# 1、Spring

## 1.1、简介

* SSH: Steuct2 + Spring + Hibernate
* SSM : SpringMvc+Spring+Mybatis

<!-- https://mvnrepository.com/artifact/org.springframework/spring-webmvc -->
  
<dependency>
  
 <groupId>org.springframework</groupId>
  
 <artifactId>spring-webmvc</artifactId>
  
 <version>5.2.0.RELEASE</version>
  
</dependency>

官网：https://spring.io/projects/spring-framework#overview

官方下载地址：

GitHub：https://github.com/spring-projects/spring-framework

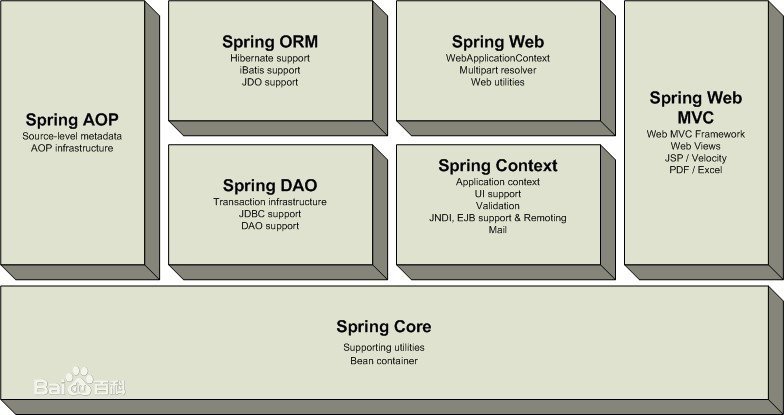
<!-- https://mvnrepository.com/artifact/org.springframework/spring-webmvc -->  
<dependency>  
 <groupId>org.springframework</groupId>  
 <artifactId>spring-webmvc</artifactId>  
 <version>5.2.0.RELEASE</version>  
</dependency>  
<dependency>  
 <groupId>org.springframework</groupId>  
 <artifactId>spring-jdbc</artifactId>  
 <version>5.2.0.RELEASE</version>  
</dependency>

## 1.2、优点

* Spring是一个开源的免费的框架（容器）
* Spring是一个轻量级的、非侵入式的框架！
* 控制反转（IOC）,面向切面编程（AOP）!
* 支持事物的处理，对框架的支持！

总结一句话：Spring就是一个轻量级的控制反转（IOC）和面向切面（AOP）的框架！

## 1.3、组成



## 1.4、扩展



* Spring Boot
  + 一个快速开发的脚手架
  + 基于SpringBoot可以快速的开发单个微服务
  + 约定大于配置
* Spring Cloud
  + SpringClode是基于SpringBoot实现的

现在大多数公司都在使用SpringBoot进行快锁开发，学习SpringBoot的前提，需要完全掌握Spring和SpringMVC！承上启下

# 2、IOC理论推导

1、UserDao接口

2、UserDaoImpl接口实现类

3、UserService业务接口

4、UserServiceImpl业务实现类

在之前的业务中，用户的需求可能会影响我们原来的代码，我们需要根据用户的需求去修改源代码！

我们使用Set接口实现

//private UserDao userDao = new UserDaoImpl();  
 private UserDao userDao;  
 //利用Set进行动态实现值的注入  
  
 public void setUserDao(UserDao userDao) {  
 this.userDao = userDao;  
 }

* 之前，程序是主动创建对象！控制权再程序员手上！
* 使用了set注入之后，程序不在具有主动性，而是变成了

这种思想，从本质上解决了问题，程序员不用再去管理用户的创建了。系统的耦合性大大降低！这是IOC的原型！

### IOC本质

**IOC:控制反转 Inversion of Control，是一种设计思想，DI是实现IoC的一种方法**

应用程序本身不负责依赖对象的创建和维护，而是由外部容器的负责创建和维护

DI:依赖注入 Dependency Injection 是一种实现方式

采用xml方式配置Bean，Bean的定义信息和实现是分离的，而采用注解的方式可以将两者合为一体，

Bean的定义消息直接以注解的形式定义再实现类中，从而达到了零配置的目的。

控制反转是一种通过描述（XML或注解）并通过第三方去生产或获取特定对象的方式。在Spring中实现控制反转的是IoC容器，其实现方法是依赖注入（DI）

# 3、HelloSpring

beans.xml

<?xml version="1.0" encoding="UTF-8"?>  
<beans xmlns="http://www.springframework.org/schema/beans"  
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  
 xsi:schemaLocation="http://www.springframework.org/schema/beans  
 https://www.springframework.org/schema/beans/spring-beans.xsd">  
  
 <bean id="daoImpl" class="com.kuang.dao.UserDaoImpl"/>  
 <bean id="mysqlImpl" class="com.kuang.dao.UserDaoMysqlImpl"/>  
  
 <bean id="UserServiceImpl" class="com.kuang.service.UserServiceImpl">  
<!-- ref:引用Spring容器中创建好的对象  
 value:具体的值，基本类型  
 -->  
 <property name="userDao" ref="mysqlImpl"/>  
 </bean>  
  
 <!-- more bean definitions go here -->  
  
</beans>

java使用bean

public static void main(String[] args) {  
 //获取ApplicationContext  
 ApplicationContext context = new ClassPathXmlApplicationContext("beans.xml");  
  
 //容器  
 UserServiceImpl userServiceImpl = (UserServiceImpl) context.getBean("UserServiceImpl");  
 userServiceImpl.getUser();  
}  
  
//输出获取默认Mysql用户数据

# 4、IOC对象创建的方式

1、使用无参构造方法创建对象。默认事件！

2、使用有参构造创建方法

1、下标赋值

<bean id="hello" class="com.kuang.pojo.Hello">  
<!-- <property name="name" value="wbw"/>-->  
 <constructor-arg index="0" value="kuangs"/>  
 </bean>

2、类型赋值(不建议使用)

<bean id="hello" class="com.kuang.pojo.Hello">  
 <!-- <property name="name" value="wbw"/>-->  
 <constructor-arg type="java.lang.String" value = "123"/>  
</bean>

3、参数名赋值

<bean id="hello" class="com.kuang.pojo.Hello">  
 <!-- <property name="name" value="wbw"/>-->  
 <constructor-arg name="name" value = "123"/>  
 </bean>

总结：在配置文件加载的时候，容器中管理的对象就已经初始化了

# 5、Spring配置

## 5.1、别名

<!-- 别名，如果添加了别名，也可以使用别名-->  
 <alias name="hello" alias="helloNew"/>

## 5.2、Bean的配置

<!--  
 id:bean的唯一标识符，也就是相当于我们学的对象名  
 class:bean对象所对应的全限定名：包名+类型  
 name:也是别名,而且name,可以同时取多个别名  
 -->  
 <bean id = "userT" class="com.kuang.pojo.UserT" name="userNew u2, u3">  
 <constructor-arg name = "name" value="456"/>  
 </bean>

其中name的分隔符，可以是多个

## 5.3、import

这个import，一般用于团队开发使用，他可以将多个配置文件，导入合并为一个

假设，现在项目中有多个人开发，这三个人复制不同的类开发，不同的类需要注册在不同的bean中，我们可以使用import将所有人的beans.xml合并为一个总的！

使用的时候

* applicationContext.xml

<import resource="beans.xml"/>  
<import resource="beans2.xml"/>

使用的时候，直接使用总的配置就可以了

# 6、依赖注入

## 6.1、构造器注入

## 6.2、Set方式注入【重点】

* 依赖注入：Set注入！
  + 依赖：bean对象的创建依赖于容器
  + 注入：bean对象中的所有容器，由容器来注入!

[环境搭建]

1、复杂类型

public class Address {  
 public String address;  
  
 public String getAddress() {  
 return address;  
 }  
  
 public void setAddress(String address) {  
 this.address = address;  
 }  
}

2、真实测试对象

public class Student {  
 private String name;  
 private Address address;  
 private String[] books;  
 private List<String> hobbys;  
 private Map<String, String> card;  
 private Set<String> games;  
 private String wife;  
 private Properties info;  
  
 public String getName() {  
 return name;  
 }  
  
 public void setName(String name) {  
 this.name = name;  
 }  
  
 public Address getAddress() {  
 return address;  
 }  
  
 public void setAddress(Address address) {  
 this.address = address;  
 }  
  
 public String[] getBooks() {  
 return books;  
 }  
  
 public void setBooks(String[] books) {  
 this.books = books;  
 }  
  
 public List<String> getHobbys() {  
 return hobbys;  
 }  
  
 public void setHobbys(List<String> hobbys) {  
 this.hobbys = hobbys;  
 }  
  
 public Map<String, String> getCard() {  
 return card;  
 }  
  
 public void setCard(Map<String, String> card) {  
 this.card = card;  
 }  
  
 public Set<String> getGames() {  
 return games;  
 }  
  
 public void setGames(Set<String> games) {  
 this.games = games;  
 }  
  
 public String getWife() {  
 return wife;  
 }  
  
 public void setWife(String wife) {  
 this.wife = wife;  
 }  
  
 public Properties getInfo() {  
 return info;  
 }  
  
 public void setInfo(Properties info) {  
 this.info = info;  
 }  
}

3、beans.xml

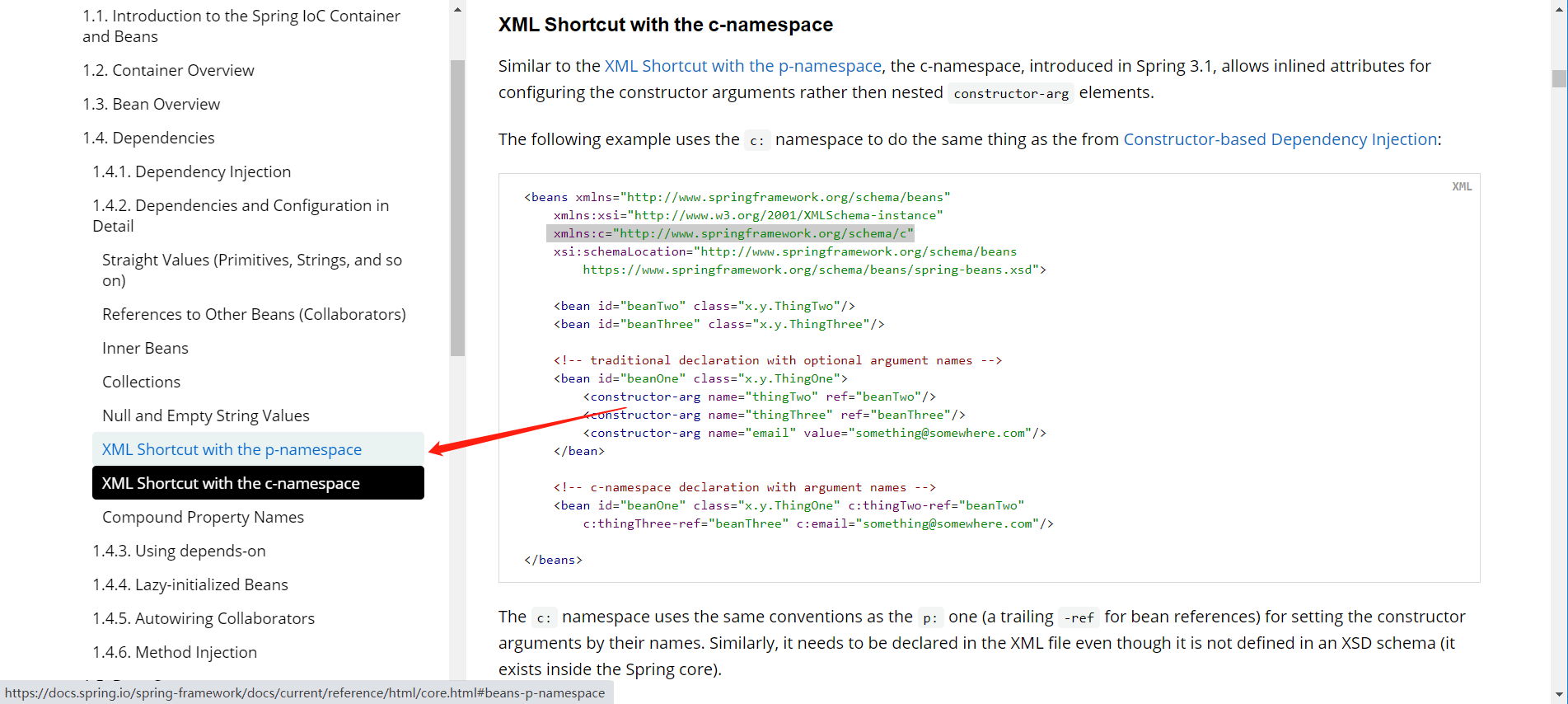
<?xml version="1.0" encoding="UTF-8"?>  
<beans xmlns="http://www.springframework.org/schema/beans"  
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  
 xsi:schemaLocation="http://www.springframework.org/schema/beans  
 https://www.springframework.org/schema/beans/spring-beans.xsd">  
  
  
 <bean id="student" class="com.kuang.pojo.Student">  
 <!--第一种，普通注入，value-->  
 <property name="name" value="wbw"/>  
 </bean>  
</beans>

4、测试类

## 6.3、拓展方式注入

我们可以使用p命名空间和c命名空间进行注入

官方解释



使用！

<?xml version="1.0" encoding="UTF-8"?>  
<beans xmlns="http://www.springframework.org/schema/beans"  
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  
 xmlns:p="http://www.springframework.org/schema/p"  
 xmlns:c="http://www.springframework.org/schema/c"  
 xsi:schemaLocation="http://www.springframework.org/schema/beans  
 https://www.springframework.org/schema/beans/spring-beans.xsd">  
  
<!-- p命名空间注入，可以直接注入属性的值：property-->  
 <bean id="User" class="com.kuang.pojo.User" p:name="金" p:age="19"/>  
 <!-- c命名空间注入，通过构造器注入：property-->  
 <bean id="User2" class="com.kuang.pojo.User" c:age="18" c:name="测试"/>  
  
</beans>

测试:

@Test  
 public void test(){  
 ApplicationContext context = new ClassPathXmlApplicationContext("userbeans.xml");  
 User user2 = context.getBean("User", User.class);  
 System.out.println(user2.toString());  
 }

注意点：p命名和c命名空间不能直接使用，需要导入xml约束！

xmlns:p="http://www.springframework.org/schema/p"  
xmlns:c="http://www.springframework.org/schema/c"

## 6.4、bean的作用域

| Scope | Description |
| --- | --- |
| [singleton](https://docs.spring.io/spring-framework/docs/current/reference/html/core.html#beans-factory-scopes-singleton) | (Default) Scopes a single bean definition to a single object instance for each Spring IoC container. |
| [prototype](https://docs.spring.io/spring-framework/docs/current/reference/html/core.html#beans-factory-scopes-prototype) | Scopes a single bean definition to any number of object instances. |
| [request](https://docs.spring.io/spring-framework/docs/current/reference/html/core.html#beans-factory-scopes-request) | Scopes a single bean definition to the lifecycle of a single HTTP request. That is, each HTTP request has its own instance of a bean created off the back of a single bean definition. Only valid in the context of a web-aware Spring ApplicationContext. |
| [session](https://docs.spring.io/spring-framework/docs/current/reference/html/core.html#beans-factory-scopes-session) | Scopes a single bean definition to the lifecycle of an HTTP Session. Only valid in the context of a web-aware Spring ApplicationContext. |
| [application](https://docs.spring.io/spring-framework/docs/current/reference/html/core.html#beans-factory-scopes-application) | Scopes a single bean definition to the lifecycle of a ServletContext. Only valid in the context of a web-aware Spring ApplicationContext. |
| [websocket](https://docs.spring.io/spring-framework/docs/current/reference/html/web.html#websocket-stomp-websocket-scope) | Scopes a single bean definition to the lifecycle of a WebSocket. Only valid in the context of a web-aware Spring ApplicationContext. |

1、单例模式（Spring默认机制）,单线程使用这个

<bean id="User2" class="com.kuang.pojo.User" c:age="18" c:name="测试" scope="singleton"/>

2、原型模式：每次从容器中get的时候，都会产生一个新的对象 多线程使用这个

<bean id="User2" class="com.kuang.pojo.User" c:age="18" c:name="测试" scope="prototype"/>

3、其余的request、session、application 这些个只能在web开发中使用到！

# 7、Bean的自动装配

* 自动装配是Spring满足bean依赖的一种方式
* Spring会在上下文中自动寻找，并自动给bean装配属性！

在Spring中有三种装配的方式

1、在xml中显示的配置

2、在java中显示配置

3、隐式的自动装配bean【重要】

### 7.1、测试

<bean id="cat" class="com.kuang.pojo.Cat"/>  
 <bean id="dog" class="com.kuang.pojo.Dog"/>  
 <bean id="people" class="com.kuang.pojo.People">  
 <property name="cat" ref="cat"/>  
 <property name="dog" ref="dog"/>  
 <property name="name" value="BinWei"/>  
 </bean>

### 7.2、ByName自动装配

<bean id="cat" class="com.kuang.pojo.Cat"/>  
 <bean id="dog" class="com.kuang.pojo.Dog"/>  
 <bean id="people" class="com.kuang.pojo.People" autowire="byName">  
<!-- <property name="cat" ref="cat"/>-->  
<!-- <property name="dog" ref="dog"/>-->  
 <property name="name" value="BinWei"/>  
 </bean>

PS : 1、byType:会在容器上下文中查找，和自己对象属性类型相同的bean!

2、byName:会在容器上下文中查找，和自己对象set方法后面的值对应的beanid!

### 7.3、ByType自动装配

小结：

* byname的时候，需要保证所有bean的id唯一，并且这个bean需要和自动注入的属性的set方法的值一致！
* bytype的时候，需要保证所有的bean的class唯一，并且这个bean需要和自动注入的属性的set方法的值一致！

### 7.4、使用注解自动装配

jdk1.5支持的注解，Spring2.5支持注解！

要使用注解须知：

1、导入约束:context约束

2、配置注解的支持context:annotation-config/

<?xml version="1.0" encoding="UTF-8"?>  
<beans xmlns="http://www.springframework.org/schema/beans"  
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  
 xmlns:context="http://www.springframework.org/schema/context"  
 xsi:schemaLocation="http://www.springframework.org/schema/beans  
 https://www.springframework.org/schema/beans/spring-beans.xsd  
 http://www.springframework.org/schema/context  
 https://www.springframework.org/schema/context/spring-context.xsd">  
  
 <context:annotation-config/>  
  
  
</beans>

**@Autowired**

直接在属性上用即可！也可以在set方法上使用！

使用Autowired我们可以不用编写Set方法，前提是你这个自动装配的属性在IOC(Spring)容器中存在，且符合名字byname

科普

@Nullable 字段标记了这个注解，说明这个字段可以为null

public class People {  
 //如果显示定义了Autowired的required属性为false，说明这个对象可以为null，否则不允许为空  
 @Autowired(required = false)  
 private Cat cat;  
 @Autowired  
 private Dog dog;  
 private String name;  
}

如果@Autowired自动装配的环境比较复杂，自动装配无法通过一个注解【@Autowired】完成的时候、我们可以使用@Qualifier(value="xxx")去配置@Autowired的使用，指定一个唯一的bean对象的注入！

public class People {  
 @Autowired  
 @Qualifier("cat22")  
 private Cat cat;  
 @Autowired  
 @Qualifier("dog22")  
 private Dog dog;  
 private String name;  
 }

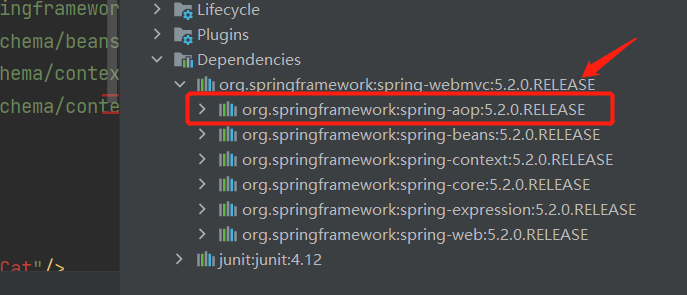
小结：

@Resource和@Autowired的区别

* 都是用来自动装配的，都可以放在属性字段上
* @Autowired通过byType的方式实现
* @Resource默认通过byname 的方式实现，如果找不到名字，则通过byType实现!如果两个都找不到的情况下，就报错！【常用】
* 执行的顺序不同：@Autowired优先byType @Resource优先byName

# 8、使用自动注解开发

再Spring4之后，要使用注解开发，必须保证aop的包导入了



使用注解需要导入context约束

1、bean

2、属性如何注入

@Component  
public class User {  
 //相当于<property name="name" value="123">  
 @Value("123")  
 public String name;  
}

3、衍生的注解

@Component有几个衍生注解，我们web开发中，会按照mvc三层架构分层！

* dao 【@Repository】
* service 【@Serivce】
* controller 【@Controller】
* 这四个注解功能都是一样的，都是代表将某个类注册到Spring中，装配Bean

4、自动装配

- @Autowired:自动装配通过类型。名字  
  
 自动装配无法通过一个注解【@Autowired】完成的时候、我们可以使用@Qualifier(value="xxx")去配置@Autowired的使用，指定一个唯一的bean对象的注入！  
  
- @Nullable：字段标记了这个注解，说明这个字段可以为null  
- @Resource:自动装配通过名字、类型  
- @Component:组件，放在类上，说明这个类被Spring管理了

6、小结

xml与注解：

* xml更加万能，适用于任何场合！维护简单方便
* 注解 不是自己类使用不来，维护相对复杂

最佳实践

* xml用来管理bean
* **注解只负责完成属性的注入**

# 9、使用Java的方式配置Spring

@Configuration代表这是一个配置类，就和我们之前看到beans.xml

纯java的配置方式，在SpringBoot中随处可见

Config

@Configuration //这个也会Spring容器托管，注册到容器中，因为他本来就是一个@Component  
//@Configuration 代表这是一个配置类，就和我们之前看的beans.xml  
@ComponentScan("com.kuang.pojo")  
public class Config {  
 @Bean  
 public User user(){  
 return new User();  
 }  
}

pojo

@Component  
public class User {  
 //相当于<property name="name" value="123">  
 @Value("123")  
 public String name;  
}

main

public class Mytest {  
 public static void main(String[] args) {  
  
 ApplicationContext context = new AnnotationConfigApplicationContext(Config.class);  
 User user = (User) context.getBean("user");  
 System.out.println(user.name);  
 }  
}

# 10、代理模式

为什么要学习代码模式？这是SpringAOP的底层【SpringAOP和SpringMVC】

代理模式的分类：

* 静态代理
* 动态代理

## 10.1、静态代理

角色分析：

* 抽象角色：一般会使用接口或者抽象类来解决
* 真实角色：被代理的角色
* 代理角色：代理真实角色，代理真实角色后，我们一般会做一些附属操作
* 客户：访问代理对象的人！

代码步骤：

1、接口

public interface Rent {  
 public void rent();  
}

2、真实角色

//房东  
public class Host implements Rent {  
 public void rent() {  
 System.out.println("房东要租房子");  
 }  
}

3、代理角色

public class Proxy implements Rent{  
 private Host host;  
  
 public Proxy() {  
 }  
  
 public Proxy(Host host) {  
 this.host = host;  
 }  
 public void rent(){  
 seeHouse();  
 host.rent();  
 fare();  
 }  
 //看房  
 public void seeHouse(){  
 System.out.println("中介带你看房");  
 }  
 //收中介费  
 public void fare(){  
 System.out.println("中介收费");  
 }  
}

4、客服端访问代理角色

public class Client {  
 public static void main(String[] args) {  
 Host host = new Host();  
 //代理，代理角色一般会有附属操作！  
 Proxy proxy = new Proxy(host);  
 proxy.rent();  
 }  
}

代理模式的好处：

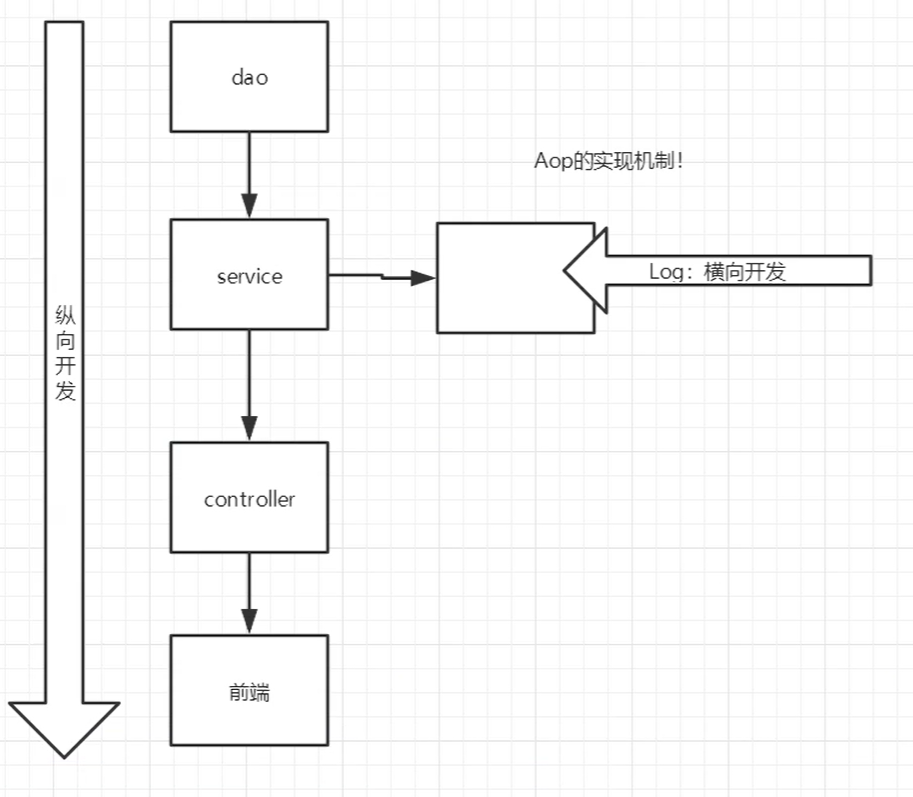
* 可以使真实角色的操作更加纯粹！不用去关注一些公共的业务
* 公共也就交给代理角色！实现业务的分工！
* 公共业务发生扩展的时候，方便集中管理！

缺点：

* 一个真实角色会产生一个代理角色；代理会翻倍

## 10.2、 加深理解

AOP



## 10.3、动态代理

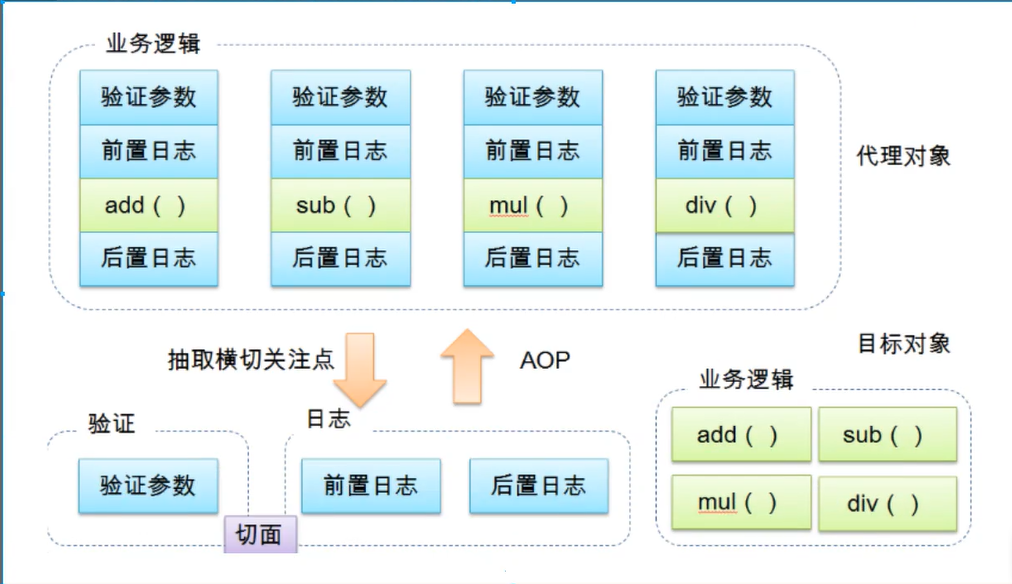
* 动态代理和静态代理角色一样
* 动态代理的代理类是动态生成的，不是我们直接写好的！
* 动态代理分为两大类：基于接口的动态代理，基于类的动态代理
  + 基于接口——JDK动态代理
  + 基于类：cglib
  + java字节码实现：javasist
* 需要了解两个类：Proxy: 代理类，InvocationHandler ： 调用处理程序

//Proxy是生成动态代理类，提供了创建动态代理类和实例的静态方法，它也是由这些方法创建的所有动态代理类的超类。  
//InvocationHandler-- invoke 调用处理程序并返回接口， 是由代理实例的调用处理程序实现的接口 。

动态代理的好处：

* 可以使真实角色的操作更加纯粹！不用去关系一些公共的业务
* 公共也就交给代理角色！实现

# 11、AOP









# 注解说明

* @Autowired:自动装配通过类型。名字
* 自动装配无法通过一个注解【@Autowired】完成的时候、我们可以使用@Qualifier(value="xxx")去配置@Autowired的使用，指定一个唯一的bean对象的注入！
* @Nullable：字段标记了这个注解，说明这个字段可以为null
* @Resource:自动装配通过名字、类型
* @Component:组件，放在类上，说明这个类被Spring管理了
* @Controller 控制器Controller负责处理由DispatcherServlet分发的请求，它把用户请求的数据经过业务处理层处理之后封装成一个Model,将当前修饰的类注入到SpringBoot IOC容器，使得从该类得项目跑起来的过程中，这个类就被实例化。
* @RestController : = @Controller和@ResponseBody
* @ResponseBody 该类中的所有的API接口返回的数据，不管你设置的是Map还是其他Object，它会以Json字符串的形式返回客户端
* @RequestMapping("/user/") 把地址打在/user的地址之下
* @RequestParam:将请求参数绑定到你控制器的方法参数上

# **Bug项目**：

就是pojo项目上有类爆红的时候，但是程序可以与运行的时候，点击Invalidate Caches即可解决

