说明：使用二级标题和三级标题

Spring包下载地址https://repo.spring.io

## 拦截器

### 实现

一、编写一个拦截器类，实现HandlerInterceptor接口

默认有三个方法

三个方法中的入参中都有一个Object，它表示拦截的目标对象

1. preHandle 在请求被处理之前调用，返回值为Boolean，表示是否将当前请求拦截下来，如果返回false，请求将被终止，true则继续运行。
2. postHandle 在请求被处理之后进行调用，入参中含有ModelAndView，可以改变显示的视图或修改发往视图的方法
3. afterCompletion 在请求结束之后进行调用，通常用做关闭资源。入参有一个Exception

二、将拦截器注册到springMVC中

配置文件中添加注册拦截器

<mvc:interceptor>

<bean class=””/>

</mvc:interceptor>

三、配置拦截器的拦截规则

<mvc:interceptors>

<mvc:interceptor>

<mvc:mapping path=””/>

<bean class=””/>

</mvc:interceptor>

</mvc:interceptors>

### 多个拦截器

* 注册方法

<mvc:interceptors>

<bean class=””/>

<bean class=””/>

</mvc:interceptors>

* 执行顺序

preHandle从上到下

…………（程序处理）

postHandle从下到上

afterCompletion从下到上

### 拦截器的其他实现方式

* 实现WebRequestInterceptor接口
* 里边包含三个方法

preHandle 无返回值

postHandle

afterCompletion

* 弊端，preHandle没有返回值，不能终止请求

### 使用场景

* 使用原则：处理所有请求的共同问题

1. 解决乱码问题
2. 解决权限验证问题

### 拦截器和过滤器的区别

* 过滤器Filter依赖于Servlet容器，基于回调函数，过滤范围大
* 拦截器Interceptor依赖于框架容器，基于反射机制，只过滤请求

### 总结

* 拦截器可以处理Web应用中请求的一些通用性问题
* 共性问题在拦截器中处理，可以减少重复代码，便于维护
* 拦截器只能对action请求起作用，而过滤器则可以对几乎所有的请求起作用。
* 拦截器可以访问action上下文、值栈里的对象，而过滤器不能访问。
* 在action的生命周期中，拦截器可以多次被调用，而过滤器只能在容器初始化时被调用一次。
* 拦截器可以获取IOC容器中的各个bean，而过滤器就不行，这点很重要，在拦截器里注入一个service，可以调用业务逻辑

## Spring事务管理

### 回顾事务

* 事务：逻辑上的一组操作，这组操作要么全部成功，要么全部失败
* 事务的特性：原子性、一致性、隔离性、持久性

原子性：事务是一个不可分割的工作单位，事务中的操作要么都发生，要么都不发生。

一致性：事务前后数据的完整性必须保持一致。

隔离性：多个用户并发访问数据库时，一个用户的事务不能被其他用户的事务所干扰，多个并发事务之间数据要相互隔离。

持久性：一个事务一旦被提交，它对数据库中数据的改变就是永久性的，即使数据库发生故障也不应该对其有任何影响。

### Spring事务管理

* Spring事务管理高层抽象主要包括3个接口

PlatformTransactionManager 平台事务管理器

TransactionDefinition 事务定义信息（隔离、传播、超时、只读）

TransactionStatus 事务具体运行状态

### PlatformTransactionManager

Spring为不同的持久性框架提供了不同的PlatformTransactionManager接口实现

|  |  |
| --- | --- |
| 事务 | 说明 |
| Org.springframework.jdbc.datasource.DataSourceTransaction | 使用Spring JDBC或iBatis进行持久化数据时使用 |
| Org.springframework.orm.hibernate3.HibernateTransactionManager | 使用Hibernate3.0版本进行持久化数据时使用 |
| Org.springframework.orm.jpa.JpaTransactionmanager | 使用JPA进行持久化使用 |
| Org.springframework.jdo.JdoTransactionManager | 当持久化机制是Jdo时使用 |
| Org.springframework.transaction.jta.JtaTransactionManager | 使用一个JTA实现来管理事务，在一个事务跨越多个资源时必须使用 |

### TransactionDefinition

#### 事务隔离级别

* 脏读：一个事务读取了另一个事务改写但还未提交的数据，如果这些数据被回滚，则读到的数据是无效和。
* 不可重复读：在同一个事务中，多次读取同一数据返回的结果有所不同。
* 幻读：一个事务读取了几行记录后，另一个事务插入了一些记录，幻读就发生了。再后来的查询中，第一个事务就会发现有些原来没有的记录。

事务的隔离级别（四种）

|  |  |
| --- | --- |
| 隔离级别 | 含义 |
| DEFAULT | 使用后端数据库默认的隔离级别（spring中的选择项） |
| READ\_UNCOMMITED | 允许读取还未提交的改变了的数据，可能导致脏、幻、不可重复读 |
| READ\_COMMITTED | 允许在并发事务已经提交后读取，可防止脏读，但幻读和不可重复读仍可发生 |
| REPEATABLE\_READ | 对相同字段的多次读取是一致的，除非数据被事务本身改变。可防止脏、不可重复读，但幻读仍可能发生 |
| SERIALIZABLE | 完全服从ACID的隔离级别，确保不发生脏、幻、不可重复读。这在所有的隔离级别中是最慢的的，它是典型的通过完全锁定在事务中涉及的数据表来完成的。 |

#### 事务的传播行为（七种）

|  |  |
| --- | --- |
| 事务传播行为类型 | 说明 |
| PROPAGATION\_REQUIRED | 支持当前事务，如果不存在就新建一个 |
| PROPAGATION\_SUPPORTS | 支持当前事务，如果不存在，就不使用事务 |
| PROPAGATION\_MANDATORY | 支持当前事务，哪果不存在，抛出异常 |
| PROPAGATION\_REQUIRES\_NEW | 如果有事务存在，挂起当前事务，创建一个新的事务 |
| PROPAGATION\_NOT\_SUPPORTED | 以非事务方式运行，如果有事务存在，挂起当前事务 |
| PROPAGATION\_NEVER | 以非事务方式运行，如果有事务存在，抛出异常 |
| PROPAGATION\_NESTED | 如果当前事务存在，则嵌套事务执行 |

### Spring事务管理方式

* 编程式的事务管理

在实际应用中很少使用

通过TransactionTemplate手动管理事务

* 使用XML配置声明式事务

开发中推荐使用（代码侵入性最小）

Spring的声明式事务是通过AOP实现的