1、Spring

1.1、简介

- spring：春天 --> 给软件行业带来了春天
- 2002年首次推出了Spring框架的雏形：interface21框架
- 2004年3月24号，Spring框架以interface21框架为基础，经过重新设计，并不断丰富其内涵，发布了1.0正式版
- **Rod Johnson** 很难想象Rod Johnson的学历，真的让好多人大吃一惊，他是悉尼大学的博士，然而他的专业不是计算机，而是音乐学。
- Spring理念：使现有的技术更加容易使用，本身是一个大杂烩，整合了现有的技术框架

- SSH：Struct + Spring + Hibernate
- SSM：SpringMVC + Spring + Mybatis

官网（Spring框架文档）：https://docs.spring.io/spring-framework/docs/current/reference/html/

官方下载地址：https://www.docs4dev.com/docs/zh/spring-framework/5.1.3.RELEASE/reference

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

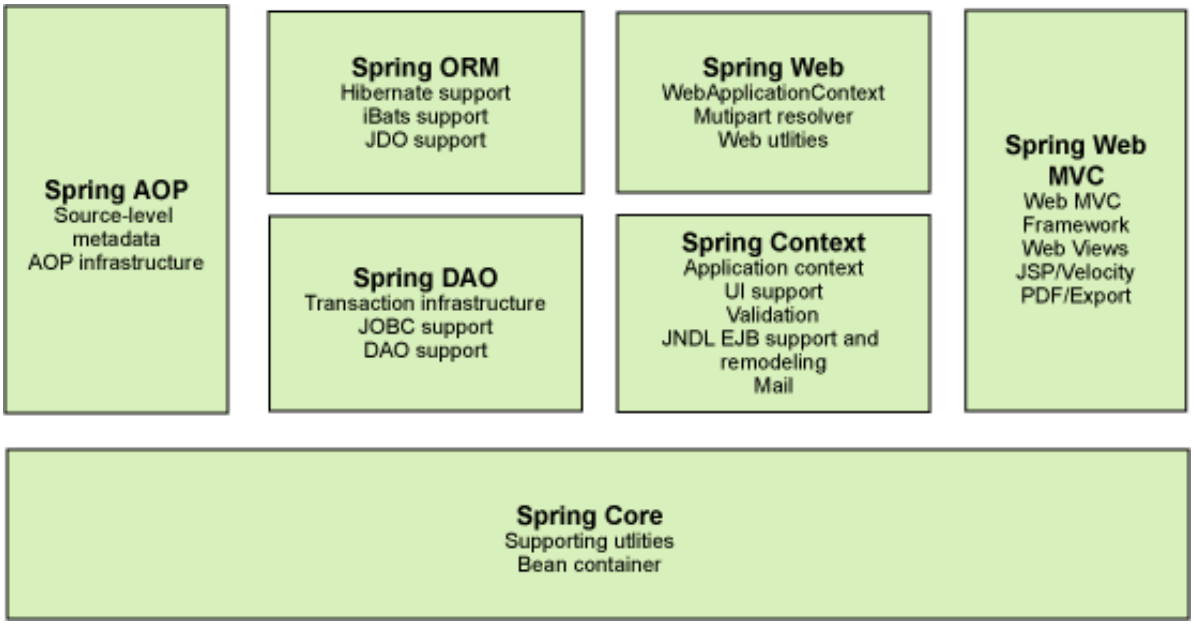
```
<!-- https://mvnrepository.com/artifact/org.springframework/spring-webmvc -->
<dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-webmvc</artifactId>
    <version>5.3.3</version>
</dependency>
<!-- https://mvnrepository.com/artifact/org.springframework/spring-webmvc -->
<dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-jdbc</artifactId>
    <version>5.3.3</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**非入侵**的框架
- **控制反转（IOC）、面向切面编程（AOP）**
- 支持事务的处理，支持对框架的整合

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

1.3、Spring的组成



1.4、拓展

在Spring官网有次介绍：现代化的Java开发说白了就是基于Spring开发

- 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
  - SpringCloud是基于SpringBoot实现的

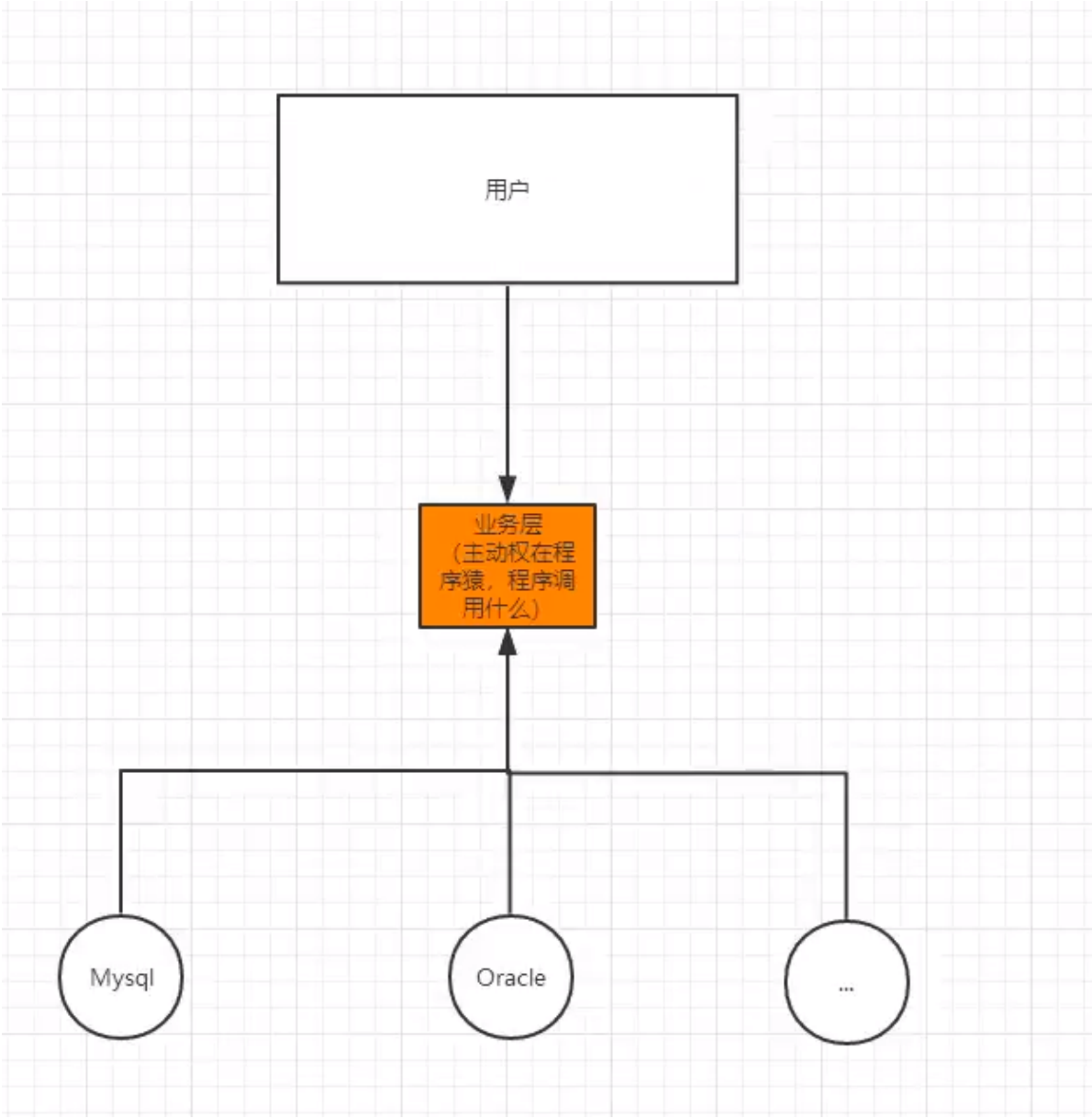
因为现在大多数公司都在使用SpringBoot开发，学习SpringBoot的前提是完全掌握Spring及SpringMVC

Spring起承上启下的作用

Spring的弊端：发展太久后便违背了原来的理念，现在的配置十分繁琐，人称：“配置地狱”

2、IOC理论推导

1. UserDao接口
2. UserImpl
3. UserService 业务接口
4. UserService业务实现类



在之前的业务中，用户的需求可能会影响原来的代码。我们需要根据用户的需求去修改原代码。如果程序代码量很大，则修改一次的代价十分昂贵。

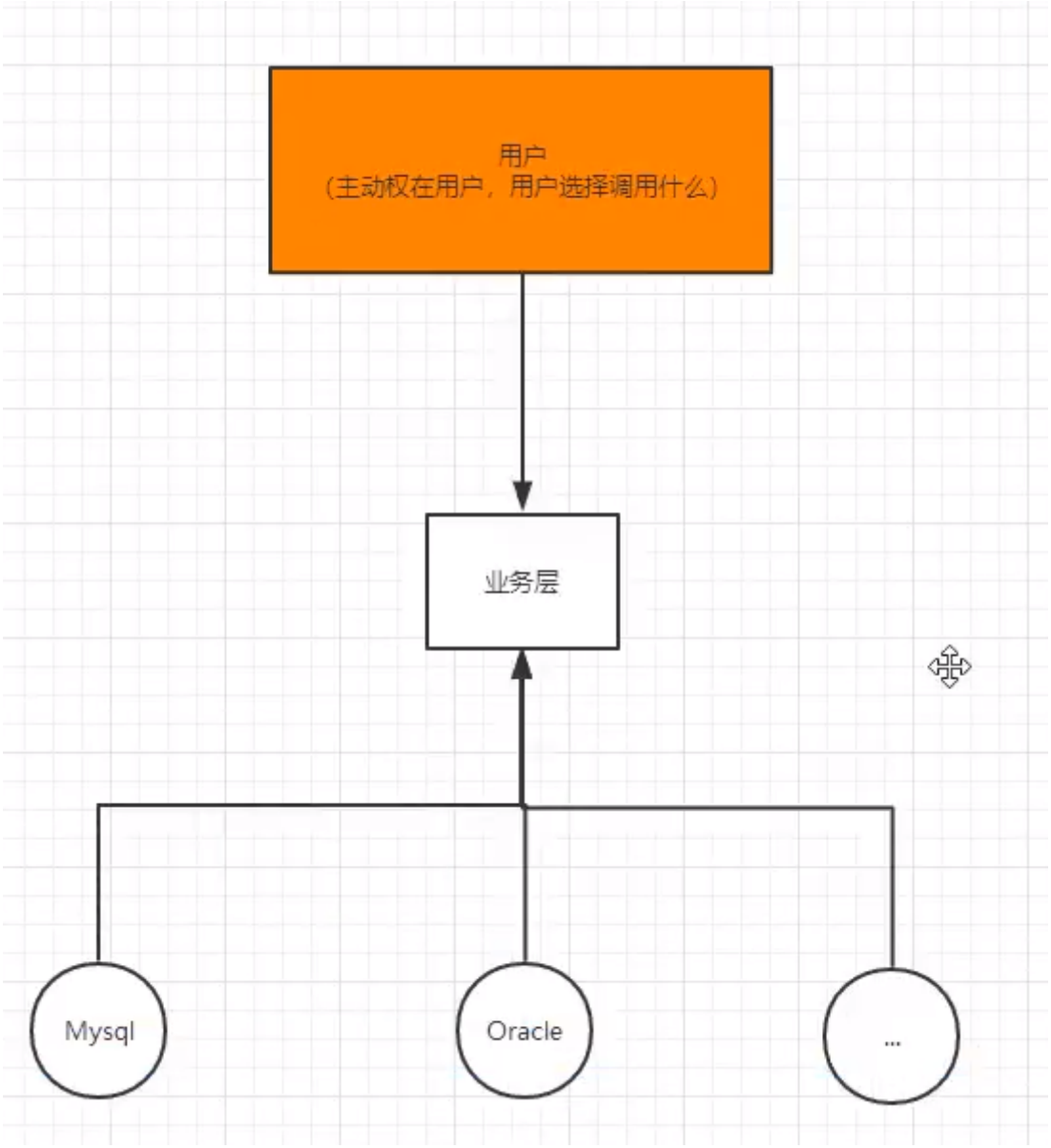
故，我们使用一个Set接口实现。已经发生了革命性的变化。

```
private UserDao userDao;

//利用set动态进行值得注入
public void setUserDao(UserDao userDao){
    this.userDao = userDao;
}
```

- 之前，程序主动创建对象。控制权在程序员受伤
- 使用了set注入后，程序不再具有主动性，而是变成了被动地接收对象

这种思想从本质上解决了问题，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，可以更加专注业务的实现。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的一种方法。**所谓控制反转就是：获得依赖的对象反转了。**

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、HelloSpring

- Hello对象是谁创建的?

  hello对象是由Spring创建的
- Hello对象的属性是怎么设置的?

  hello对象的属性是由spring容器设置的

这个过程就叫**控制反转**

控制：谁来控制对象的创建，传统应用程序的对象是由程序本身控制创建的，使用Spring后，对象是由Spring来创建的。

反转：程序本身不创建对象，而变成被动的接收对象。

依赖注入：就是利用**set方法**来进行注入的。

IOC是一种编程思想，由主动的编程变成被动的接收。

可以通过**newClassPathXmlApplicationContext**去浏览一下底层源码。

到了现在，我们彻底不用在程序中去改动了，要实现不同的操作，只需要在xml配置文件中进行修改，所谓IOC，就是：对象由Spring来创建、管理、装配！

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. 假设我们要使用有参构造创建对象
    1. 下标赋值

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

    2. 通过参数类型赋值（不建议使用）

```
<bean id="exampleBean" class="examples.ExampleBean">
    <constructor-arg type="int" value="7500000"/>
    <constructor-arg type="java.lang.String" value="42"/>
</bean>
```

    3. 直接通过参数名赋值

```
<beans>
    <bean id="beanOne" class="x.y.ThingOne">
        <constructor-arg ref="beanTwo"/>
    </bean>
    <bean id="beanTwo" class="x.y.ThingTwo"/>
</beans>

<!--使用Spring来创建对象，在Spring这些都称为Bean
类型 变量名 = new 类型();
bean = 对象      new Hello();

id = 变量名
class = new 的对象
property 相当于给对象中的属性设置一个值
-->
<bean id="hello" class="pojo.Hello">
    <property name="str" value="Spring~"/>
</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，并且只有一份实例。

5、Spring配置

5.1、别名

```
<!--hello的别名为hello2-->
<alias name="hello" alias="hello2"/>
```

5.2、Bean的配置

```
<!--
id：bean的唯一标识符，相当于对象名
class：bean对象所对应的全限定名（包名：类型）
name：别名，而且name可以同时取多个别名
-->
<bean id="user" class="pojo.Hello" name="hello2,hello3"/>
```

5.3、import

import一般用于团队开发，他可以将多个配置文件导入合并为一个

applicationContext.xml：

```
<import resource="beans.xml"/>
<import resource="beans2.xml"/>
<import resource="beans3.xml"/>
```

假设一个项目有多人开发，每人负责不同的类开发，不同的类需要注册在不同的bean中，我们可以利用import将所有人的be'an.xml合并为一个总的applicationContext.xml。使用时直接使用applicationContext.xml就可以了。

6、依赖注入

6.1、构造器注入

前文已经说过

6.2、通过Set方式注入 ☆

官方文档：https://docs.spring.io/spring-framework/docs/current/reference/html/core.html

- 依赖注入，本质是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

```
<bean id="address" class="pojo.Address">
    <property name="address" value="四川"/>
</bean>


<bean id="student" class="pojo.Student">
    <!--第一种：普通值注入-->
    <property name="name" value="王萌"/>

    <!--第二种，bean注入 ref-->
    <property name="address" ref="address"/>

    <!--数组注入-->
    <property name="books">
        <array>
            <value>红楼梦</value>
            <value>西游记</value>
            <value>三国演义</value>
            <value>水浒传</value>
        </array>
    </property>

    <!--Lsit集合-->
    <property name="hobbies">
        <list>
            <value>睡觉</value>
            <value>看电影</value>
            <value>敲代码</value>
        </list>
    </property>

    <!--Map-->
    <property name="card">
        <map>
            <entry key="身份证" value="11111111"/>
            <entry key="银行卡" value="12465656"/>
        </map>
    </property>

    <!--Set-->
    <property name="games">
        <set>
            <value>lol</value>
            <value>m3</value>
        </set>
    </property>

    <!--null-->
```

```
    <property name="wife">
        <null/>
    </property>

    <property name="info">
        <props>
            <prop key="driver"></prop>
            <prop key="url"></prop>
            <prop key="username">root</prop>
            <prop key="password">7295wangmeng</prop>
        </props>
    </property>
</bean>
```

6.3、拓展方式注入

可以使用p命名空间和c命名空间进行注入：

```
<!--p命名空间注入，可以直接注入属性的值-->
<bean id="user" class="pojo.User" p:name="王萌" p:age="18"/>

<!--c命名空间注入，通过构造器注入：construct-args-->
<bean id="user2" class="pojo.User" c:age="18" c:name="王萌萌"/>
```

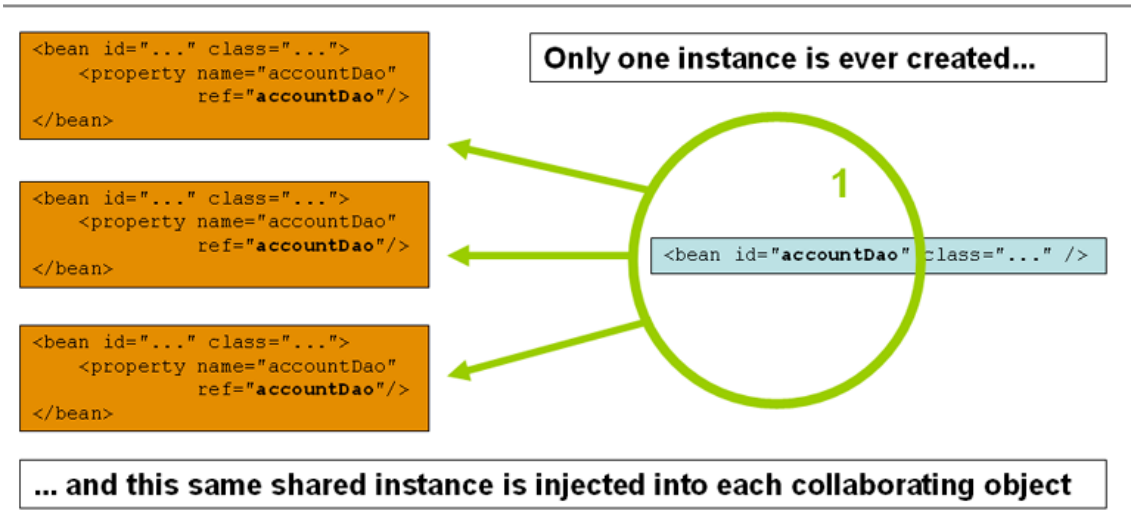
注意点：p命名和c命名空间不能直接使用，需要导入**xml约束**

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

6.4、bean的作用域

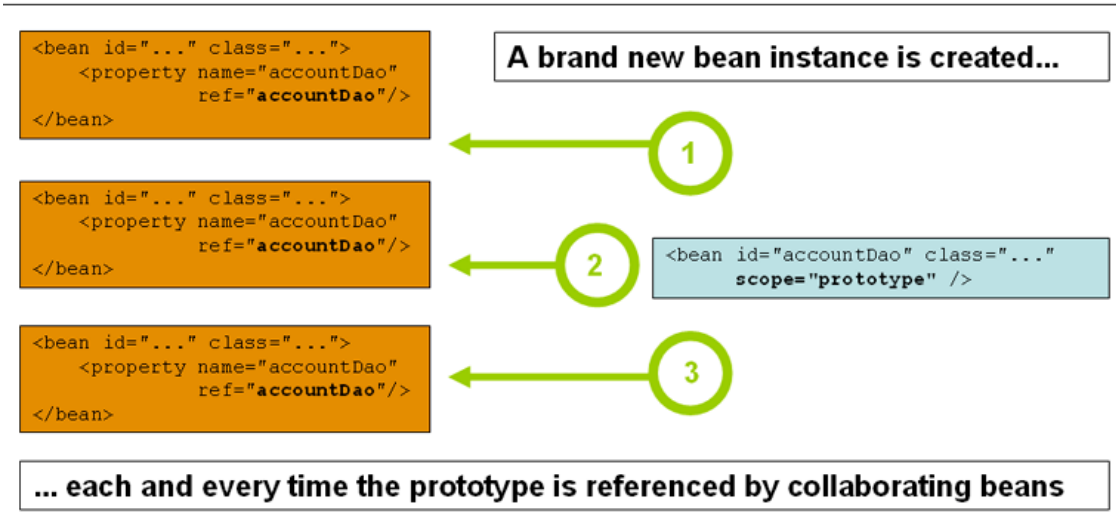
Scope	Description
singleton	(Default) Scopes a single bean definition to a single object instance for each Spring IoC container.
prototype	Scopes a single bean definition to any number of object instances.
request	Scopes a single bean definition to the lifecycle of a single HTTP request. That is, each 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 `ApplicationContext`.
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	Scopes a single bean definition to the lifecycle of a `ServletContext`. Only valid in the context of a web-aware Spring `ApplicationContext`.
websocket	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




```
<bean id="user2" class="pojo.User" c:age="18" c:name="王萌萌"
scope="singleton"/>
```

2. 原型模式（protptype）：每次从容器中get的时候都会产生一个新对象



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

环境搭建：一个人有两个宠物

7.2、ByName自动装配

```
<bean id="cat" class="pojo.Cat"/>
<bean id="dog" class="pojo.Dog"/>
<!--ByName：会自动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-->
<bean id="people" class="pojo.People" autowire="byName">
    <property name="name" value="王萌"/>
</bean>
```

7.3、ByType自动装配

```
<bean id="cat" class="pojo.Cat"/>
<bean id="dog111" class="pojo.Dog"/>

<!--byType：会自动在容器上下文中查找，和自己对象属性类型相同的bean-->
<bean id="people" class="pojo.People" autowire="byType">
    <property name="name" value="王萌"/>
</bean>
```

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bean的class唯一，并且这个bean需要和自动注入的属性的类型一致

7.4、使用注解实现自动装配

要使用注解需知：

1. 导入约束 context约束
2. 配置注解的支持：context:annotation-config/

```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"
    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

直接在属性上使用既可，也可以在set方式上使用

使用Autowired之后就可以不用编写Set方法了，前提是这个自动装配的属性在IOC（Spring）容器中存在，且符合bytype的规则

科普：

```
@Nullable    字段标记了这个注解就说明该字段可以为null
```

```
public @interface Autowired{
    boolean required() default true;
}
```

测试代码：

```
public class People {
    //如果显式定义了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;
}
```

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**@Qualifiler(value="xxx)"**去配合@Autowired的使用，指定一个唯一的bean对象。

```
public class People {
    @Autowired
    @Qualifier(value = "dog222")
    private Dog dog;
}
```

@Resource

```
public class People {
    //如果显式定义了Autowired的required属性为false，说明这个对象可以为null，否则不允许为空
    @Resource(name = "cat1111")
    private Cat cat;
}
```

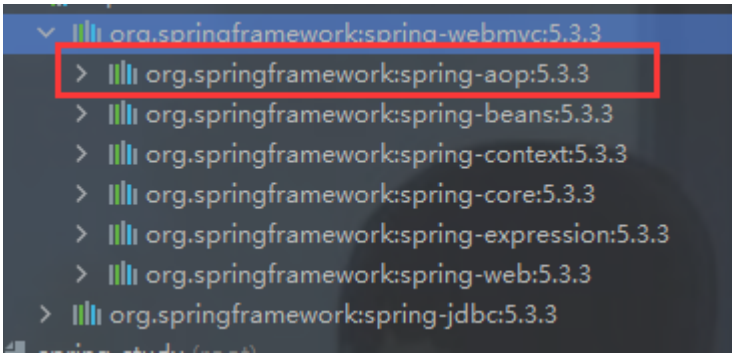
小结：

比较@Resource和@Autowired

- 相同点：都是用来自动装配的，都可以放在属性字段上
- 不同点：
  - @Autowired通过bytype的方式实现（类型重复按名字匹配），而且必须要求这个对象存在【常用】
  - @Resource默认通过byname的方式实现，如果找不到则通过bytype实现。如果两个都找不到才会报错【常用】
  - 执行顺序不同：@Autowired通过byType的方式实现。@Resource默认通过byname的方式实现

8、使用注解开发

在Spring4之后，要使用注解开发，必须要保证aop的包的导入了



使用注解需要导入context约束，增加注解的支持

```
<?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>
```

1. bean

2. 属性如何注入

```
@Component
public class User {
    //相当于<property name="name" value="王萌"/>
    public String name;

    @Value("王萌")
    public void setName(String name) {
        this.name = name;
    }
}
```

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 【@Repositoryr】
   - service 【@Service】
   - 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容器中，装配bean

4. 自动装配

```
-@Autowired：自动装配通过类型，名字
-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
```

5. 作用域

```
@Component
@Scope("prototype")
public class User {
    //相当于<property name="name" value="王萌"/>
    public String name;

    @Value("王萌")
    public void setName(String name) {
        this.name = name;
    }
}
```

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
   - 注解不是自己的类使用不了，维护相对复杂

   xml与注解最佳实践：

   - 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：要让注解生效，就必须开启注解的支持

```
<!--指定要扫描的包，这个包下面的注解就会生效-->
<context:component-scan base-package="Meng"/>
<context:annotation-config/>
```

9、使用java的方式配置spring

完全不使用spring的xml配置，全部由java来实现

JavaConfig是Spring的一个子项目，在Spring4之后，它成了一个核心功能

实体类：

```
//说明这个类被spring接管了，注册到了容器中
@Component
public class User {
    private String name;

    public String getName() {
        return name;
    }

    @Value("萌萌")//属性注入值
    public void setName(String name) {
        this.name = name;
    }

    @Override
    public String toString() {
        return "User{" +
                "name='" + name + '\'' +
                '}';
    }
}
```

配置类：

```
//这个也会被spring容器托管，注册到容器中，因为它本来就是一个@Component
//@Configuration代表这是一个配置类，和bean.xml相同
@Configuration
@ComponentScan("pojo")
@Import(MyConfig2.class)
public class MyConfig {
    //注册一个bean，就相当于之前写的bean标签
    //方法名就为bean标签中的id，返回值就为bean标签中的class属性
    @Bean
    public User getUser(){
        return new User();//返回要注入到bean的对象
    }
}
```

测试类：

```
public class MyTest {
    public static void main(String[] args) {
        //如果完全使用了配置类，我们就只能通过AnnotationConfig上下文来获取容器，通过配置类的
class对象加载
        ApplicationContext context = new
AnnotationConfigApplicationContext(MyConfig.class);
        User getUser = (User) context.getBean("user");
        System.out.println(getUser.getName());
    }
}
```

这种纯java的配置方式，在SpringBoot中随处可见