单例模式

确保某一个类只能有一个实例存在并被外界访问到, 方便控制并节约系统资源.
比如, 我通过一个 配置类来读取配置文件的信息。如果在程序运行期间，有很多地方都需要使用配置文件的内容，也就是说，很多地方都需要实例化配置类，这就导致系统中存在多个配置类对象，而这样会严重浪费内存资源，尤其是在配置文件内容很多的情况下。事实上类似配置类，我们希望在程序运行期间只存在一个实例对象。这个时候就用到了单例模式.

实现单例模式

1. 模块导入式

```
class Base():
        def __init__(self,name,pwd):
            self.name = name
            self.pwd = pwd
bs = Base("ming","123123")


from s01 import bs
a1 = bs
a2 = bs
a3 = bs
print(id(a1))
print(id(a2))
print(id(a3))
```

2. __new__构建

```
class Base(object):
    __instance = None  # 私有属性, 不被外界直接访问到

    def __new__(cls, *args, **kwargs):
        if not cls.__instance:
        obj = super(Base, cls).__new__(cls, *args, **kwargs)
        cls.__instance = obj
        return cls.__instance

b1 = Base()
b2 = Base()
b3 = Base()
print(id(b1))
print(id(b2))
print(id(b3))
```

3. 装饰器构建

```
def singleton(cls):
    __instance = {}
    def wrapper():
        if not __instance.get(cls):
            __instance[cls] = cls()
        return __instance.get(cls)
    return wrapper

@singleton
class Base:
    pass

b1 = Base()
b2 = Base()
b3 = Base()
print(id(b1))
print(id(b2))
print(id(b3))
```

djangorestframework-jwt 模块的使用

jwt 是一种认证机制, 生成一个 token 值(一段字符串, 由三段文本信息+ . 构成 jwt 字符串), 可以用来记录用户的一种登录状态.

jwt 认证机制与 session-cookie 机制的比较

jwt 机制: 说白了就是签发和检验 token 值得一种机制. 一般被用来在身份提供者和服务提供者间传递被认证的用户身份信息，以便于从资源服务器获取资源，也可以增加一些额外的其它业务逻辑所必须的声明信息.

jwt 是在服务端生成的, 封装进请求头中并由用户带给客户端, 在客户端进行存储.

优点:

1. 不需要在服务端去保留用户的认证信息或者会话信息，意味着支持 token 认证机制的应用不需要考虑用户在哪一台服务器登陆了, 特别适用于分布式站点的单点登录（SSO）场景, 有利于应用的拓展
2. 通用性好, 支持跨语言
3. 可以在 payload 部分放上一些其他业务需要的敏感信息
4. jwt 的构成非常简单(一段 json 字符串), 字节占用很小, 传输方便
5. 用户退出登录时不再需要服务端销毁 session, 而是直接由客户端进行销毁与服务端无关系.