framework 实现业务 api 并自动文档化

使用MySql替换SQL

业务场景下，还是需要使用扩展性较好的Mysql等主流数据库，SQL只适合入门调试使用。

为Django项目创建数据库

```
create database rouboinfo default charset utf8 collate utf8_general_ci;
```

配置Django项目的数据库信息

```
## settings.py
DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.mysql',
        'NAME': 'rouboinfo',
        'USER': 'root',
        'PASSWORD': 'junjian11',
        'HOST': '127.0.0.1'
    }
}
## top __init__.py
import pymysql
pymysql.install_as_MySQLdb()
```

初始化数据库

```
python manage.py makemigrations
python manage.py migrate
```

观察mysql内的数据库是否初始化完成

```
mysql> use rouboinfo;
Reading table information for completion of table and column names
You can turn off this feature to get a quicker startup with -A

Database changed
mysql> show tables;
+----------------------------+
| Tables_in_rouboinfo        |
+----------------------------+
| auth_group                 |
| auth_group_permissions     |
| auth_permission            |
```

```
| auth_user                  |
| auth_user_groups           |
| auth_user_user_permissions |
| django_admin_log           |
| django_content_type        |
| django_migrations          |
| django_session             |
+----------------------------+
10 rows in set (0.00 sec)
```

Django ORM 实践

> Django通过自定义Python类的形式来定义具体的模型，每个模型的物理存在方式就是一个Python的类Class，每个模型代表数据库中的一张表，每个类的实例代表数据表中的一行数据，类中的每个变量代表数据表中的一列字段。Django通过模型，将Python代码和数据库操作结合起来，实现对SQL查询语言的封装。也就是说，你可以不会管理数据库，可以不会SQL语言，你同样能通过Python的代码进行数据库的操作。Django通过ORM对数据库进行操作，奉行代码优先的理念，将Python程序员和数据库管理员进行分工解耦。

ORM（Object-Relational Mapping) 即对象-关系映射，从效果上看，实现了一个可在编程语言里直接使用的虚拟对象数据库。这种思想在很多架构设计中都有体现，android开发中常用的GreenDao就是一个轻量的ORM设计。

设计model（即数据库结构）

因为Django的model机制是ORM的一种实践，所以说，设计了model，就设计了数据库结构。这里我们只演示一个记录设备启动次数的api接口所需的基本字段。

```
from django.db import models

class DeviceReport(models.Model):
    """
    收集设备相关的统计数据
    report_id
        自增id作为主键
    report_type
        上报类型，比如启动上报：open
    report_time
        上报时间戳
    device_id
        可以描述设备的id
    ip_address
        公网ip地址
    """
    report_id = models.IntegerField(primary_key=True)
    report_type = models.CharField(max_length=100)
    report_time = models.DateTimeField(auto_now_add=True)
    device_id = models.CharField(max_length=200)
    ip_address = models.CharField(max_length=50)
```

生效数据库 在每次修改model之后，都需要进行迁移和生效动作，毕竟我们修改了数据库结构了。

```
» python manage.py makemigrations rouboapi
Migrations for 'rouboapi':
  rouboapi/migrations/0001_initial.py
    - Create model DeviceReport

» python manage.py migrate
Operations to perform:
  Apply all migrations: admin, auth, contenttypes, rouboapi, sessions
Running migrations:
  Applying rouboapi.0001_initial... OK
```

执行成功后，会生成rouboapi_devicereport的表。

实现序列化器

我们使用Django REST framework 提供的序列化器简化代码。

```
from rest_framework import serializers
from rouboapi.models import DeviceReport

class DeviceReportSerializer(serializers.HyperlinkedModelSerializer):
    """
    序列化上报接口数据
    """
    class Meta:
        model = DeviceReport
        fields = ('report_id', 'report_type', 'report_time', 'device_id', 'ip_address')
```

实现视图

先使用APIView来简化代码，目前还没研究清楚如果使用ViewSets实现单一的get接口（其他接口不能实现的）的方法。

```
from rouboapi.serializers import DeviceReportSerializer
from rest_framework.views import APIView
from rest_framework.response import Response
from rest_framework import status

class DeviceReport(APIView):
    """
    上报设备信息，无需鉴权
    """
       authentication_classes = []
    permission_classes = []

    def get(self, request, format=None):
        serializer = DeviceReportSerializer(data=request.data)
        if serializer.is_valid():
            serializer.save()
            return Response(serializer.data, status=status.HTTP_200_OK)
```

```
        return Response(serializer.errors, status=status.HTTP_400_BAD_REQUEST)
```

接口测试

浏览器访问：http://127.0.0.1:8000/rouboapi/v1/report/?report_type=open&device_id=testid&ip_address=127.0.0.1 ，出现REST默认界面，作为接口访问时，加入format=json参数即可。

检查数据库的变化：

```
mysql> select * from rouboapi_devicereport;
+-------------+----------------------------+-----------+------------+
| report_type | report_time                | device_id | ip_address |
+-------------+----------------------------+-----------+------------+
| open        | 2018-08-28 11:54:08.724136 | testid    | 127.0.0.1  |
| open        | 2018-08-28 11:55:06.205909 | testid    | 127.0.0.1  |
| open        | 2018-08-28 11:55:08.746301 | testid    | 127.0.0.1  |
+-------------+----------------------------+-----------+------------+
```

文档化

Api接口文档化功能是Django REST Framework提供的一大特色功能，操作也简单。

```
pip install coreapi pygments markdown
```

安装上面依赖包后，只要修改urls.py文件即可：

```
from django.conf.urls import url
from . import views
from rest_framework.documentation import include_docs_urls
API_TITLE = 'roubo api documentation'
API_DESCRIPTION = 'roubo api server for rouboinfo'

urlpatterns = [
    url(r'v1/report/$', views.DeviceReport.as_view(), name='DeviceReport'),
    url(r'docs/', include_docs_urls(title=API_TITLE, description=API_DESCRIPTION,
authentication_classes=[], permission_classes=[]))
]
```

查看样式，包含交互式的接口测试功能。

完成

GitHub - roubo/rouboApi: 基于Django REST framework 实现一些业务api 哔哔哔。