

Spring

1、Spring简介

- 给软件行业带来了春天。
- Rod Johnson
- interface21框架为基础
- spring理念：使现有的技术更加容易使用，整合了现有的技术框架
- SSH：struct2+Spring+Hibernate
- SSM：SpringMVC+Spring+Mybatis

Spring官网：<https://docs.spring.io/spring-framework/docs/current/reference/html/>

GitHub地址：<https://github.com/spring-projects/spring-framework>

```
1 <!-- https://mvnrepository.com/artifact/org.springframework/spring-webmvc
   -->
2 <dependency>
3     <groupId>org.springframework</groupId>
4     <artifactId>spring-webmvc</artifactId>
5     <version>5.3.3</version>
6 </dependency>
7
8 <dependency>
9     <groupId>org.springframework</groupId>
10    <artifactId>spring-jdbc</artifactId>
11    <version>5.3.3</version>
12 </dependency>
```

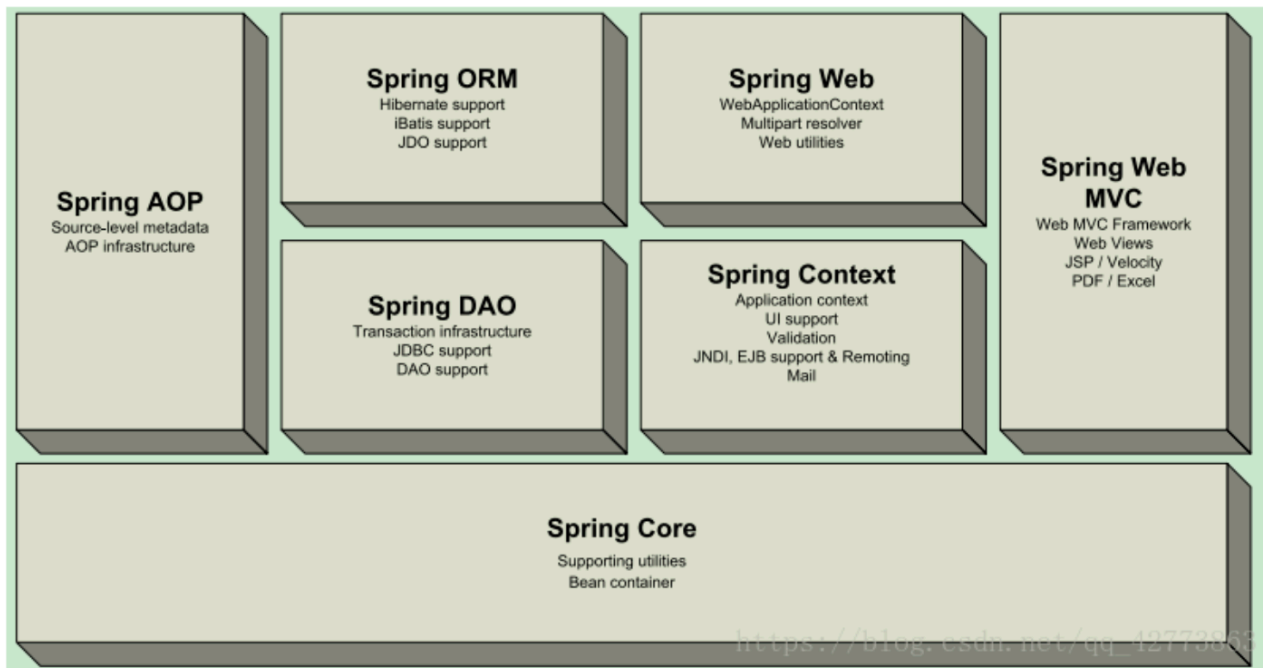
1.1、优点

- Spring是一个开源的免费的框架（容器）！
- Spring是一个轻量级的、非入侵式的框架！
- 控制反转（IOC）、面向切面编程（AOC）
- 支持事务的处理、对框架整合的支持！

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

1.2、组成

（七大组成）



1.3、拓展

- 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
- Spring Cloud
 - SpringCloud是基于SpringBoot实现的

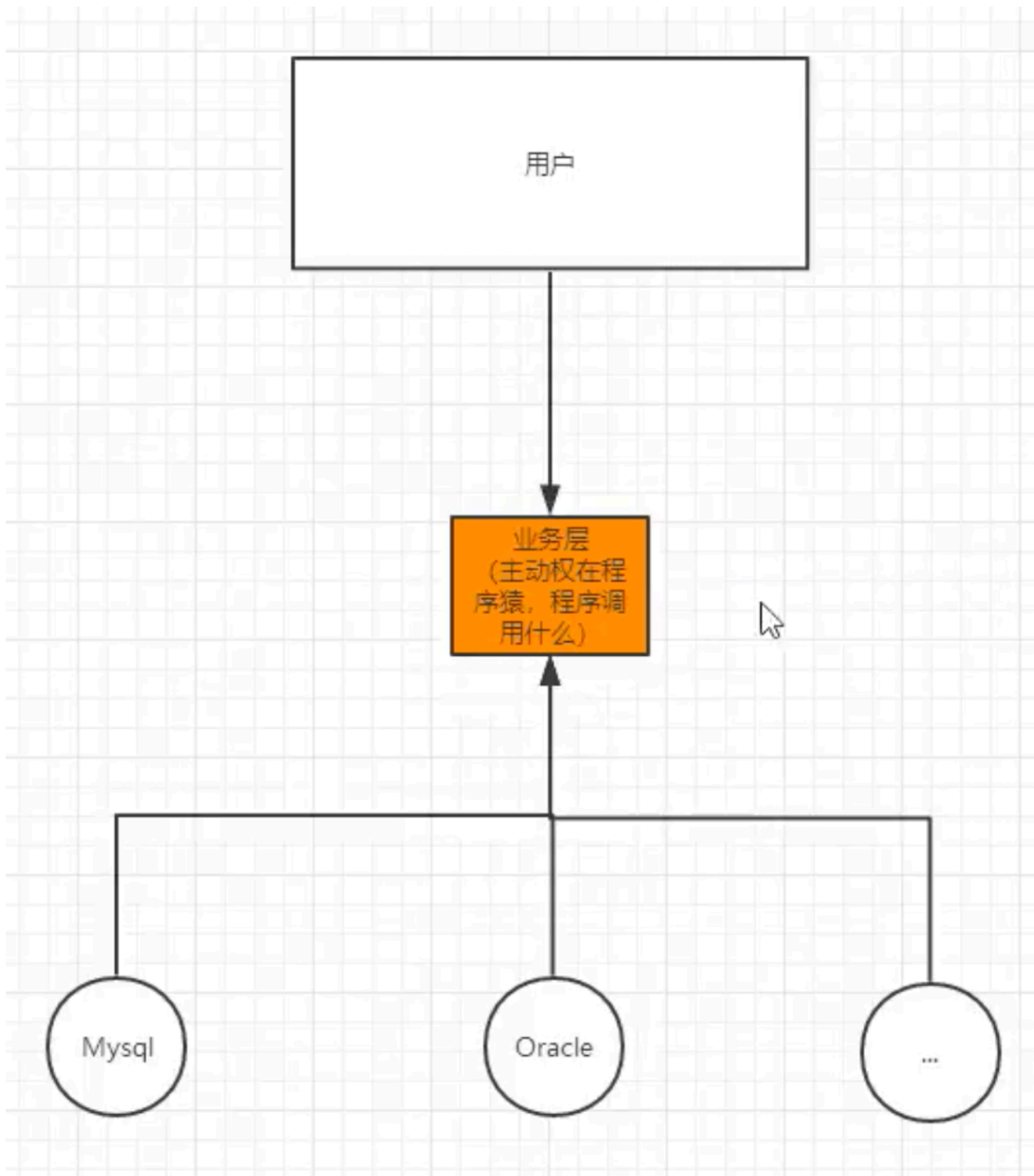
约定大于配置；学习SpringBoot的前提是需要完全掌握Spring和SpringMVC

Spring的弊端：发展太久了之后，违背了原来的理念！配置十分繁琐，人称：“配置地狱！”

2、IOC理论推导

- 1、UserDao接口
- 2、UserDaoImpl实现类
- 3、UserService业务实现类
- 4、UserServiceImpl业务实现类

在我们之前的业务中，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，我们需要根据用户的需求去修改源代码！如果程序代码量十分庞大，修改一次的成本代价十分昂贵！！



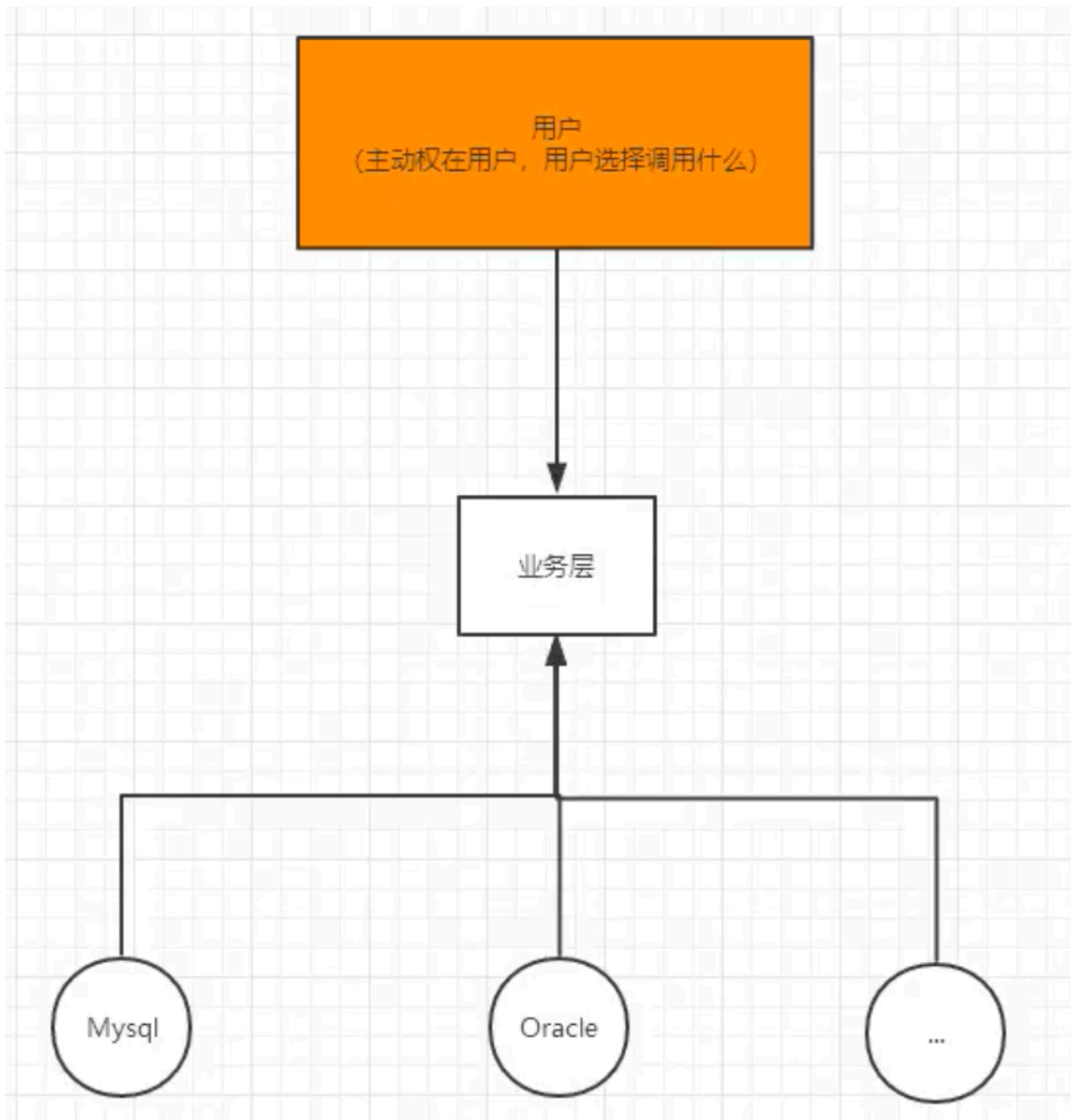
我们使用一个Set接口实现：

```
1 //利用Set进行动态实现值的注入
2 public void setUserDao(UserDao userDao) {
3     this.userDao = userDao;
4 }
```

之前是程序员修改对象代码，现在是用户选择业务对象

- 之前，程序是主动创建对象！控制权在程序员手中！
- 使用了Set注入后，程序不再具有主动性！而是变成了被动的接收对象

这种思想，从本质上解决了问题，我们程序员不用再去管理对象的创建了。系统的耦合性大大降低，可以更加专注的在业务的实现上！这是IOC的原型！



IoC本质：

控制反转IoC(Inversion of Control)，是一种设计思想，DI（依赖注入）是实现IoC的一种方式。

采用xml方式配置Bean的时候，Bean的定义信息是和实现分离的，而采用注解的方式可以把两者合为一体，Bean的定义信息直接以注解的形式定义在实现类中，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（Dependency Injection，DI）。

3、Hello Spring

3.1、编写实体类

```
1 package com.chamfers.pojo;
2
3 public class Hello {
4     private String str;
5
6     public String getStr() {
7         return str;
8     }
9
10    public void setStr(String str) {
11        this.str = str;
12    }
13
14    @Override
15    public String toString() {
16        return "Hello{" +
17            "str='" + str + '\'' +
18            '}';
19    }
20 }
21
```

3.2、编写我们的spring配置文件，这里我们命名为beans.xml

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <beans xmlns="http://www.springframework.org/schema/beans"
3       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4       xsi:schemaLocation="http://www.springframework.org/schema/beans
5         https://www.springframework.org/schema/beans/spring-beans.xsd">
6
7     <!--
8     使用Spring创建对象，在Spring中这些都称为Bean
9     原先创建对象：
10    类型 类型名 = new 类型();
11    Hello hello = new Hello();
12
13    现在：
14    id = 变量名
15    class = new 的对象
16    property 相当于给对象中的属性设置一个值!
17    -->
18    <bean id="hello" class="com.chamfers.pojo.Hello">
19        <!--
20        ref : 引用Spring容器中创建好的对象
21        value : 具体的值，基本数据类型。
22    -->
23    </bean>
24 </beans>
```

```

22         -->
23         <property name="str" value="Hello Spring"/>
24     </bean>
25
26 </beans>

```

3.3、进行测试

```

1  import com.chamfers.pojo.Hello;
2  import org.springframework.context.ApplicationContext;
3  import org.springframework.context.support.ClassPathXmlApplicationContext;
4  /**
5   The location path or paths supplied to an ApplicationContext constructor
6   are resource strings that let the container load configuration metadata
7   from a variety of external resources, such as the local file system, the
8   Java CLASSPATH, and so on.
9   */
10 public class MyTest {
11     public static void main(String[] args) {
12         //解析beans.xml文件，生成管理相应的Bean对象
13         ApplicationContext context = new
14         ClassPathXmlApplicationContext("beans.xml");
15         //getBean():参数即为spring配置文件中bean的id
16         Hello hello = (Hello) context.getBean("hello");
17         System.out.println(hello);
18     }
19 }

```

4、IoC创建对象的方式

- 1、使用无参构造创建对象，默认实现！
- 2、假设我们需要使用有参构造创建对象。

- 下标赋值

```

1  <bean id="exampleBean" class="examples.ExampleBean">
2      <constructor-arg index="0" value="7500000"/>
3      <constructor-arg index="1" value="42"/>
4  </bean>

```

- 类型赋值（不建议使用）

```

1 <!--当两个参数类型相同时不能使用-->
2 <bean id="exampleBean" class="examples.ExampleBean">
3     <constructor-arg type="int" value="7500000"/>
4     <constructor-arg type="java.lang.String" value="42"/>
5 </bean>

```

- 通过参数名

```

1 <!--第三种，直接通过参数名来设置-->
2 <bean id = "user" class = "com.chamfers.pojo.User">
3     <constructor-arg name = "name" value = "chamfers"/>
4 </bean>
5
6 <bean id="exampleBean" class="examples.ExampleBean">
7     <constructor-arg name="years" value="7500000"/>
8     <constructor-arg name="ultimateAnswer" value="42"/>
9 </bean>

```

3、注入bean中的实体类，都被放入IoC容器中了，当容器被创建了（即，ApplicationContext context = new ClassPathXmlApplicationContext("beans.xml");时就已经创建了对象），里面的对象都被创建了。可以设置懒加载，

总计：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

Beans骨架

```

1 <?xml version="1.0" encoding="UTF-8"?>
2 <beans xmlns="http://www.springframework.org/schema/beans"
3     xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4     xsi:schemaLocation="http://www.springframework.org/schema/beans
5         http://www.springframework.org/schema/beans/spring-beans.xsd">
6
7     <bean id="..." class="...">
8         <!-- collaborators and configuration for this bean go here -->
9     </bean>
10
11    <bean id="..." class="...">
12        <!-- collaborators and configuration for this bean go here -->
13    </bean>
14
15    <!-- more bean definitions go here -->
16
17 </beans>

```

5.1、别名

```
1 <!--别名，如果添加了别名，我们也可以使用别名获取到这个对象-->
2 <alias name="hello" alias="helloSpring"/>
```

5.2、Bean的配置

```
1 <!--
2 id: bean的唯一标识符，也就是相当于我们学的对象名
3 class: bean对象所对应的全限定名：包名+类型名
4 name: 也是别名，而且name可以取多个别名，用分隔符",", ";", " " 隔开
5 -->
6 <bean id="hello2" class="com.chamfers.pojo.Hello" name="hello3,h4,h5">
7     <property name="str" value="HelloChamfers"/>
8 </bean>
```

5.3、import

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。

```
1 <beans>
2     <import resource="services.xml"/>
3     <import resource="resources/messageSource.xml"/>
4     <import resource="/resources/themeSource.xml"/>
5
6     <bean id="bean1" class="..." />
7     <bean id="bean2" class="..." />
8 </beans>
```

6、依赖注入(DI)

6.1、构造器注入

上述

```
1 <bean id="exampleBean" class="examples.ExampleBean">
2     <!-- constructor injection using the nested ref element -->
3     <constructor-arg>
4         <ref bean="anotherExampleBean"/>
5     </constructor-arg>
6
7     <!-- constructor injection using the neater ref attribute -->
```

```

8      <constructor-arg ref="yetAnotherBean"/>
9
10     <constructor-arg type="int" value="1"/>
11 </bean>
12
13 <bean id="anotherExampleBean" class="examples.AnotherBean"/>
14 <bean id="yetAnotherBean" class="examples.YetAnotherBean"/>

```

```

1 public class ExampleBean {
2
3     private AnotherBean beanOne;
4
5     private YetAnotherBean beanTwo;
6
7     private int i;
8
9     public ExampleBean(
10         AnotherBean anotherBean, YetAnotherBean yetAnotherBean, int i) {
11         this.beanOne = anotherBean;
12         this.beanTwo = yetAnotherBean;
13         this.i = i;
14     }
15 }

```

6.2、Set方式注入【重点】

```

1 <bean id="exampleBean" class="examples.ExampleBean">
2     <!-- setter injection using the nested ref element -->
3     <property name="beanOne">
4         <ref bean="anotherExampleBean"/>
5     </property>
6
7     <!-- setter injection using the neater ref attribute -->
8     <property name="beanTwo" ref="yetAnotherBean"/>
9     <property name="integerProperty" value="1"/>
10 </bean>
11
12 <bean id="anotherExampleBean" class="examples.AnotherBean"/>
13 <bean id="yetAnotherBean" class="examples.YetAnotherBean"/>

```

```

1 public class ExampleBean {
2
3     private AnotherBean beanOne;
4
5     private YetAnotherBean beanTwo;
6
7     private int i;

```

```

8
9     public void setBeanOne(AnotherBean beanOne) {
10         this.beanOne = beanOne;
11     }
12
13     public void setBeanTwo(YetAnotherBean beanTwo) {
14         this.beanTwo = beanTwo;
15     }
16
17     public void setIntegerProperty(int i) {
18         this.i = i;
19     }
20 }

```

- 依赖注入(DI): Set注入!
 - 依赖: bean对象的创建依赖于容器!
 - 注入: bean对象中的所有属性, 由容器来注入!

【环境搭建】

1、复杂类型

```

1 public class Address {
2     private String address;
3
4     public String getAddress() {
5         return address;
6     }
7
8     public void setAddress(String address) {
9         this.address = address;
10    }
11 }

```

涵盖: bean | ref | idref | list | set | map | props | value | null

2、真实测试对象

```

1 public class Student {
2     private String name;
3     private Address address;
4     private String[] books;
5     private List<String> hobbies;
6     private Map<String,String> cards;
7     private Set<String> games;
8     private String wife;
9     private Properties info;
10 }

```

3、beans.xml

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <beans xmlns="http://www.springframework.org/schema/beans"
3       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4       xsi:schemaLocation="http://www.springframework.org/schema/beans
5       https://www.springframework.org/schema/beans/spring-beans.xsd">
6
7     <bean id="student" class="com.chamfers.pojo.Student">
8         <property name="name" value="陈飞"/>
9     </bean>
10
11 </beans>
```

4、测试类

```
1 public class MyTest {
2     public static void main(String[] args) {
3         ApplicationContext context = new
4         ClassPathXmlApplicationContext("beans.xml");
5         Student student = (Student) context.getBean("student");
6         System.out.println(student.getName());
7     }
8 }
```

完整注入信息

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <beans xmlns="http://www.springframework.org/schema/beans"
3       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4       xsi:schemaLocation="http://www.springframework.org/schema/beans
5       https://www.springframework.org/schema/beans/spring-beans.xsd">
6
7     <bean id="address" class="com.chamfers.pojo.Address">
8         <property name="address" value="浙江"/>
9     </bean>
10
11     <bean id="student" class="com.chamfers.pojo.Student">
12         <!--第一种：普通值注入，value-->
13         <property name="name" value="陈飞"/>
14         <!--第二种：bean注入，ref-->
15         <property name="address" ref="address"/>
16         <!--数组-->
17         <property name="books">
18             <array>
19                 <value>红楼梦</value>
20                 <value>水浒传</value>
21                 <value>三国演义</value>
```

```
22         <value>西游记</value>
23     </array>
24 </property>
25 <!--List-->
26 <property name="hobbies">
27     <list>
28         <value>打游戏</value>
29         <value>看书</value>
30         <value>写代码</value>
31     </list>
32 </property>
33 <!--map-->
34 <property name="cards">
35     <map>
36         <entry key="学生证" value="S20202001002"/>
37         <entry key="身份证" value="111111222222223333"/>
38     </map>
39 </property>
40 <!--Set-->
41 <property name="games">
42     <set>
43         <value>LOL</value>
44         <value>COC</value>
45         <value>BOB</value>
46     </set>
47 </property>
48 <!------>
49 <property name="info">
50     <props>
51         <prop key="url">jdbc://332.1.1.2</prop>
52         <prop key="username">username</prop>
53         <prop key="root">root</prop>
54         <prop key="password">123456</prop>
55     </props>
56 </property>
57 <property name="wife">
58     <null/>
59 </property>
60 </bean>
61 </beans>
```

6.3、拓展方式注入

- p-namespace 【properties-namespace 对应属性注入】

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <beans xmlns="http://www.springframework.org/schema/beans"
3       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4       xmlns:p="http://www.springframework.org/schema/p"
5       xsi:schemaLocation="http://www.springframework.org/schema/beans
6       https://www.springframework.org/schema/beans/spring-beans.xsd">
7
8 <!--p-标签命名空间注入，可以直接注入属性的值： properties-->
9 <bean id="hello6" class="com.chamfers.pojo.Hello" p:age="18" p:name="陈飞"
  p:str="hello 6"/>
```

- c-namespace 【constructor-namespace对应构造器注入】

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <beans xmlns="http://www.springframework.org/schema/beans"
3       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4       xmlns:c="http://www.springframework.org/schema/c"
5       xsi:schemaLocation="http://www.springframework.org/schema/beans
6       https://www.springframework.org/schema/beans/spring-beans.xsd">
7 <!--c-标签命名空间注入，通过构造器注入： constructor-->
8 <bean id="hello7" class="com.chamfers.pojo.Hello" c:age="17" c:name="陈飞"
  c:str="hello7"/>
```

我们可以使用p-namespace和c-namespace进行注入

注意点： p-namespace和c-namespace不能直接使用，需要导入xml约束

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

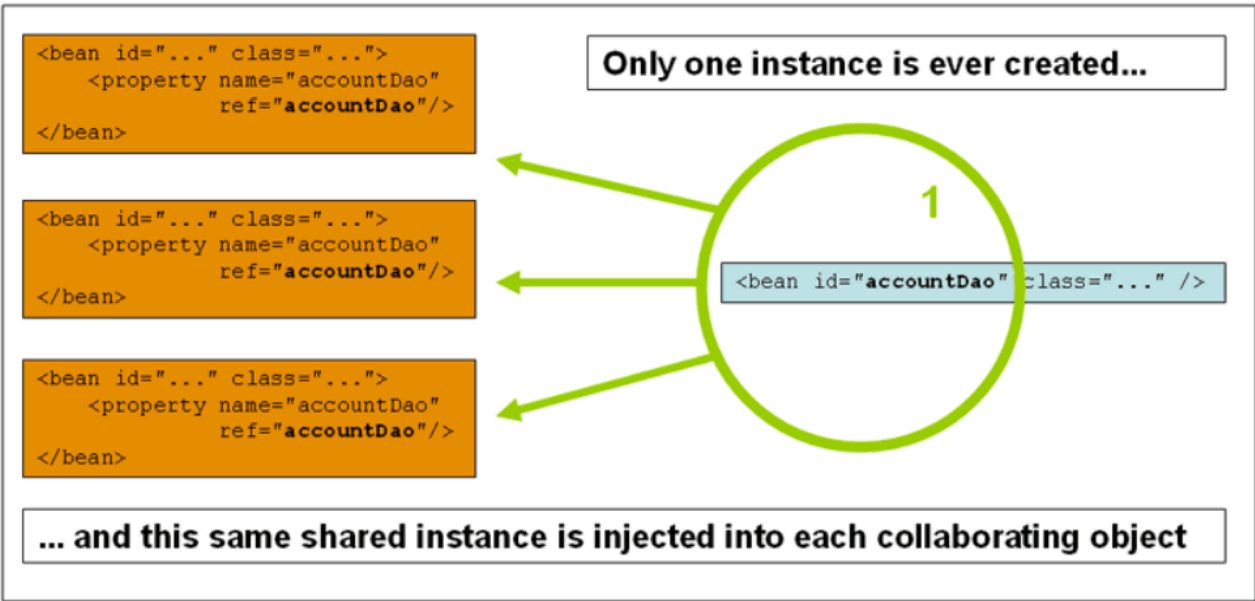
6.4、Bean的作用域

Table 3. Bean scopes

Scope	Description
singleton	(Default) Scopes a single bean definition to a single object instance for each Spring IoC container.
prototype	Scopes a single bean definition to any number of object instances.
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 <code>ApplicationContext</code> .
session	Scopes a single bean definition to the lifecycle of an HTTP <code>Session</code> . Only valid in the context of a web-aware Spring <code>ApplicationContext</code> .
application	Scopes a single bean definition to the lifecycle of a <code>ServletContext</code> . Only valid in the context of a web-aware Spring <code>ApplicationContext</code> .
websocket	Scopes a single bean definition to the lifecycle of a <code>WebSocket</code> . Only valid in the context of a web-aware Spring <code>ApplicationContext</code> .

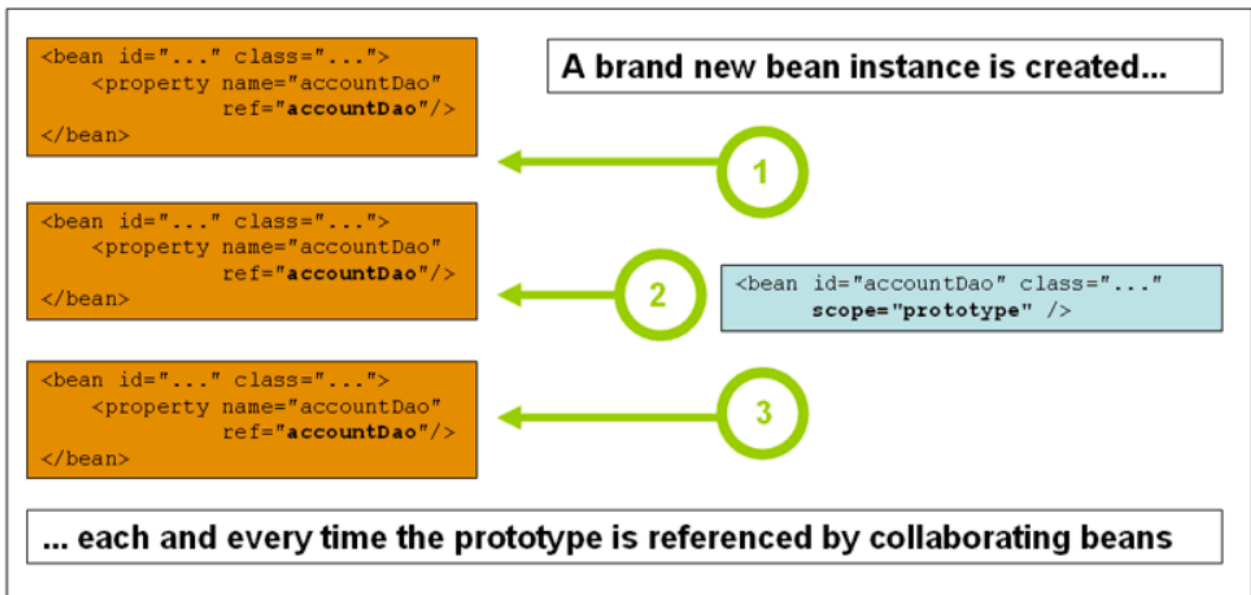
1、单例模式（Spring默认机制）

```
1 <bean id="hello7" class="com.chamfers.pojo.Hello" c:age="17" c:name="陈飞"
  c:str="hello7" scope="singleton"/>
```



2、原型模式:每次从容器中get的时候，都会产生一个新对象！

```
1 <bean id="hello7" class="com.chamfers.pojo.Hello" c:age="17" c:name="陈飞"
  c:str="hello7" scope="prototype"/>
```



3、其余的request、session、application，这些只能在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中显式的配置
- 2、在Java中显式的装配
- 3、隐式的自动装配bean **【重要】**

7.1、测试

搭建环境：一个人有两个宠物！

7.2、自动装配

7.2.1、ByName和ByType自动装配

```

1 <bean id="dog" class="com.chamfers.pojo.Dog">
2     <property name="name" value="小黄" />
3 </bean>
4 <bean id="cat" class="com.chamfers.pojo.Cat" />
5 <!--
6 byName:会自动在容器上下文中去查找, 和自己对象set方法后面的值对应的bean id!
7 byType:会自动在容器上下文中去查找, 和自己对象属性类型相同的bean!
8 -->
9 <bean id="people" class="com.chamfers.pojo.People" autowire="byName">
10     <property name="name" value="陈小飞" />
11 <bean id="people" class="com.chamfers.pojo.People" autowire="byType">
12     <property name="name" value="陈小飞" />
13 </bean>

```

小结:

- byName时, 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方法后的值一致!
- byType时, 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类型一致!

7.3、使用注解实现自动装配

Jdk1.5支持的注解, Spring2.5就支持了注解

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/>支持

```

1 <?xml version="1.0" encoding="UTF-8"?>
2 <beans xmlns="http://www.springframework.org/schema/beans"
3     xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4     xmlns:context="http://www.springframework.org/schema/context"
5     xsi:schemaLocation="http://www.springframework.org/schema/beans
6         https://www.springframework.org/schema/beans/spring-beans.xsd
7         http://www.springframework.org/schema/context
8         https://www.springframework.org/schema/context/spring-
9 context.xsd">
10     <context:annotation-config/>
11
12 </beans>

```

@Autowirde

默认按照byType注入，当有多个同类型时按照byName注入

直接在属性上使用即可！也可以在set方式上使用！

使用Autowired我们可以不编写Set方法了，前提是你的自动装配的属性在IoC（Spring）容器中存在，且符合名字byType

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

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <beans xmlns="http://www.springframework.org/schema/beans"
3       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4       xmlns:context="http://www.springframework.org/schema/context"
5
6       xsi:schemaLocation="http://www.springframework.org/schema/beans
7         https://www.springframework.org/schema/beans/spring-beans.xsd
8         http://www.springframework.org/schema/context
9         https://www.springframework.org/schema/context/spring-
context.xsd">
10
11     <context:annotation-config/>
12     <bean id="dog1" class="com.chamfers.pojo.Dog"/>
13     <bean id="dog2" class="com.chamfers.pojo.Dog"/>
14     <bean id="cat1" class="com.chamfers.pojo.Cat"/>
15     <bean id="cat2" class="com.chamfers.pojo.Cat"/>
16     <!--
17         byName:会自动在容器上下文中去查找，和自己对象set方法后面的值对应的bean
id!
18         byType:会自动在容器上下文中去查找，和自己对象属性类型相同的bean!
19         -->
20     <bean id="people" class="com.chamfers.pojo.People"/>
21 </beans>
```

```
1 public class People {
2     private String name;
3     @Autowired
4     @Qualifier(value = "dog1")
5     private Dog dog;
6     @Autowired
7     @Qualifier(value = "cat1")
8     private Cat cat;
9     @Autowired
10    private Dog dog2;
11 }
```

@Resource

和@Autowired类似

如果无法唯一确定bean，可以用@Resource(name = "xxx")来唯一确定

```
1 public class People {
2     private String name;
3     @Autowired
4     @Qualifier(value = "dog1")
5     private Dog dog;
6
7     @Resource
8     private Cat cat;
9
10    @Resource(name = "dogT")
11    private Dog dog2;
12 }
```

小结

@Autowired和@Resource的区别：

- 都是用来自动装配的，都可以放在属性字段上
- @Autowired是先通过byType的方式实现，如果唯一则注入，否则按照byName注入
- @Resource则是先通过byName的方式注入，如果唯一则注入，否则按照byType注入
- 执行顺序不同

8、使用注解开发

在Spring4之后，要使用注解开发，必须要导入AOP包

使用注解要导入context约束，要导入注解的支持

1、bean

```
1 //等价于 <bean id="user" class="com.chamfers.pojo.User"/>
2 //@Component 组件
3 @Component
4 public class User{
5     public String name;
6     public void setName(String name){
7         this.name = name;
8     }
9 }
```

2、属性如何注入

```
1  @Component
2  public class User{
3      // @Value("陈飞")
4      public String name;
5      //相当于<property name="name" value="陈飞">
6      @Value("陈飞")
7      public void setName(String name){
8          this.name = name;
9      }
10 }
```

3、衍生的注解

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

- dao 【@Repository】
- service 【@Service】
- controller 【@Controller】
- @Component

这四个注解的功能都是一样的，都代表将某个类注册到Spring容器中、装配Bean。

4、自动装配

- @Autowired
- @Qualifier
- @Resource

5、作用域

```
1  @Component
2  @Scope("singleton")
3  public class User{
4      // @Value("陈飞")
5      public String name;
6      //相当于<property name="name" value="陈飞">
7      @Value("陈飞")
8      public void setName(String name){
9          this.name = name;
10     }
11 }
```

6、小结

xml和注解：

- xml：更加万能，适用于任何场合，维护简单方便
- 注解：不是自己的类使用不了，维护相对复杂！

xml与注解的最佳实践：

- xml用来管理bean
- 注解只负责完成属性的注入
- 我们在使用的过程中，只需要注意一个问题：必须让注解生效，就必须要开启注解的支持！

```
1 <!--指定要扫描的包，这个包下的注解就会生效-->
2 <context:component-scan base-package="com.chamfers"/>
3 <context:annotation-config/>
```

9、使用Java的方式配置

我们现在要完全不使用Spring的xml配置，全权交给java来做！

Java Config是Spring的一个子项目，在Spring4之后，他成为了核心功能

1、实体类

经过测试，此处的@Component注解不加也可以注入进bean中

```
1 @Component
2 public class User {
3     private String name;
4
5     public String getName() {
6         return name;
7     }
8     @Value("Chamfers")
9     public void setName(String name) {
10        this.name = name;
11    }
12
13    @Override
14    public String toString() {
15        return "User{" +
16            "name='" + name + '\'' +
17            '}';
18    }
19 }
```

2、配置文件

```

1  @Configuration
2  @ComponentScan(value = "com.chamfers.pojo")
3  public class UserConfig {
4
5      @Bean
6      public User getUser(){
7          return new User();
8      }
9  }

```

3、测试

这种纯Java的配置方式，在SpringBoot中随处可见！

10、代理模式

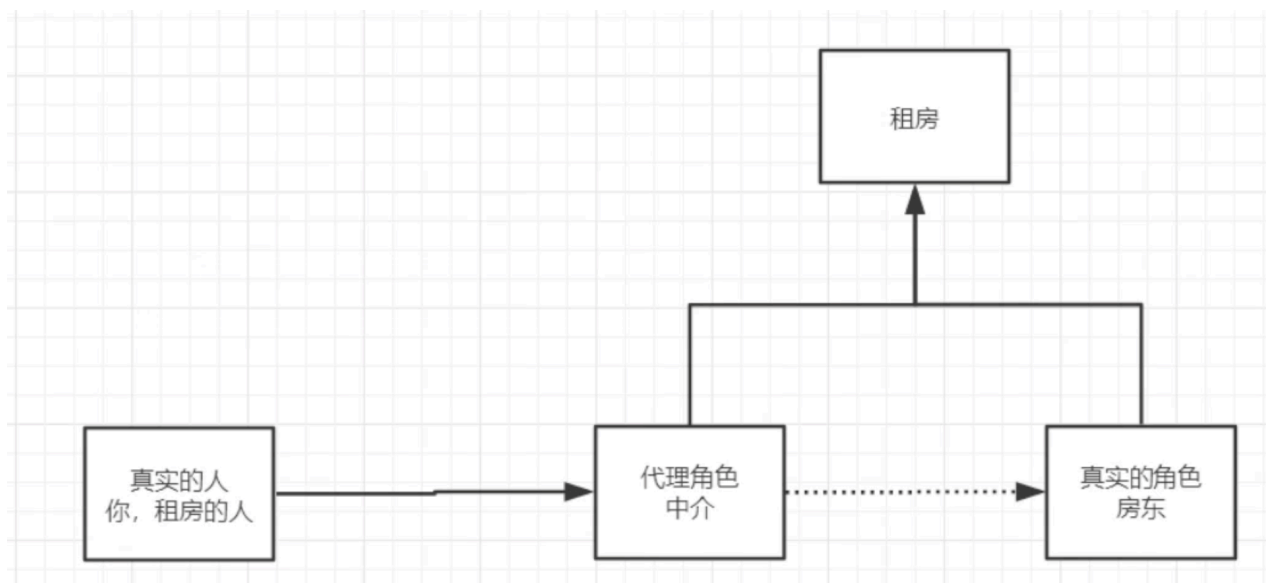
一般是为了给需要实现的方法添加预处理或者添加后续操作，但是不干预实现类的正常业务，把一些基本业务和主要的业务逻辑分离。

为什么要学习代理模式？

因为这就是SpringAOP的底层【SpringAOP和SpringMVC】

代理模式的分类：代理模式就是在不改变原有代码的基础上进行功能增强！

- 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：访问代理的人

代码步骤：

- 1、接口
- 2、真实角色
- 3、代理角色
- 4、客户端访问代理角色

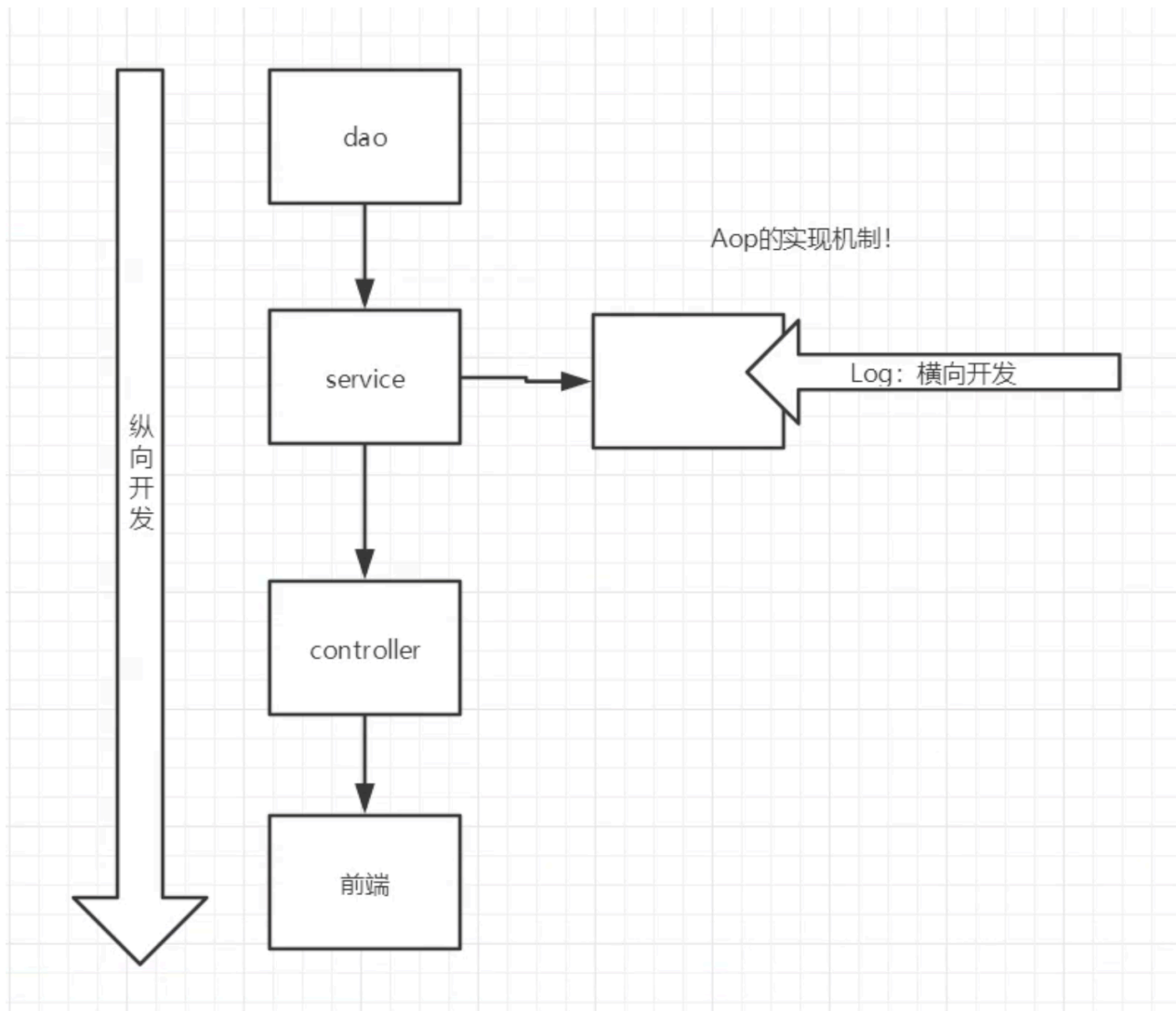
代理模式的好处：

- 可以使真实角色的操作更加纯粹！不用去光洙一些公共业务
- 公共业务就交给代理角色！实现了业务的分工！
- 公共业务发生拓展时，方便集中管理！

缺点：

- 一个真实角色就会产生一个代理角色；代理量会翻倍，开发效率变低！

AOP思想



10.2、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，基于类的代理
 - 基于接口：JDK动态代理【在这里我们使用】
 - 基于类：cglib
 - Java字节码实现：javassist

需要了解两个类：Proxy：接口，InvocationHandler：调用处理程序

InvocationHandler

```
1 //当我们使用这个类时，自动生成代理
2 public class ProxyInvocationHandler implements InvocationHandler {
3     //被代理的接口
4     private Object target;
5     public void setTarget(Object target) {
6         this.target = target;
7     }
8     //生成并获得代理类
```

```

9      public Object getProxy(){
10          return Proxy.newProxyInstance(this.getClass().getClassLoader(),
target.getClass().getInterfaces(), this);
11      }
12      //处理代理实例，返回结果
13      public Object invoke(Object proxy, Method method, Object[] args)
throws Throwable {
14          //在invoke方法中添加要实现的业务
15          log(method.getName());
16          Object invoke = method.invoke(target, args);
17          return invoke;
18      }
19      //增加业务实现
20      public void log(String msg){
21          System.out.println("[log]+"msg);
22      }
23  }

```

动态代理的好处：

- 可以使真实角色的操作更加纯粹！不用去光洙一些公共业务
- 公共业务就交给代理角色！实现了业务的分工！
- 公共业务发生拓展时，方便集中管理！
- 一个动态代理类代理的是一个接口，一般就是对应一类业务！
- 一个动态代理类可以代理多个类，只要是实现了同一个接口即可！

11、AOP

11.1、什么是AOP

方式一：使用Spring的API接口【主要SpringAPI接口实现】

方式二：自定义类来实现AOP【主要是切面定义】

方式三：使用注解实现

12、整合Mybatis

1、传统Mybatis方式

1、配置环境

```

1  <?xml version="1.0" encoding="UTF-8"?>
2  <project xmlns="http://maven.apache.org/POM/4.0.0"
3      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

```

```
4         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/xsd/maven-4.0.0.xsd">
5     <parent>
6         <artifactId>SpringStudy</artifactId>
7         <groupId>org.example</groupId>
8         <version>1.0-SNAPSHOT</version>
9     </parent>
10    <modelVersion>4.0.0</modelVersion>
11
12    <artifactId>spring09-mybatis</artifactId>
13    <dependencies>
14        <dependency>
15            <groupId>mysql</groupId>
16            <artifactId>mysql-connector-java</artifactId>
17            <version>8.0.21</version>
18        </dependency>
19
20        <dependency>
21            <groupId>org.projectlombok</groupId>
22            <artifactId>lombok</artifactId>
23            <version>1.18.16</version>
24        </dependency>
25
26        <dependency>
27            <groupId>junit</groupId>
28            <artifactId>junit</artifactId>
29            <version>4.12</version>
30        </dependency>
31
32        <dependency>
33            <groupId>org.springframework</groupId>
34            <artifactId>spring-jdbc</artifactId>
35            <version>5.2.9.RELEASE</version>
36        </dependency>
37
38        <dependency>
39            <groupId>org.springframework</groupId>
40            <artifactId>spring-webmvc</artifactId>
41            <version>5.2.9.RELEASE</version>
42        </dependency>
43
44        <dependency>
45            <groupId>org.aspectj</groupId>
46            <artifactId>aspectjweaver</artifactId>
47            <version>1.9.5</version>
48        </dependency>
49
50        <dependency>
51            <groupId>org.mybatis</groupId>
```

```

52         <artifactId>mybatis</artifactId>
53         <version>3.5.5</version>
54     </dependency>
55
56     <dependency>
57         <groupId>org.mybatis</groupId>
58         <artifactId>mybatis-spring</artifactId>
59         <version>2.0.5</version>
60     </dependency>
61 </dependencies>
62
63 <!--在build中配置resources，来防止我们资源导出失败的问题-->
64 <build>
65     <resources>
66         <resource>
67             <directory>src/main/resources</directory>
68             <includes>
69                 <include>/**/*.properties</include>
70                 <include>/**/*.xml</include>
71             </includes>
72             <filtering>true</filtering>
73         </resource>
74         <resource>
75             <directory>src/main/java</directory>
76             <includes>
77                 <include>/**/*.properties</include>
78                 <include>/**/*.xml</include>
79             </includes>
80             <filtering>true</filtering>
81         </resource>
82     </resources>
83 </build>
84 </project>

```

2、编写实体类

```

1  @Data
2  public class User {
3      private int id;
4      private String name;
5      private String pwd;
6  }

```

3、编写核心配置文件

```

1  <?xml version="1.0" encoding="UTF-8" ?>
2  <!DOCTYPE configuration

```

```

3      PUBLIC "-//mybatis.org//DTD Config 3.0//EN"
4      "http://mybatis.org/dtd/mybatis-3-config.dtd">
5  <configuration >
6
7      <typeAliases>
8          <typeAlias type="com.chamfers.pojo.User" alias="USer"/>
9      </typeAliases>
10
11     <environments default="environment">
12         <environment id="environment">
13             <transactionManager type="JDBC"></transactionManager>
14             <dataSource type="POOLED">
15                 <property name="driver" value="com.mysql.jdbc.Driver"/>
16                 <property name="url"
17                     value="jdbc:mysql://localhost:3306/mybatis?
useSSL=TRUE&useUnicode=TRUE&characterEncoding=UTF-8"/>
18                 <property name="username" value="root"/>
19                 <property name="password" value="cf19971101"/>
20             </dataSource>
21         </environment>
22     </environments>
23
24     <mappers>
25         <mapper class="com.chamfers.mapper.UserMapper"/>
26     </mappers>
27 </configuration>

```

4、编写mapper.xml

```

1  <?xml version="1.0" encoding="UTF-8" ?>
2  <!DOCTYPE mapper
3      PUBLIC "-//mybatis.org//DTD Config 3.0//EN"
4      "http://mybatis.org/dtd/mybatis-3-mapper.dtd">
5  <mapper namespace="com.chamfers.dao.UserMapper">
6      <select id="getUserList" resultType="User">
7          select * from mybatis.user
8      </select>
9  </mapper>

```

5、测试

```

1  public class MyTest {
2      @Test
3      public void test() throws IOException {
4          String resource = "mybatis-config.xml";
5          InputStream input = Resources.getResourceAsStream(resource);
6

```

```

7      SqlSessionFactory build = new
SqlSessionFactoryBuilder().build(input);
8
9      SqlSession sqlSession = build.openSession(true);
10     UserMapper mapper = sqlSession.getMapper(UserMapper.class);
11
12     List<User> userList = mapper.getUserList();
13     for (User user : userList) {
14         System.out.println(user);
15     }
16 }
17 }

```

2、整合Spring (dependence: spring-mybatis)

1、创建spring-mapper.xml配置文件

```

1  <?xml version="1.0" encoding="UTF-8"?>
2  <beans xmlns="http://www.springframework.org/schema/beans"
3         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4         xsi:schemaLocation="http://www.springframework.org/schema/beans
5             http://www.springframework.org/schema/beans/spring-beans.xsd">
6
7      <bean id="dataSource"
8      class="org.springframework.jdbc.datasource.DriverManagerDataSource">
9          <property name="driverClassName" value="com.mysql.jdbc.Driver"/>
10         <property name="url" value="jdbc:mysql://localhost:3306/mybatis?
11             useSSL=TRUE&amp;useUnicode=TRUE&amp;characterEncoding=UTF-8"/>
12         <property name="username" value="root"/>
13         <property name="password" value="cf19971101"/>
14     </bean>
15
16     <bean id="sqlSessionFactory"
17     class="org.mybatis.spring.SqlSessionFactoryBean">
18         <property name="dataSource" ref="dataSource"/>
19         <property name="mapperLocations"
20             value="classpath:com/chamfers/mapper/*.xml"/>
21         <property name="configLocation" value="mybatis-config.xml"/>
22     </bean>
23
24     <bean id="sqlSession" class="org.mybatis.spring.SqlSessionTemplate">
25         <constructor-arg index="0" ref="sqlSessionFactory"/>
26     </bean>
27
28     <bean id="getMapper" class="com.chamfers.mapper.UserMapperImpl">
29         <property name="sqlSession" ref="sqlSession"/>
30     </bean>
31 </beans>

```

2、修改原先的mybatis-config.xml文件

- 只保留一些<typeAliases>标签

```
1  <?xml version="1.0" encoding="UTF-8" ?>
2  <!DOCTYPE configuration
3      PUBLIC "-//mybatis.org//DTD Config 3.0//EN"
4      "http://mybatis.org/dtd/mybatis-3-config.dtd">
5  <configuration>
6
7      <typeAliases>
8          <typeAlias type="com.chamfers.pojo.User" alias="User"/>
9      </typeAliases>
10
11  <!-- <environments default="environment">-->
12  <!-- <environment id="environment">-->
13  <!-- <transactionManager type="JDBC"></transactionManager>-->
14  <!-- <dataSource type="POOLED">-->
15  <!-- <property name="driver"
16      value="com.mysql.jdbc.Driver"/>-->
17  <!-- <property name="url"-->
18      value="jdbc:mysql://localhost:3306/mybatis?
19      useSSL=TRUE&useUnicode=TRUE&characterEncoding=UTF-8"/>-->
20  <!-- <property name="username" value="root"/>-->
21  <!-- <property name="password" value="cf19971101"/>-->
22  <!-- </dataSource>-->
23  <!-- </environment>-->
24  <!-- </environments>-->
25
26  <!-- <mappers>-->
27  <!-- <mapper class="com.chamfers.mapper.UserMapper"/>-->
28  <!-- </mappers>-->
29  </configuration>
```

3、创建并实现UserMapperLimp类

```
1  public class UserMapperImpl implements UserMapper{
2      private SqlSessionTemplate sqlSession;
3
4      public void setSqlSession(SqlSessionTemplate sqlSession) {
5          this.sqlSession = sqlSession;
6      }
7
8      public List<User> getUserList() {
9          return sqlSession.getMapper(UserMapper.class).getUserList();
10     }
11 }
```

4、修改测试类

```
1 public class MyTest {
2     @Test
3     public void test() throws IOException {
4         //      String resource = "mybatis-config.xml";
5         //      InputStream input = Resources.getResourceAsStream(resource);
6         //
7         //      SqlSessionFactory build = new
8         //      SqlSessionFactoryBuilder().build(input);
9         //
10        //      SqlSession sqlSession = build.openSession(true);
11        //      UserMapper mapper = sqlSession.getMapper(UserMapper.class);
12        //
13        //      List<User> userList = mapper.getUserList();
14        //      for (User user : userList) {
15        //          System.out.println(user);
16        //      }
17        ApplicationContext context = new
18        ClassPathXmlApplicationContext("spring-mapper.xml");
19        UserMapper getMapepr = context.getBean("getMapper",
20        UserMapper.class);
21        List<User> userList = getMapepr.getUserList();
22        for (User user : userList) {
23            System.out.println(user);
24        }
25    }
26 }
```