设计模式

# 装饰者设计模式

### 使用场景

给某个对象(注意是对象，而不是类)的功能进行扩展时，可以考虑使用装饰者设计模式。  
 在IO流这部分内容中，JDK的源码使用了大量的装饰者设计模式。

比如BufferedReader可以对FileReader进行装饰。

### 实例

#### 在Servlet中每个页面都需要解决编码问题，这时可以创建一个filter拦截所有的请求然后统一处理编码问题,最后放行，传入request和response。

#### 但是有一个问题，当请求为get时，处理编码问题有点特殊，需用对每个参数编码

String v=request.getParameter(name);  
 byte[] bytes= v.getBytes("iso8859-1");  
 //encode="utf-8"  
 return new String(bytes,enCode);

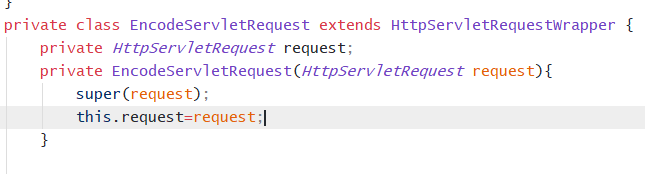
我对每个参数编码后放行时如何传递到请求的页面中，显然这是没办法做到的

#### 那能不能让请求页面的request调用getParameter(name)方法时，先解决编码问题再返回？

答案是可以的，可以自定义类，改变getParameter()方法的实现方式，放行时，将该 类传递过去，而不是传递request，所以requets和自定义类必须具有相同的父类或 父接口。

#### 实现

##### 自定义接口构造方法应传递一个request参数并保存



注意：HttpServletRequestWrapper本身已经是HttpServlet的装饰类，所以 我们这里只需要继承他重写相关方法，而不用实现HttpServerlet，重写所以有法， 简化操作

##### 实现getParameter(name)方法



##### 放行传入自定义类



##### 注意：

请求转发后又经过filter，经过二次编码又会乱码。

解决：该对象在该请求链有效，所以加个flag=true，第一次运行进入然后flag=false，后面不 再进入。

# 单例设计模式

常见的单例，只实例化一次

ServletContext 、Filter、Listener、Servlet、连接池

单例可以避免实例对象的重复创建，可以减少每次创建的时间开销，也可以节省内存空间

### 饿汉模式

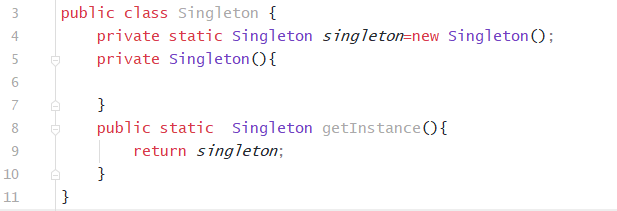
构造函数私有化，保证不被其他类实例化

在类中创建了静态的实例供别人使用，提供一个获取该实例的方法

因为是静态的，所以在类加载时创建一次实例，从此以后供别人使用,在整个程序周期都存在

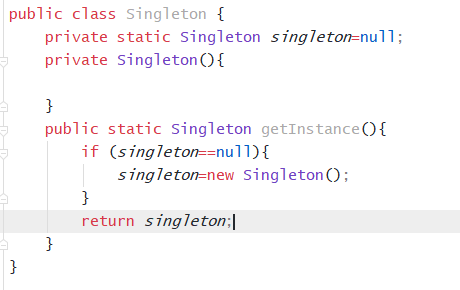
为什么叫饿汉？等不及了，非常饿，所以类加载时就创建

优点：类加载时创建实例且只有一个，不存在多线程问题



### 懒汉模式

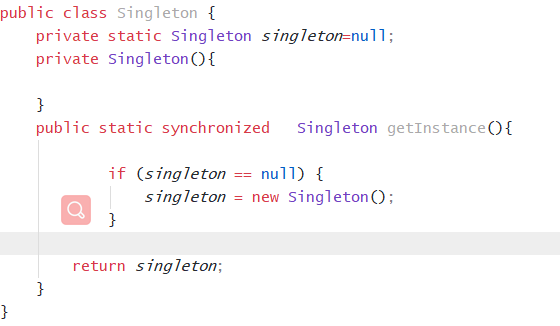
#### 第一次调用时创建



#### 但是如果有多个线程同时访问，那就不能保证只创建一个实例

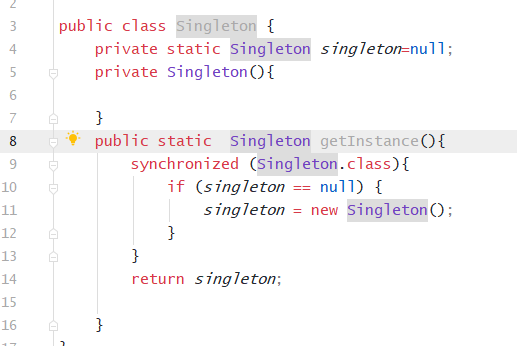
假如第一个线程进入，此时singleton为null，正在创建但还未创建，此时另一线程也判断singleton为null，也进入创建，最终创建了二个实例

解决:给方法加同步关键字



#### 但是给整个方法加锁会严重影响效率

可以给同步代码块加上锁



#### 那么还有一个问题,假如已经创建完成了，singleton不等于null，后面每次每次都要等待锁，然后判断，影响效率

其实除了第一次创建外，后面就不需要等待了，直接判断null返回即可

