SSM

学习技术Spring ,SpringMVC,Maven高级,SpringBoot,MyBatisPlus。

# Spring

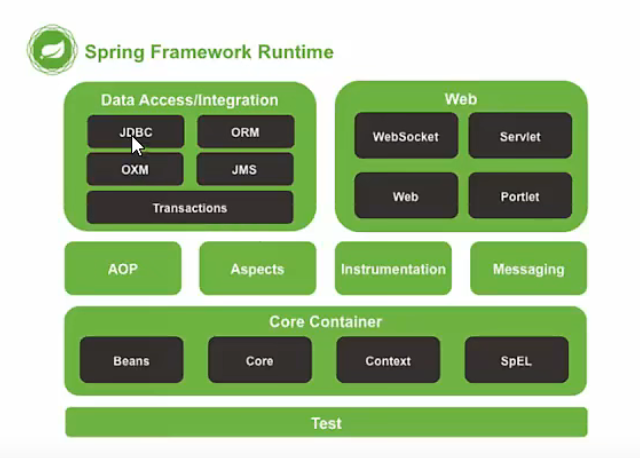
### Spring 技术

Spring Framework Spring框架 = Spring + SpringMVC

Spring Boot 快速开发

Spring Cloud 分布式开发

### Spring系统架构



Core Container 核心容器：存放对象 ⭐

Aop：面向切面编程 ⭐

Aspects：Aop思想实现 ⭐

Data Access：数据访问 ⭐

Data Intergration：数据集成 ⭐

Transaction：事务 ⭐

Web：Web开发

Test：测试，单元测试和集成测试

## 核心容器

### IOC

IOC：控制反转（Inversion of Control）

概念：使用对象时，在程序中不要主动使用new产生的对象，转换 为由外部提供对象。

目的：解耦

Spring的实现：提供了一个容器，称位IOC容器，充当外部。IOC容 器管理对象的创建和初始化，被管理的对象在IOC 容器中统称为Bean。

#### 实现

1. 导入坐标：

在pom.xml文件中导入坐标。

Spring的坐标名称为spring-context

在org.springframework包下

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-context</artifactId>

<version>5.3.10</version>

</dependency>

1. 创建配置文件applicationContext.xml，配置Bean

<!-- 配置bean-->

<!-- bean标签表示配置bean-->

<!-- id属性标识给bean起名字-->

<!-- class属性表示给bean定义类型-->

<bean id="bookdao" class="dao.impl.Bookdaoimpl" />

<bean id="bookservice" class="servie.impl.Bookserviceimpl"/>

1. 获取IOC容器

IOC容器的实现接口名称为 ApplicationContext

ApplicationContext为接口，不能new一个对象，所以new一个它的实现类ClassPathXmlApplicationContext，给我们的bean的配置文件applicationContext.xml作为参数。

public static void main(String[] args) {

//获取IOC容器

ApplicationContext context = new ClassPathXmlApplicationContext("applicationContext.xml");

}

1. 获取bean

使用getBean来获取Bean，参数为配置文件的Bean的id。

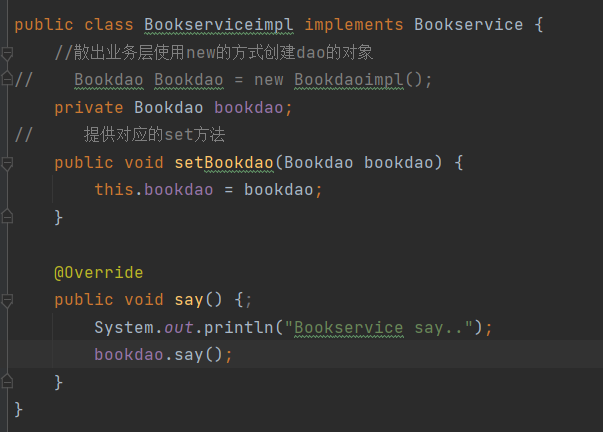


### DI

DI：依赖注入：Dependence Injection，在容器中建立bean和bean的依赖关系

#### 实现

1.在业务类中，添加set方法，提供依赖注入的入口



1. 在配置文件中进行依赖注入

<!-- 依赖注入的配置 -->

<bean id="bookservice" class="servie.impl.Bookserviceimpl">

<!-- 配置dao在service的依赖-->

<!-- property标签表示配置当前的bean的属性-->

<!-- name属性表示配置哪一个具体的属性-->

<!-- ref属性表示参照哪一个bean-->

<property name="bookdao" ref="bookdaobean"></property>

</bean>



#### 依赖注入的方法

##### set方式（setter注入）

⚪引用类型

1. 在bean中定义引用类型的属性并提供可访问的set方法
2. 在配置中使用property标签ref属性注入引用类型的对象

⚪简单类型（基本数据类型和String）

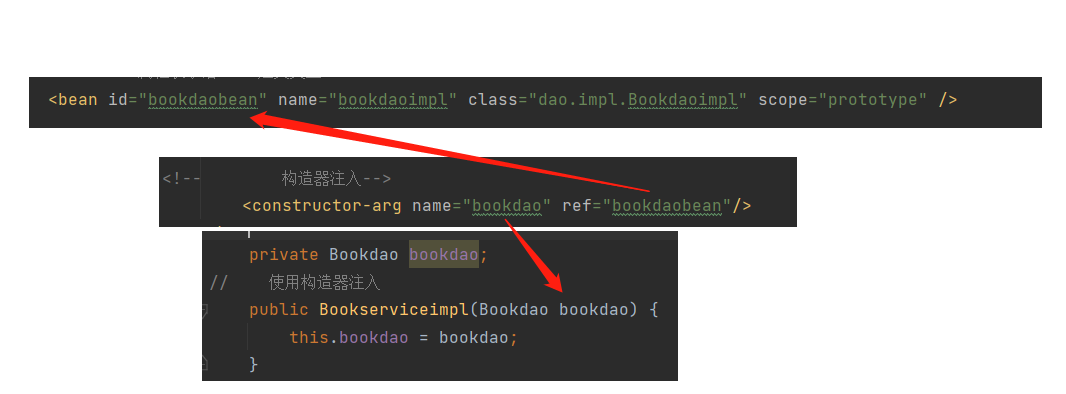
1. 在bean中定义基本类型的属性并提供可访问的set方法
2. 在配置文件在相对应的bean中配置，使用property标签的value属性注入值，name属性为基本类型的名称。



##### 构造方法（构造器注入）

⚪引用类型

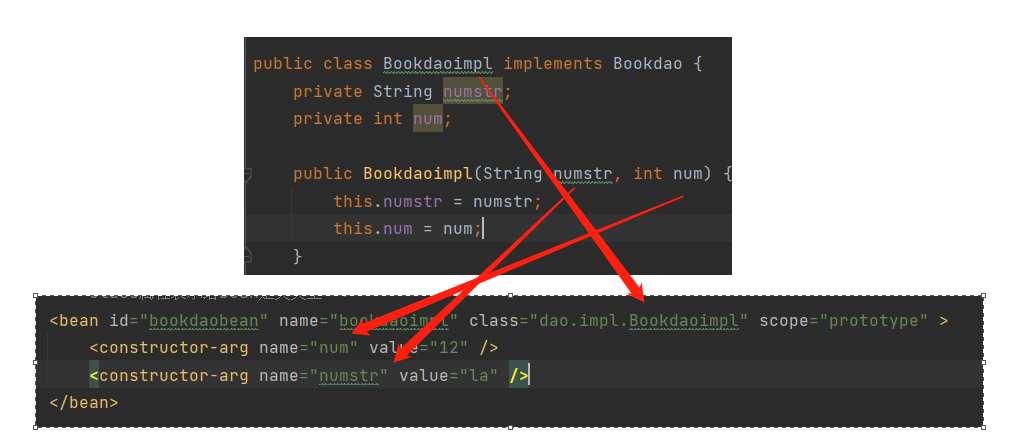
1. 在bean中定义引用类型的属性并提供可访问的构造方法
2. 在配置文件在相对应的bean中配置，使用constructor-arg标签的name属性为构造函数的参数名，ref为引用的对应的bean



⚪简单类型（基本数据类型和String）

(1)在bean中定义基本类型的属性并提供可访问的构造方法

(2)在配置文件在相对应的bean中配置，使用constructor-arg标签的name属性为构造函数的参数名，value为对应的值



##### 依赖自动装配

IOC容器根据bean所依赖的资源在容器中自动查找并注入到bean中的过程为自装配

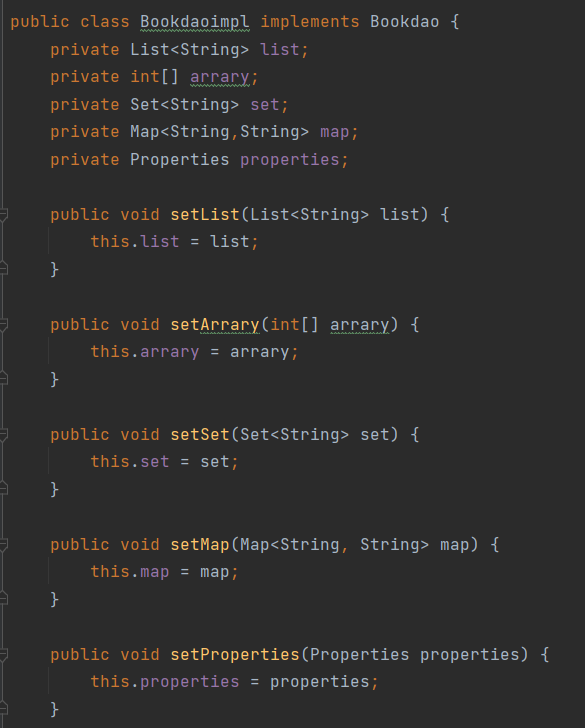
自动装配，需要提供对应stter方法，之后在配置文件对应的bean使用autowire属性byName就是按照名称自动注入，byType就是按照类型自动注入



##### 集合注入

数组，list，set，Map和Properties的注入。

在对应的类中，定义对应的类型，并给set方法



<bean id="bookdaobean" name="bookdaoimpl" class="dao.impl.Bookdaoimpl" scope="prototype" >

###### 数组的注入

<property name="arrary">

<array>

<value>100</value>

<value>200</value>

<value>300</value>

<value>400</value>

</array>

</property>

###### List的注入

<property name="list">

<list>

<value>liyanjun</value>

<value>nan</value>

<value>gas</value>

<value>123</value>

</list>

</property>

###### Set注入

<property name="set">

<set>

<value>liyanjun</value>

<value>liyanjun</value>

<value>nan</value>

<value>gas</value>

<value>123</value>

</set>

</property>

###### Map注入

<property name="map">

<map>

<entry key="country" value="china"/>

<entry key="city" value="jingmneg"/>

</map>

</property>

###### properties注入

<property name="properties">

<props>

<prop key="country">china</prop>

</props>

</property>

</bean>

### bean的基础配置

名称：bean

类型：标签

所属：beans标签

功能：定义Spring核心容器管理的对象

格式：<beans>

<bean></bean>

</beans>

属性列表：id，bean的id，使用容器可以通过id值来获取对应的bean， 在一个容器中id值唯一

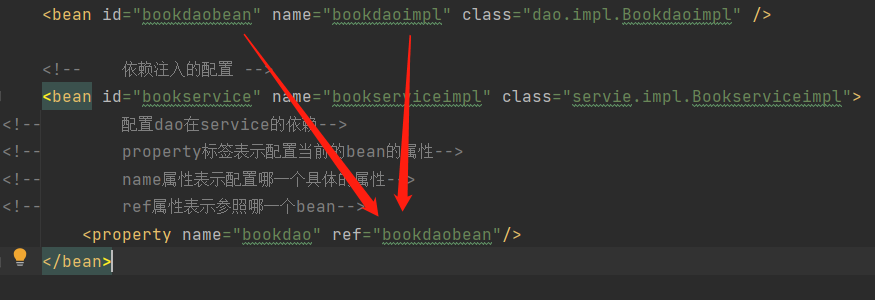
class：bean的类型，即配置的bean的全路径类名

### bean的别名配置

bean标签的name属性，可以起任意别名



id和name都可以作为依赖注入ref的值。两者等同，都作为bean的对象



### bean的作用范围配置

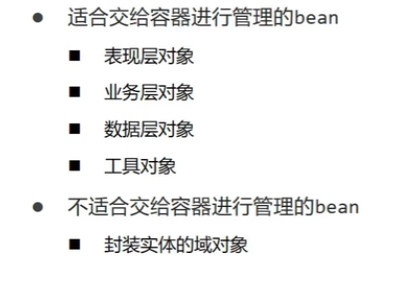
默认创建单例对象

<bean id="bookdaobean" name="bookdaoimpl" class="dao.impl.Bookdaoimpl" scope="prototype" />

属性scope，两个值，singleton和prototype。

singleton为单例模式，也是默认的模式。

prototype为多例模式，bean可以创建多个对象。

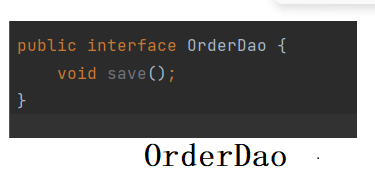


### bean实例化

#### 1.构造方法

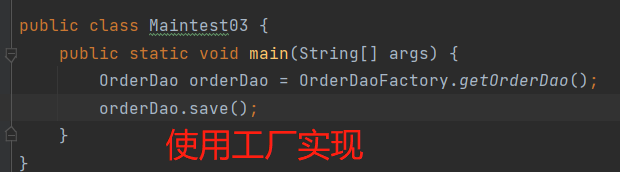
bean本质是对象，创建bean使用构造方法(使用new方法，Spring调用的是无参构造方式，没有无参构成将抛出异常BeanCreationException)。

#### 2.静态工厂









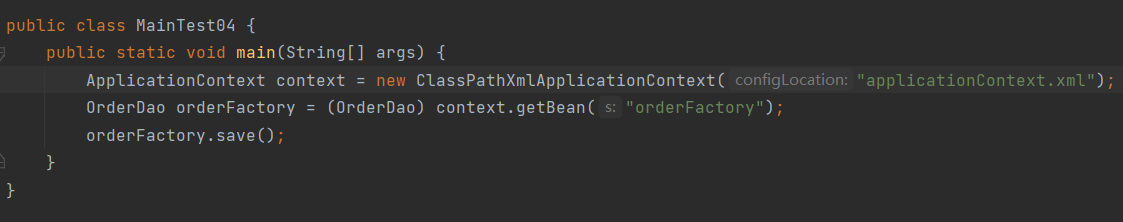
优势：对象不用自己new，可以通过工厂来创建对象。

Spring中的工厂方法：



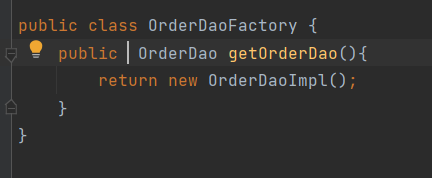
使用factory-method的属性来指定工厂的方法。

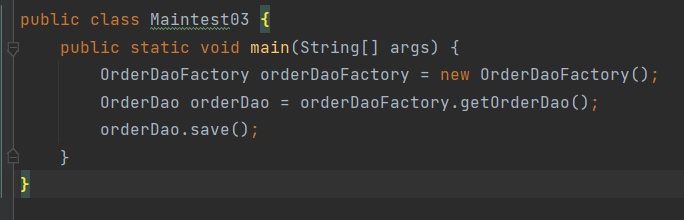
然后就可以实例化bean了。



#### 3.⭐实例工厂和FactoryBean

实例工厂就是里面的方法不为静态的





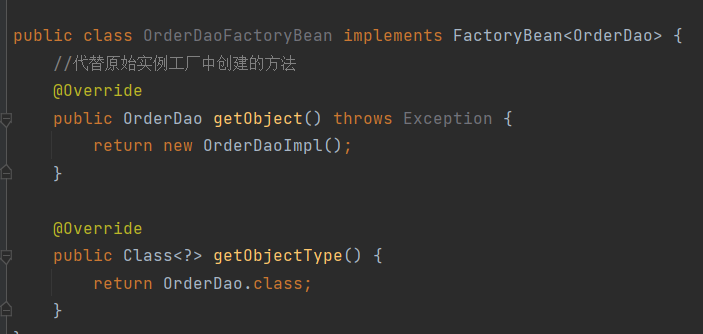
实例化时先new一个工厂的对象，用new出来的对象进行操作。

使用spring实现就是创建一个Factorybean继承接口FactoryBean实现里面的两个方法。

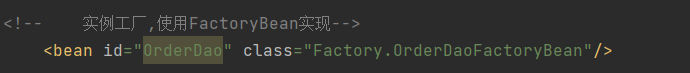
第一个方法getObject()返回实例工厂的实现类

第二个方法获取类的类型，就返回类的字节码。

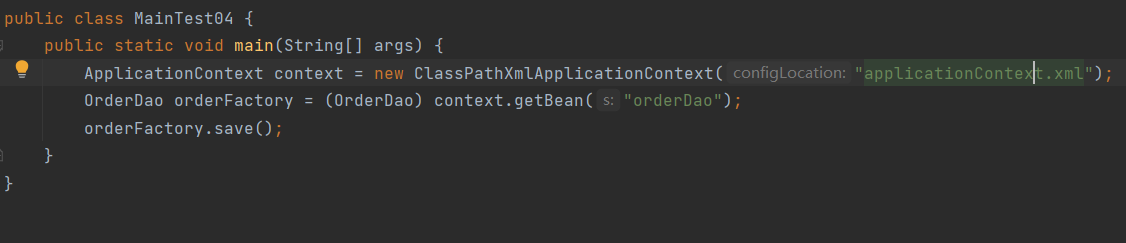
还有第三个方法是 isSingleton ,返回true就是单例对象，false就是多例对象



在去配置文件中配置



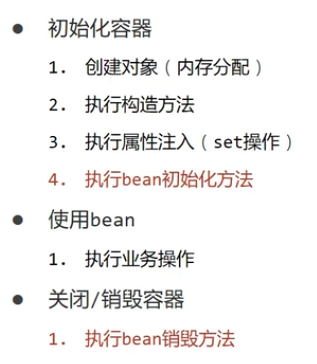
之后去获取这个bean



### bean的生命周期

在实现类中的init，destory方法，之后在配置中添加init-method属性指定初始化方法，添加destory-method属性指定销毁方法。

初始化执行bean之前就执行了，销毁bean没有执行，需要关闭容器才会执行。将ApplicationContext改为ClassPathXmlApplicationContext，然后关闭容器context.close。或者使用关闭钩子， context.registerShutdownHook。



### 数据源对象管理

#### 管理数据源druid

1.导入druid坐标

<dependency>

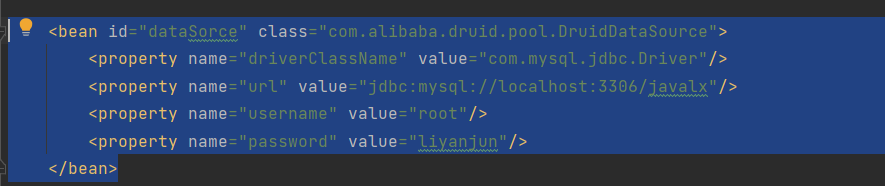
<groupId>com.alibaba</groupId>

<artifactId>druid</artifactId>

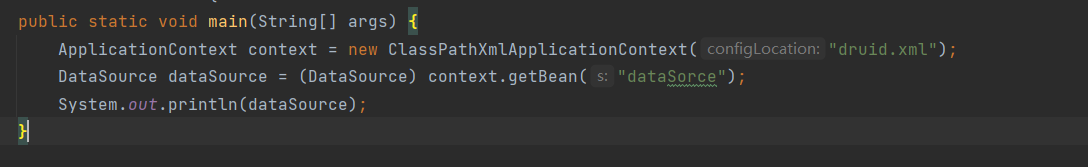
<version>1.1.16</version>

</dependency>

1. 在配置文件中配置数据源druid

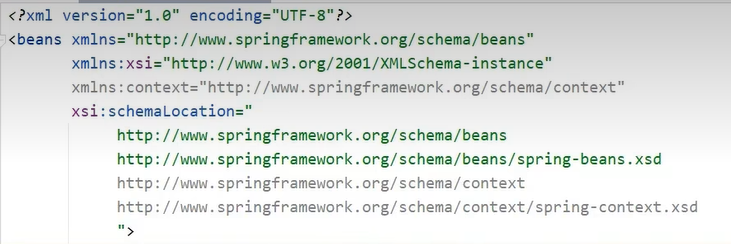


1. 连接数据源的bean，并打印



#### Spring加载properties文件

1. 开创命名空间context



1. 加载properties文件



1. 使用properties，使用为属性占位符${}读取文件属性。

### 注解开发

#### bean的注册

注册bean：@component(“bookbean”)

衍生注解：

表现层的bean @Controller

业务层的bean @Service

数据层的bean @Repository

扫描bean：@componentScan(“bean的路径”)

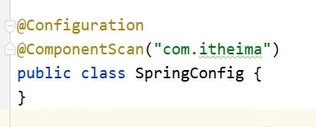
1. 创建一个配置类

用来取代原来的xml文件

@Configuration用于指明这个类是配置类

@ComponeScan(“bean路径”)扫描bean，

bean的路径为bean的类的包名

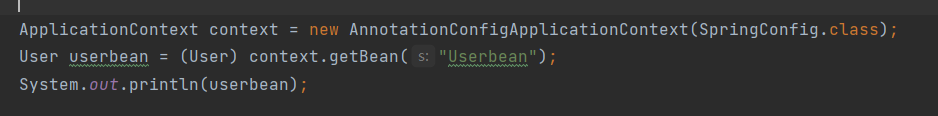


1. 在bean的类上加上注解

@Component("Userbean") ,里面的字符为bean的id

1. 在主函数中

使用AnnotationConfigApplicationContext(配置类.class)



#### bean的作用范围

@Scope(“singleton”) 为单例

@Scope(“prototype”) 为多例

#### bean的生命周期

@PostConstruct 初始方法

@PreDestory 销毁方法

#### 依赖注入

自动装备 @Autowired ，在需要的依赖注入的变量上加上这个注解

@Qualified(“bean名称”)指定注入的bean的名称

自动装配搭配无参构造创建对象

简单类型的注入 @Value(“类型的值”)

name.properties 里面有name=liyanjun

properties的加载：

在配置类中，使用，@PropertySource("name.properties")

之后简单数据类型就可以改成@Value(“${name}”)

#### 第三方的bean管理

1. 创建子配置类，在其中定义一个方法来获取管理的对象
2. 在主配置类使用@Import(子配置类名.class)，多个就用大括号
3. 给这个方法添加@Bean(“bean的名称”)

的注解，表示当前方法的返回值是一个bean。

第三方类的依赖注入

简单类型的使用@Value，引用类型的使用就为自动装配

## Spring-Mybatis

#### 注解开发

1.导入maven坐标

spring-context

druid数据源

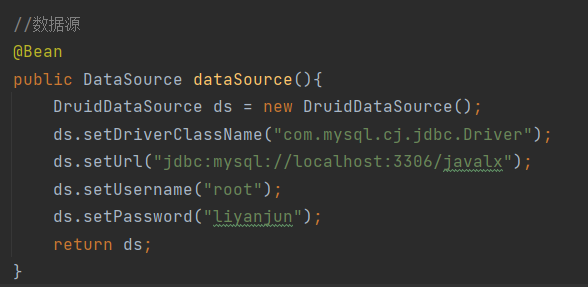
mybatis

mysql-connector-java

spring-jdbc

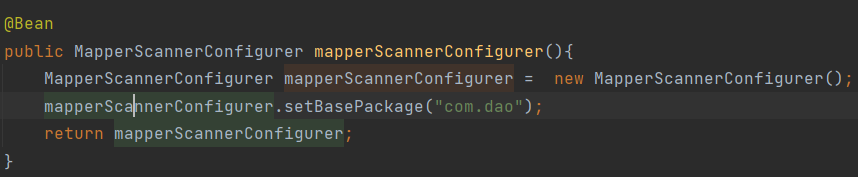
mybatis-spring

1. 将数据源在配置类中配置为bean

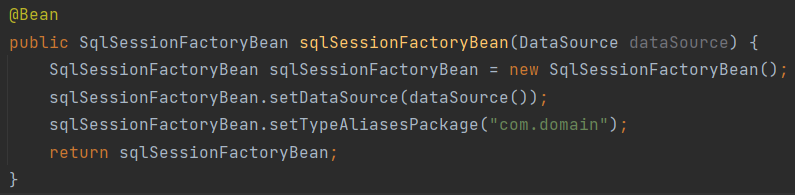


1. 将 SqlSessionFactoryBean在配置类中配置为bean

MapperScannerConfigurer配置为bean



Mapper

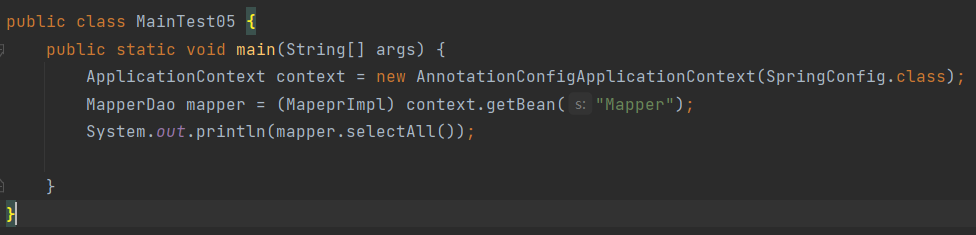


实体类

1. 实现Mapper接口，在里面写Sql语句和对应的方法。

实现Mapper接口的实现类，注意@Resource为自动填装，在Bean中引用其他的类或接口，需要依赖注入，所以@Autowired也可以

1. 在主函数中实现一下



## AOP

AOP思想：在运行时，动态将代码切入到类的指定方法，指定位置上。可以使用AOP来帮助我们在方法执行前或执行之后，做一些额外的操作。（动态代理）

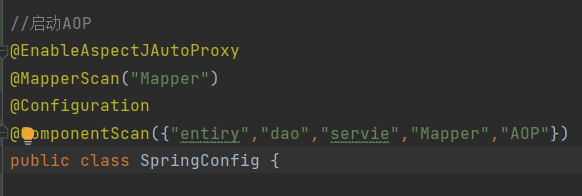
作用：不惊动原始设计的基础上为其进行功能增强。

### 流程

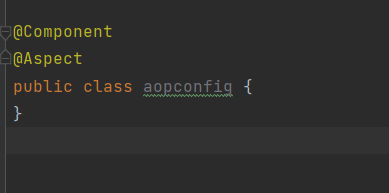
1.使用AOP我们需要先导入一个依赖：

spring-aspects

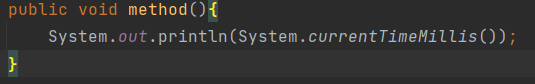
1. 在配置类中添加@EnableAspectJAutoProxy的注解



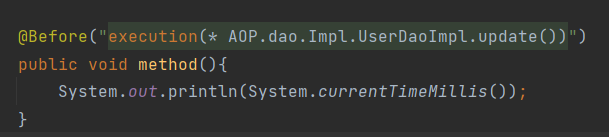
1. 定义一个AOP配置类，添加注解@Aspect和@Component将其注册为Bean



1. 写要加入的方法



5.写AOP操作



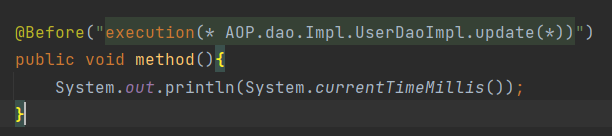
### 切入点表达式

执行AOP.dao.Impl.UserDaoImpl类下的update()方法，\*是方法的参数类型，\*是通配符，..简化包名和参数

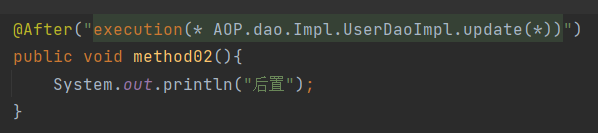
execution(\* AOP.dao.Impl.UserDaoImpl.update())

### 通知类型

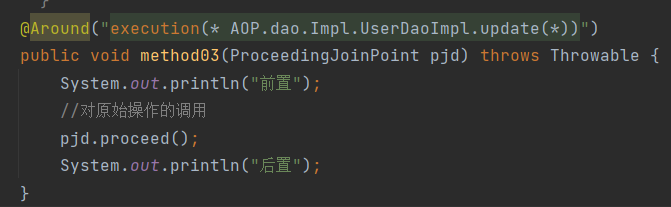
#### 前置通知



#### 后置通知

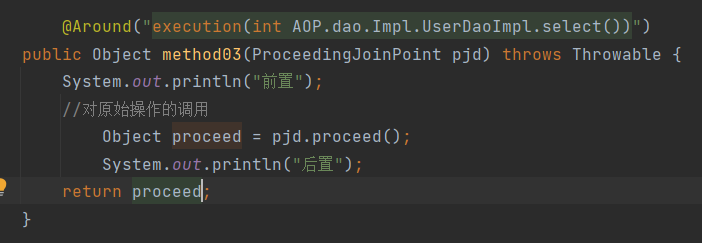


#### ⭐环绕通知

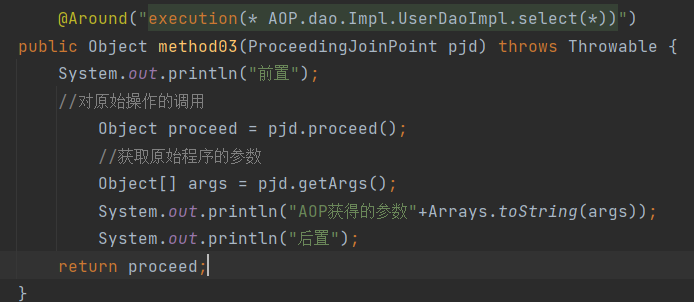


需要使用ProceedingJoinPoint来对原始程序进行调用。

proceedingJoinPoint.proceed();



原始程序带返回值，就在proceed()后调用原始程序的返回值为原始程序的返回值。



获得原始操作的参数,使用getArgs()方法，返回的是一个Object数组。

#### 返回后通知

@AfterReturning，只有方法没有抛异常才执行，和后置一样。

#### 抛出异常后通知

@AfterThrowing，没有事不运行，当有异常才运行。

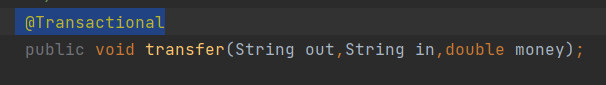
## Spring事务

在数据层或者业务层保障一系列的数据库操作都是成功或者同时失败

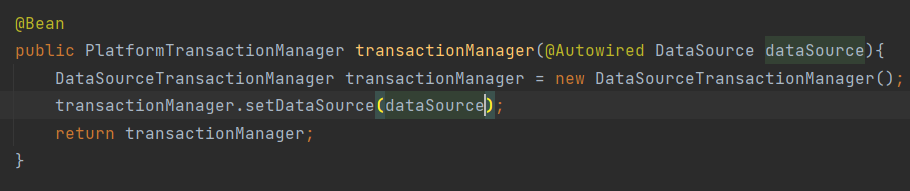
使用PlatformTransactionManager接口

DataSourceTransactionManager实现类

1.在需要事务（业务层接口）的接口的方法上提供 @Transactional注解



1. 在配置类中配置PlatformTransactionManager事务管理器



1. 在配置文件中获取注解@EnableTransactionManagement，告诉Spring我们采用注解来开启事务。

注意@Transactional打到哪个类或者注解上，我们就需要在主类里面实现这个类或者接口

## SpringMVC

SpringMVC是基于MVC模型Web框架技术，表现层的开发工作。

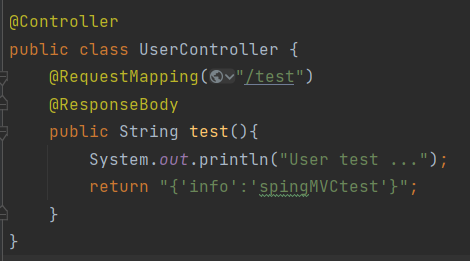
优点：快捷开发

#### 步骤：

1. 导入SpringMVC和servlet坐标

javax.servlet-api和spring-webmvc

1. 创建SpringMVC控制器



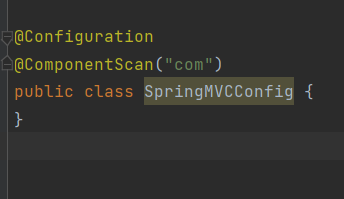
@Controller将其定义为bean（表现层的bean）

@RequestMapping("/test") 定义路径来访问

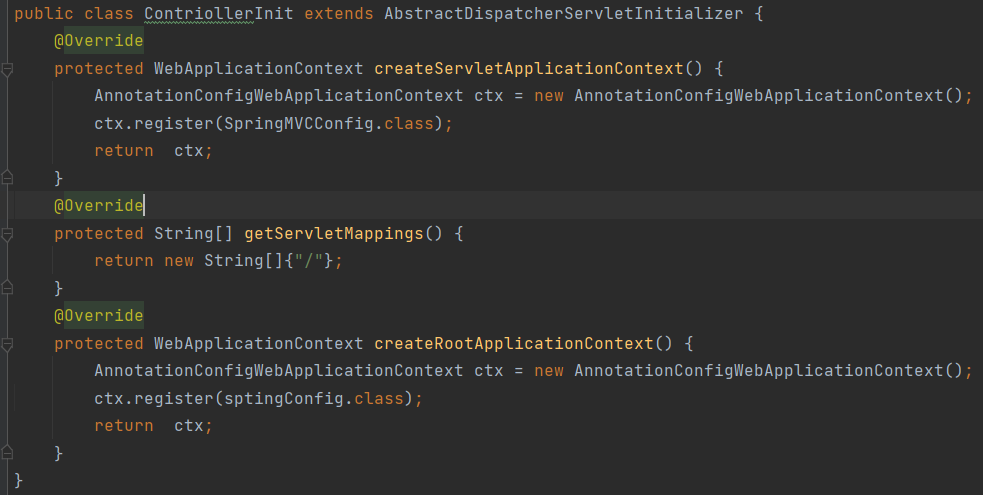
@ResponseBody来写页面的功能

返回的数据，通过retrurn直接返回，如果返回一个json数据，则直接return一个json数据： return "{'info':'spingMVCtest'}";

1. 做一个Spring的配置类



1. 初始化servlet容器，加载SpringMVC环境，并设置SpringMVC技术处理的请求。



创建初始化的类，继承AbstractDispatcherServletInitializer

实现三个方法

createServletApplicationContext是告诉Tomcat容器加载Spring配置

getServletMappings拦截所有的请求

createRootApplicationContext加载Spring配置

也可以采用下面的方式



继承AbstractAnnotationConfigDispatcherServletInitializer

1. 启动Tomcat后，访问

[http://localhost:8080//test路径](http://localhost:8080/test路径)

#### Postman

postman是Chrome插件，用于接口测试。

网页调试，发送Http请求

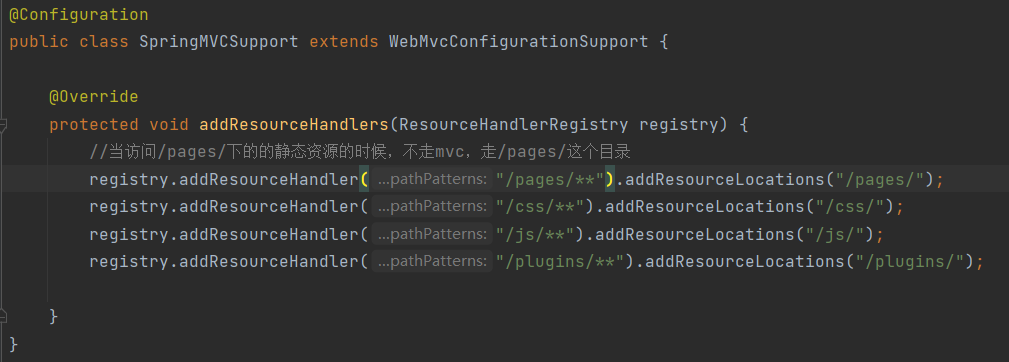
### SpringMVC响应中文

在初始化容器中，添加



### SpringMVC静态资源的访问

1.给SpringMVCConfig增加一个配置类，将需要访问的静态文件放入其中。并注册为bean



2.让SpringMVCConfig配置类扫到这个bean就好了

### 请求和响应

#### 请求映射路径



@RequestMapping,方法注解，类注解，可以添加在SpringMVC控制器的上面，访问方法统一先访问控制器的路径，在访问方法的路径。

#### 请求方式

Get请求和post请求

两种在客户端和服务器端进行请求-响应的常用方法是：GET 和 POST。

GET - 从指定的资源请求数据

POST - 向指定的资源提交要处理的数据

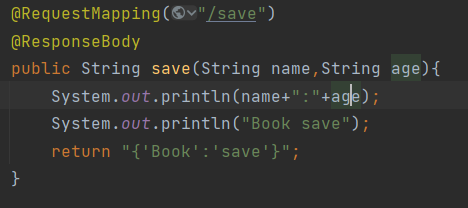
GET 基本上用于从服务器获得（取回）数据。注释：GET 方法可能返回缓存数据。例子：

POST 也可用于从服务器获取数据。不过，POST 方法不会缓存数据，并且常用于连同请求一起发送数据。

##### get请求

http://localhost:8080//Book/save?name=liyanjun&password=123456

这是携带name和password的参数



直接在方法写携带的参数

##### post请求

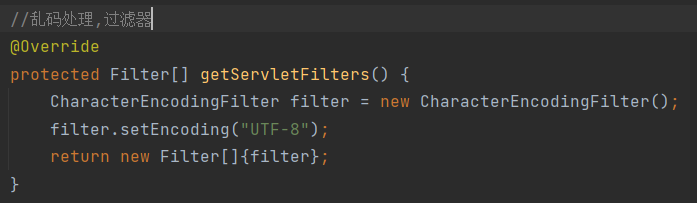
后台不变

发送的表单数据，选择，不仅可以发表单，还可以发文件

请求路径为

[http://localhost:8080//Book/save?](http://localhost:8080/Book/save?)

处理post中文乱码，在初始化的类中，写过滤器

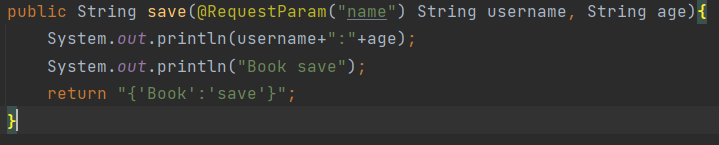


#### 请求参数

##### 普通参数

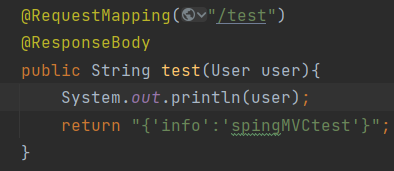
url地址传参，地址参数名和形参参数名相同，定义形参即可接收参数

当url地址参数名和形参名不同时，可以采用@RequestParam("name")绑定请求参数和形参。



##### 实体类参数

url上的属性名和实体类的属性名一样，可以自动将属性值匹配到实体类中

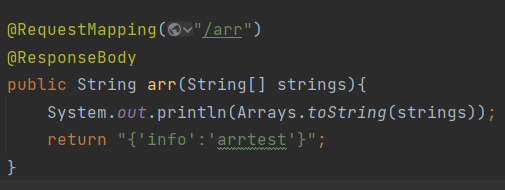


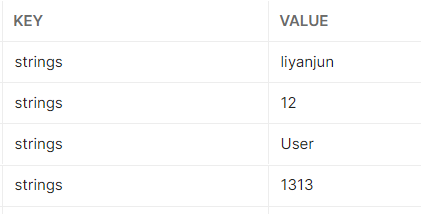
实体类中的引用属性，采用 类名.属性名 的形式传参



##### 数组类型

key值和形参相同，可自动传入

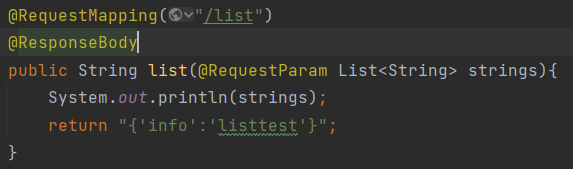


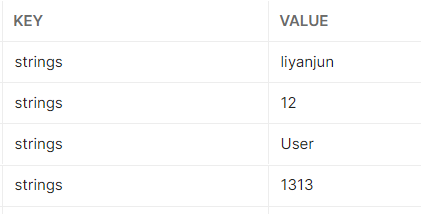


##### 集合类型

集合不能直接传参，自动传参是使用set方法

则使用@RequestParam来接数据，在将数据给到List





##### JSON类型

1.先导入jackson的坐标 jackson-databind，发送json数据请求使用postman中body中的raw的json模式

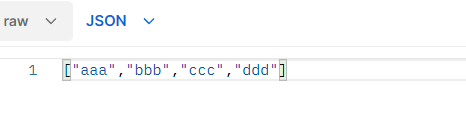
2.在SpringMVC的配置类中开启@EnableWebMvc，开启转换JSON

3.json的数据在请求体中，使用@RequestBody来接收json数据的请求体。

###### json 集合

JSON的数据都在请求体中，使用@RequestBody，获取内容

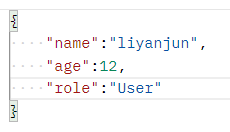




###### json实体类

和上面一样加上@RequestBody，请求的json数据格式方式改变





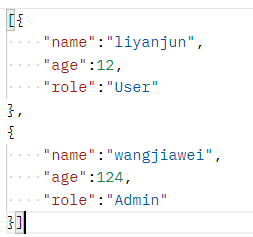
实体类中的引用参数应该为



###### json集合对象

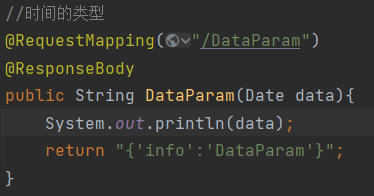
操作和上面一样加上@RequestBody，json数据格式不一样。





##### 日期类型的参数

Date可以接收2012/12/12 类型的日期

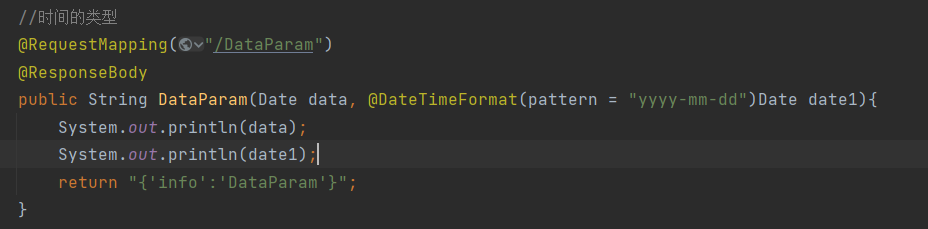




其他的两个类型，需要使用

@DateTimeFormat(pattern = "yyyy/mm/dd HH:mm:ss")

@DateTimeFormat(pattern = "yyyy-mm-dd")来指定



#### 响应

##### 响应页面

###### 页面跳转

第一，和其他的请求响应不同，不需要 @ResponseBody



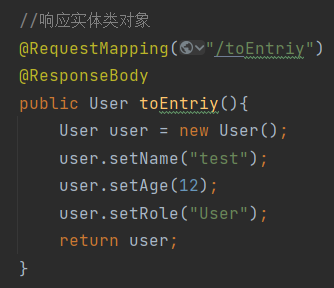
问题：没有办法跳转静态资源（自己创建的html文件，jsp文件可以）

##### 响应数据

###### 响应文本



###### 响应json数据



返回对应的实体类就行。返回的内容就自己转换为json

List一样，只是将返回类型变成List

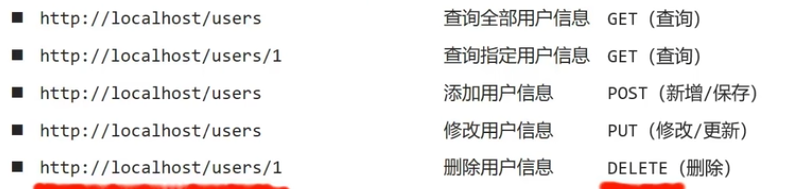
@ResponseBody。作用为设置当前控制器返回值为响应体。

### REST风格的请求路径设置和参数传递

REST表现形式状态转换，访问网络资源的格式不同



请求方式来确定访问行为：RESTful



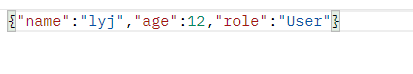
##### REST

查询用GET，新增/保存用POST，修改/更新用PUT，删除用DELETE

###### update风格

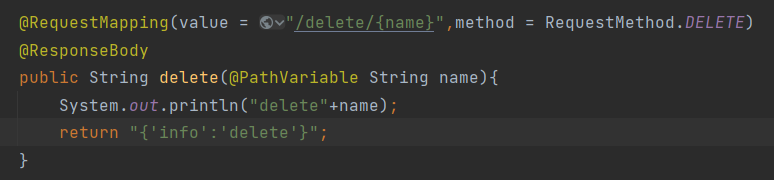
添加method = RequestMethod.PUT

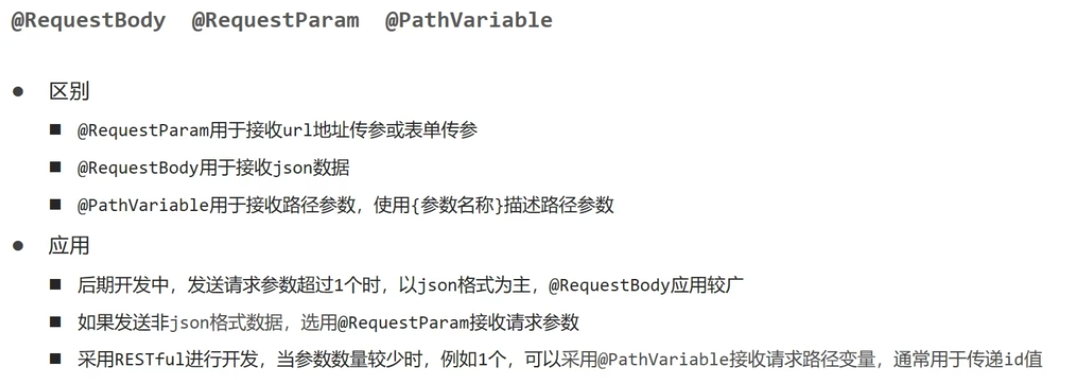




###### delete风格

1. 修改value，传的参数要加上/delete/{name}和方法的参数名一致
2. 添加method = RequestMethod.DELETE
3. 方法的形参添加@PathVariable，告诉springMVC这是路径参数





##### REST快速开发

方法上的@ResponseBody可以不要，写到类的上面

可以和@Controller合体为@RestController

method = RequestMethod.POST可以改成@PostMapping

method = RequestMethod.Delete可以改成@DeleteMapping(“{id}”)

method = RequestMethod.Put可以改成@PutMapping

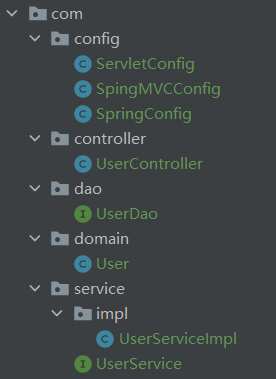
method = RequestMethod.Get可以改成@GetMapping

加载

## 基于SSM整合技术开发任意业务模块



初始化配置

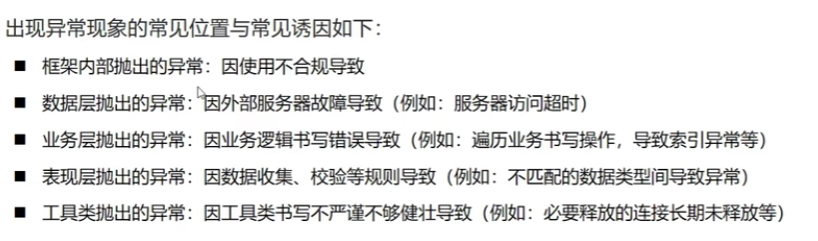


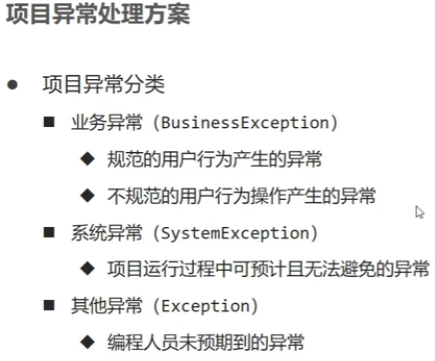
### 表现层和页面的数据协议

表现层数据的封装模型：

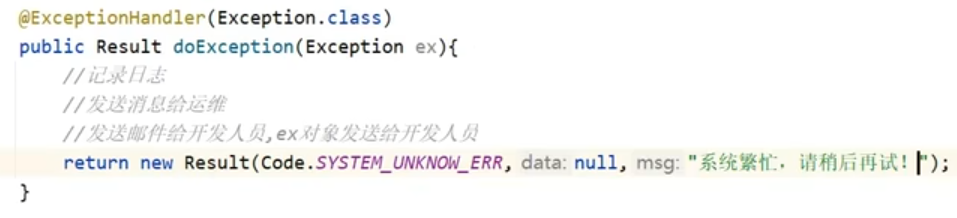


### 处理异常





异常处理器

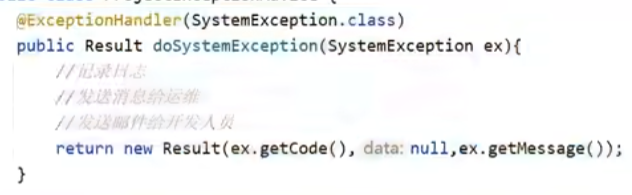


分类异常

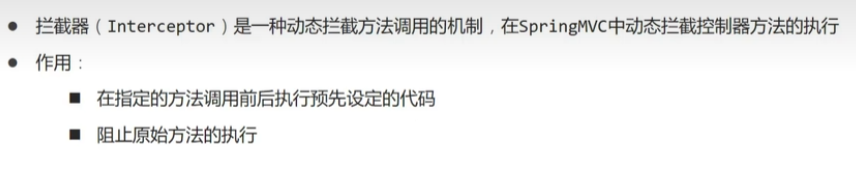




在异常处理器中添加



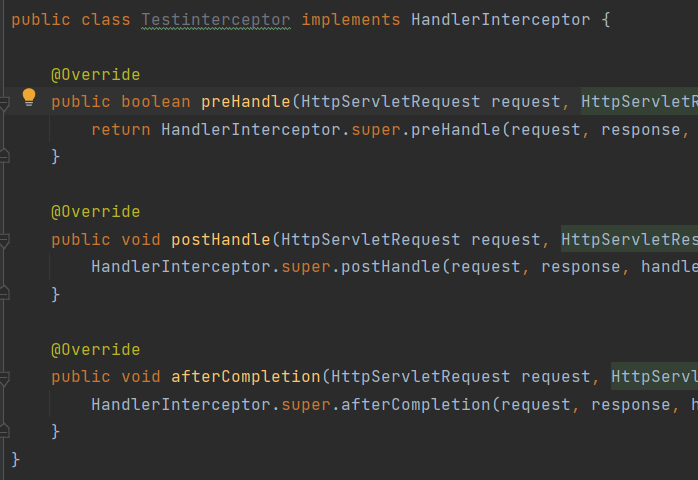
## 拦截器



验证是否登陆、预先设置数据以及统计方法的执行效率之类的。

在controller包中创建拦截器

1. 首先继承 HandlerInterceptor接口，实现3个方法

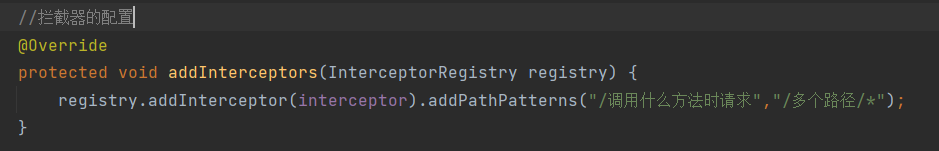


1. 将其设置为bean，在SpringMVC配置类中添加扫描路径

在SpingMVC支持类中重写 addInterceptors 方法。通过依赖注入获取拦截器的对象，以及设定拦截的方法。



1. 多个路径的拦截，使用通配符\*来拦截一个大范围的路径。



## SpringMVC支持类的简化

在SpringMVC配置类上实现WebMvcConfigurer接口。



## 分模块开发（略过）

多个模块之间利用maven坐标引入来使用。

1.新键新的模块，将新的模块的坐标复制到要使用这个模块的pom文件里，新的模块使用install安装到对应目录为jar包。



### maven私服

独立的服务器，用于解决团队内部的资源共享于资源同步问题

Nexus：是一个maven私服产品

下载地址：

https://help.sonatype.com/repomanager3/product-information/download