Spring5框架高级应用

Lecture:波波老师

Spring官网： https://spring.io/

SpringFramework的核心

> IoC[DI]
>
> AOP

一、IoC控制反转

IoC（Inversion of Controller）翻译过来'**控制反转**'

IoC本质上是一个概念，是一种思想，控制反转就是对对象控制权的转移，SpringIoC容器创建对象，然后将对象的使用权交出去

SpringFramework相关Jar地址： https://repo.spring.io/libs-release-local/org/springframework/spring/

二、基于XML配置文件方式的使用

1.基本使用

通过maven相关构建

1.1 添加依赖

需要引入Spring相关的依赖

```
<dependencies>
    <dependency>
        <groupId>org.springframework</groupId>
        <artifactId>spring-context</artifactId>
        <version>5.1.17.RELEASE</version>
    </dependency>

    <!-- https://mvnrepository.com/artifact/junit/junit -->
    <dependency>
        <groupId>junit</groupId>
        <artifactId>junit</artifactId>
        <version>4.12</version>
        <scope>test</scope>
    </dependency>
```

```
</dependencies>
```

1.2 添加Spring的配置文件

```
<?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
http://www.springframework.org/schema/beans/spring-beans.xsd">


</beans>
```

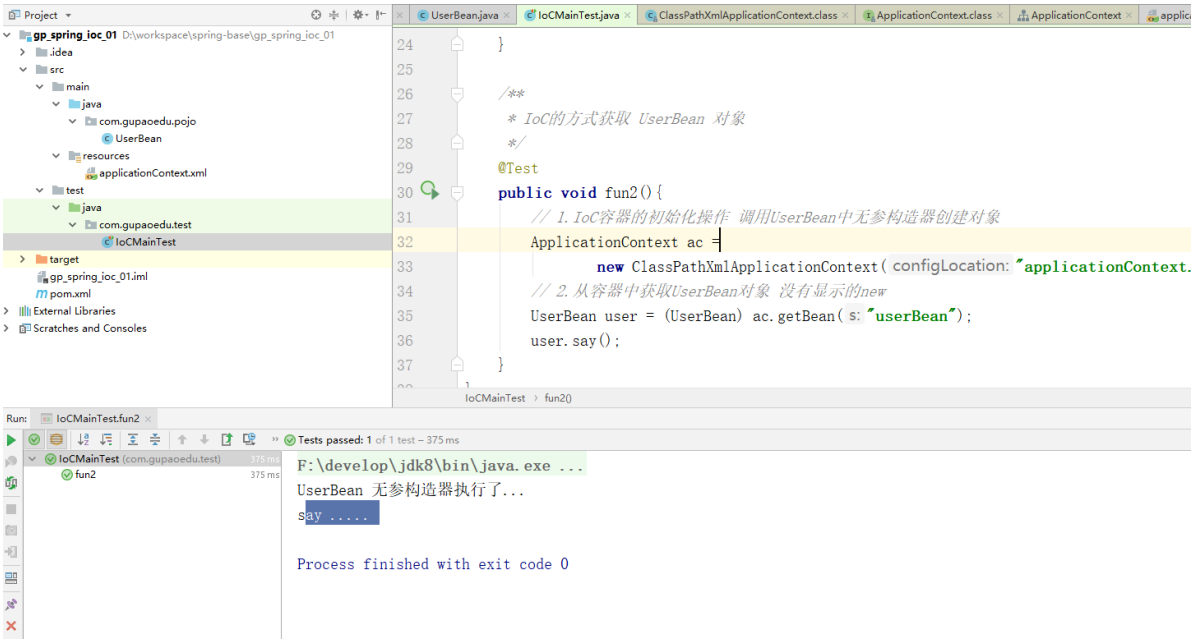
1.3 注册Bean

将需要被IoC容器管理的类型通过标签来注册

```
<?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
http://www.springframework.org/schema/beans/spring-beans.xsd">

    <!--  添加需要被容器管理的内容-->
    <bean class="com.gupaoedu.pojo.UserBean" />
</beans>
```

1.4 测试获取

```
/**
     * IoC的方式获取 UserBean 对象
     */
@Test
public void fun2(){
    // 1.IoC容器的初始化操作 调用UserBean中无参构造器创建对象
    ApplicationContext ac =
        new ClassPathXmlApplicationContext("applicationContext.xml");
    // 2.从容器中获取UserBean对象 没有显示的new
    UserBean user = (UserBean) ac.getBean("userBean");
    user.say();
}
```



2.从容器中获取对象的方式

2.1 根据ID

id只能声明一个

2.2 根据name

可以声明一个或者多个

```
<?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
http://www.springframework.org/schema/beans/spring-beans.xsd">

    <!--  添加需要被容器管理的内容-->
    <!--<bean class="com.gupaoedu.pojo.UserBean" id="userBean"
name="userBean2"/>-->
    <bean class="com.gupaoedu.pojo.UserBean" id="user1,user2,user3"
name="u1,u2,u3"/>
</beans>
```

> id="user1,user2,user3" 只表示一个
>
> name="u1,u2,u3" 表示会被拆分为3个name属性【拆分会根据 ',' ';' ' ' 空格 】

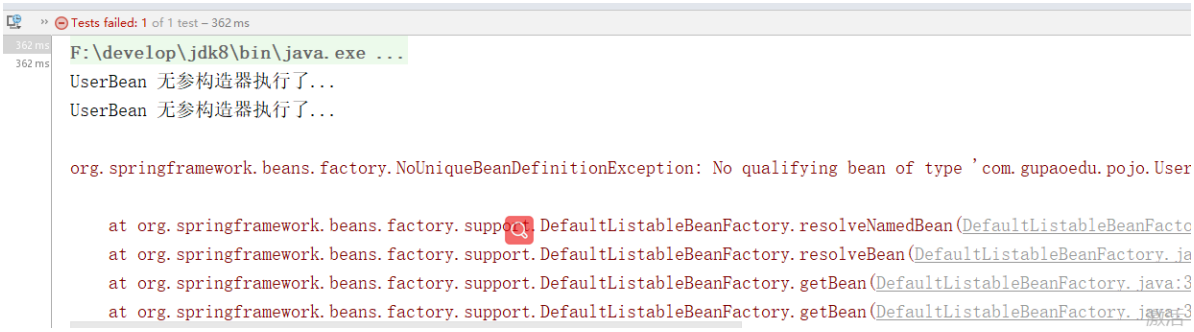
2.3 根据类型

我们可以根据需要获取的对象的类型从容器中获取对象

```
@Test
public void fun6(){
    ApplicationContext ac =
        new ClassPathXmlApplicationContext("applicationContext.xml");
    UserBean bean = ac.getBean(UserBean.class);
    bean.say();
}
```

如果同一类型的对象在容器中有多个，如果我们仅仅只是通过类型来查找，那么就会报错

```
<?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
http://www.springframework.org/schema/beans/spring-beans.xsd">

    <!--  添加需要被容器管理的内容-->
    <!--<bean class="com.gupaoedu.pojo.UserBean" id="userBean"
name="userBean2"/>-->
    <bean class="com.gupaoedu.pojo.UserBean" id="user1,user2,user3" name="u1 u2
u3"/>

    <bean class="com.gupaoedu.pojo.UserBean" id="userBean1" name="ub1"/>
</beans>
```



那怎么解决呢?

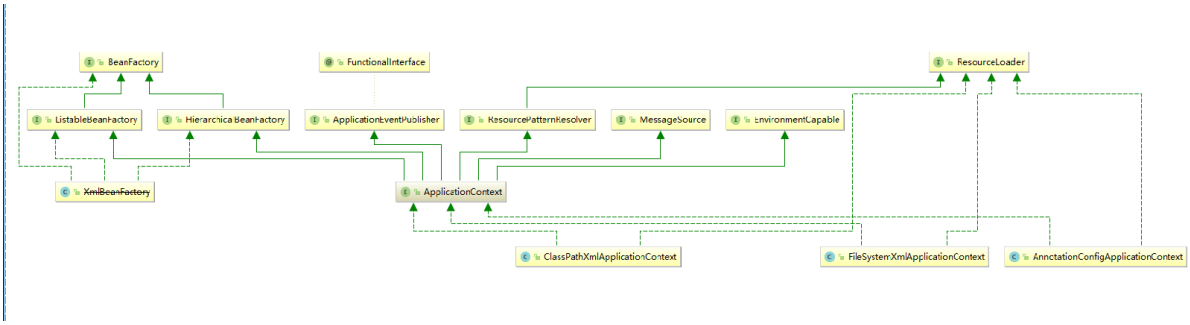
在getBean方法中通过组合条件查找

```
@Test
public void fun6(){
    ApplicationContext ac =
        new ClassPathXmlApplicationContext("applicationContext.xml");
    // UserBean bean = ac.getBean(UserBean.class);
    UserBean bean = ac.getBean("u1",UserBean.class);
    bean.say();
}
```

还要就是我们可以在 中设置 primary属性为true，那么当同一类型有多个对象时，会优先返回primary属性的对象

3.BeanFactory和ApplicationContext的区别

从类图结构中我们可以很清晰的看到ApplicationContext具有BeanFactory的所有功能，同时扩展了很多BeanFactory不具备的功能【事件广播，资源加载，web支持等等...】




```
package com.gupaoedu.test;

import com.gupaoedu.pojo.UserBean;
import org.junit.Test;
import org.springframework.beans.factory.BeanFactory;
import org.springframework.beans.factory.xml.XmlBeanFactory;
import org.springframework.context.ApplicationContext;
import org.springframework.context.support.ClassPathXmlApplicationContext;
import org.springframework.core.io.ClassPathResource;

/**
 * 让每一个人的职业生涯不留遗憾
 *
 * @author 波波老师【咕泡学院】
 */
public class IoCMainTest2 {

    /**
     * ApplicationContext
     *    默认在IoC容器初始化的时候就会实例化对象
     */
    @Test
    public void fun1(){
        ApplicationContext ac =
                new ClassPathXmlApplicationContext("applicationContext.xml");

    }

    /**
     * BeanFactory
     *    IoC容器初始化的时候不会实例化对象
     *    在调用获取的时候才会创建对象
     */
    @Test
```

```
    public void fun2(){
        BeanFactory factory = new XmlBeanFactory(new
ClassPathResource("applicationContext.xml"));
        factory.getBean("u1");
    }
}
```

4.工厂注入

4.1 静态工厂注入

```
package com.gupaoedu.factory;

import com.gupaoedu.pojo.UserBean;

import java.util.HashMap;
import java.util.Map;

/**
 * 让每一个人的职业生涯不留遗憾
 *
 * @author 波波老师【咕泡学院】
 */
public class StaticFactoryDemo {

    public static Map<String,UserBean> hashMap ;

    static {
        hashMap = new HashMap<String, UserBean>();
        hashMap.put("a1",new UserBean());
        hashMap.put("a2",new UserBean());
        hashMap.put("a3",new UserBean());
    }

    /**
     * 静态工厂提供的方法
     * @return
     */
    public static UserBean getInstance(){
        return hashMap.get("a1");
    }
}
```

```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
http://www.springframework.org/schema/beans/spring-beans.xsd">

    <!--  通过静态工厂的方式注入 -->
    <bean class="com.gupaoedu.factory.StaticFactoryDemo" factory-
method="getInstance" id="user"></bean>

</beans>
```

4.2 动态工厂注入

```java
package com.gupaoedu.factory;

import com.gupaoedu.pojo.UserBean;

/**
 * 让每一个人的职业生涯不留遗憾
 *
 * @author 波波老师【咕泡学院】
 */
public class DynamicFactoryDemo {


    public UserBean getInstance(){
        return new UserBean();
    }
}
```

```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
http://www.springframework.org/schema/beans/spring-beans.xsd">

    <!--  通过动态工厂的方式注入 -->
    <bean class="com.gupaoedu.factory.DynamicFactoryDemo"
id="dynamicFactoryDemo" ></bean>

    <!--  从工厂对象中获取 需要的对象-->
    <bean id="user2" factory-bean="dynamicFactoryDemo" factory-
method="getInstance"/>

</beans>
```

5.属性注入【DI】

属性注入主要是指如何给对象中的属性赋值

5.1 构造注入

通过构造方法注入

首先得提供对应的**构造方法**

既可以通过`name`也可以通过`index`来指定要赋值的参数

```
<?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
http://www.springframework.org/schema/beans/spring-beans.xsd">

    <bean class="com.gupaoedu.pojo.UserBean" id="user">
        <!-- 构造注入 -->
        <!--<constructor-arg name="id" value="666"/>
        <constructor-arg name="userName" value="bobo"/>-->
        <constructor-arg index="0" value="999" />
        <constructor-arg index="1" value="gp" />
    </bean>
</beans>
```

5.2 设值注入

注入的属性必须提供对应的setter方法

```
<bean class="com.gupaoedu.pojo.UserBean" id="user1">
    <!-- 设值注入 -->
    <property name="id" value="1"/>
    <property name="userName" value="张三"/>
</bean>
```

简化构造注入和设值注入的操作

首先引入对应的名称空间

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

```
<?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 http://www.sprir

    <bean class="com.gupaoedu.pojo.UserBean" id="user">
        <!-- 构造注入 -->
        <!--<constructor-arg name="id" value="666"/>
        <constructor-arg name="userName" value="bobo"/>-->
        <constructor-arg index="0" value="999" />
        <constructor-arg index="1" value="gp" />
    </bean>

    <bean class="com.gupaoedu.pojo.UserBean" id="user1">
```

5.3 其他注入类型

```
    private Cat cat;

    private String[] favorites;

    private List<Cat> cats;

    private Map<String,Object> map;

    private Properties props;
```

```
<?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
http://www.springframework.org/schema/beans/spring-beans.xsd">

    <!--  注册一个Cat对象-->
    <bean class="com.gupaoedu.pojo.Cat" id="cat" p:nick="花花" p:color="黑色"/>

    <bean class="com.gupaoedu.pojo.UserBean" id="user">
        <!-- 设值注入 -->
        <property name="cat" ref="cat">
            <!--<bean class="com.gupaoedu.pojo.Cat" />-->
        </property>
        <property name="favorites">
            <array>
                <value>篮球</value>
                <value>爬山</value>
                <value>逛街</value>
            </array>
```

```
        </property>
        <property name="cats">
            <list>
                <bean class="com.gupaoedu.pojo.Cat" p:nick="小花1" p:color="红
色"/>
                <bean class="com.gupaoedu.pojo.Cat" p:nick="小花2" p:color="绿
色"/>
                <bean class="com.gupaoedu.pojo.Cat" p:nick="小花3" p:color="黄
色"/>
            </list>
        </property>

        <property name="map" >
            <map>
                <entry key="name1" value="张三1"/>
                <entry key="name2" value="张三2"/>
                <entry key="name3" value="张三3"/>
            </map>
        </property>
        <property name="props">
            <props>
                <prop key="username" >root</prop>
                <prop key="password">123</prop>
            </props>
        </property>
    </bean>
</beans>
```

三、基于Java配置的实现方式

SpringBoot流行之后，Java配置的方式开始被广泛使用。

Java配置类

```
package com.gupaoedu;

import com.gupaoedu.pojo.User;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;

/**
 * 让每一个人的职业生涯不留遗憾
 *      Java配置类 相当于applicationContext.xml
 * @author 波波老师【咕泡学院】
 */
@Configuration
public class JavaConfig {

    /**
     * @Bean 作用和我们在applicationContext.xml中添加的<bean> 效果一样</>
     * 默认的name是方法名称
     * 自定义的name 可以通过value属性或者name属性来指定
     * @return
```

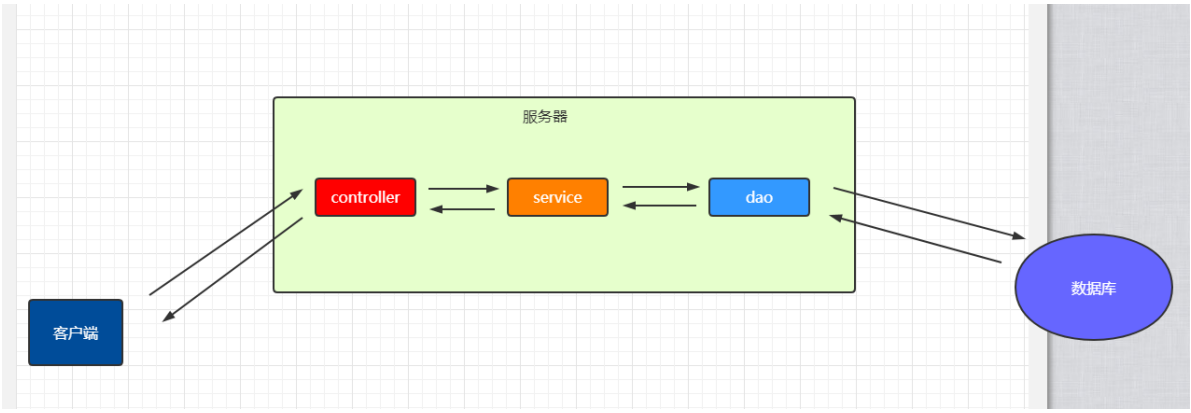
```
     */
    @Bean(name = {"aaa","bbb"})
    public User getUser(){
        User user = new User();
        //user.set....
        return user;
    }
}
```

测试

```
public class MainTest1 {

    @Test
    public void fun1(){
        // 通过@Configuraction注解来初始化IoC容器
        ApplicationContext ac = new
AnnotationConfigApplicationContext(JavaConfig.class);
        System.out.println(ac.getBean("aaa",User.class));
    }
}
```

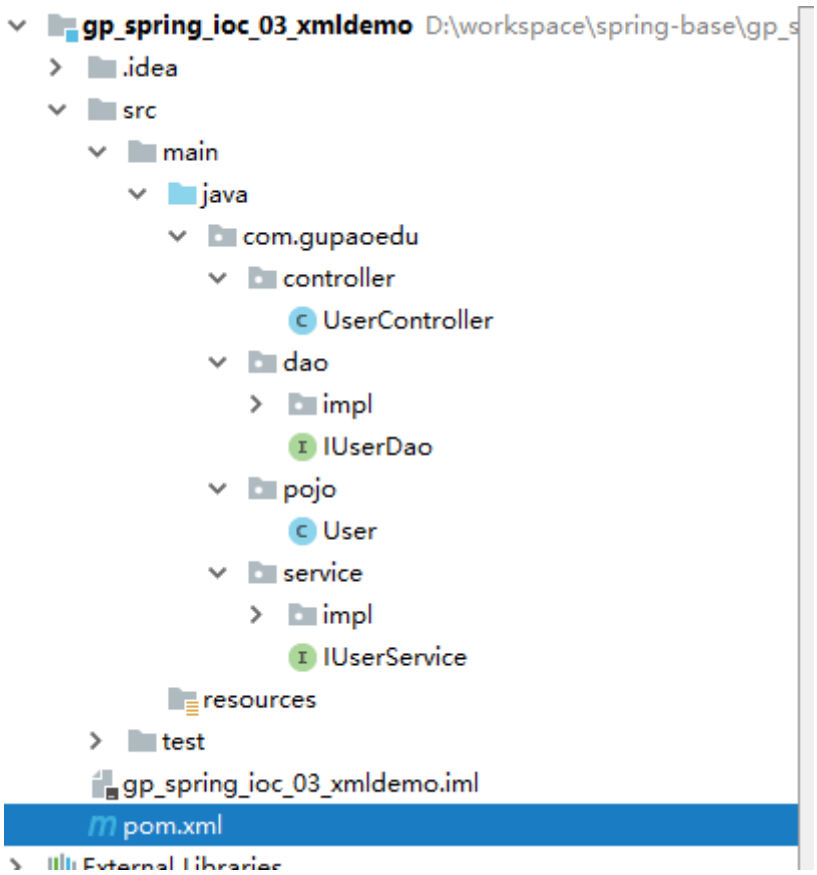
四、时代的潮流之注解编程

4.1 综合小案例



1.基于XML方式的实现

创建基础案例代码



添加Spring相关的依赖

添加Spring的配置文件

```
<?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
http://www.springframework.org/schema/beans/spring-beans.xsd">

    <!-- 将Dao对象注册到容器中 -->
    <bean class="com.gupaoedu.dao.impl.UserDaoImpl" id="userDao"></bean>

    <!-- 将Service对象注册到容器中 -->
    <bean class="com.gupaoedu.service.impl.UserServiceImpl" id="userService">
        <!-- 通过设值注入的方式引入Dao对象 -->
        <property name="dao" ref="userDao"/>
    </bean>

    <!-- 将控制器注册到容器中 -->
    <bean class="com.gupaoedu.controller.UserController" >
        <!-- 通过设值注入的方式引入Service对象 -->
        <property name="service" ref="userService" />
    </bean>
</beans>
```

> 注意：对应的设值注入的属性需要添加对应的setter方法

测试

```
package com.gupaoedu;
```

```
import com.gupaoedu.controller.UserController;
import org.springframework.context.ApplicationContext;
import
org.springframework.context.annotation.AnnotationConfigApplicationContext;
import org.springframework.context.support.ClassPathXmlApplicationContext;

/**
 * 让每一个人的职业生涯不留遗憾
 *
 * @author 波波老师【咕泡学院】
 */
public class AppStarter {

    public static void main(String[] args) {
        ApplicationContext ac = new
ClassPathXmlApplicationContext("applicationContext.xml");
        UserController bean = ac.getBean(UserController.class);
        System.out.println(bean.query());
    }
}
```

2.基于Java配置的方式的实现

基础代码和上面一样

创建对应的Java配置类

```
package com.gupaoedu;

import com.gupaoedu.controller.UserController;
import com.gupaoedu.dao.IUserDao;
import com.gupaoedu.dao.impl.UserDaoImpl;
import com.gupaoedu.service.IUserService;
import com.gupaoedu.service.impl.UserServiceImpl;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;

/**
 * 让每一个人的职业生涯不留遗憾
 *
 * @author 波波老师【咕泡学院】
 */
@Configuration
public class JavaConfig {

    @Bean
    public IUserDao userDao(){
        return new UserDaoImpl();
    }

    /**
     *
```

```
     * @param userDao 本方法调用的时候会从IoC容器中查找类型为IUserDao的名称为 userDao的对象
     * @return
     */
    @Bean
    public IUserService userService(IUserDao userDao){
        IUserService userService = new UserServiceImpl();
        ((UserServiceImpl) userService).setDao(userDao);
        return userService;
    }

    @Bean
    public UserController userController(IUserService userService){
        UserController controller = new UserController();
        controller.setService(userService);
        return controller;
    }
}
```

测试

```
public class AppStarter {

    public static void main(String[] args) {
        ApplicationContext ac = new AnnotationConfigApplicationContext(JavaConfig.class);
        System.out.println(ac.getBean(UserController.class).query());
    }
}
```

4.2 配置注解

注解名称	说明
@Configuration	把一个类作为一个IoC容器，它的某个方法头上如果注册了@Bean，就会作为这个Spring容器中的Bean。
@ComponentScan	在配置类上添加 @ComponentScan 注解。该注解默认会扫描该类所在的包下所有的配置类，相当于之前的 context:component-scan
@Scope	用于指定scope作用域的（用在类上）
@Lazy	表示延迟初始化
@Conditional	Spring4开始提供，它的作用是按照一定的条件进行判断，满足条件给容器注册Bean。
@Import	导入外部资源
生命周期控制	@PostConstruct用于指定初始化方法（用在方法上）@PreDestory用于指定销毁方法（用在方法上）@DependsOn：定义Bean初始化及销毁时的顺序

扩展：SpringBoot中的ConditionalXXX

@Conditional扩展注解	作用(判断是否满足当前指定条件)
@ConditionalOnJava	系统的Java版本是否符合要全
@ConditionalOnBean	容器中存在指定的Bean
@ConditionalOnMissingBean	容器中不存在指定的Bean
@ConditionalOnExpression	满足SpEL表达式
@ConditionalOnClass	系统中有指定的类
@ConditionalOnMissingClass	系统中没有指定的类
@ConditionalOnSingleCandidate	容器中只有一个指定的Bean，或者这个Bean是首选Bean
@ConditionalOnProperty	系统中指定的属性是否有指定的值
@ConditionalOnResource	类路径下是否存在指定的资源文件
@ConditionalOnWebApplication	当前是Web环境
@ConditionalOnNotWebApplication	当前不是Web环境
@ConditionalOnJndi	JNDI存在指定项

4.3 赋值注解

注解名称	说明
@Component	泛指组件，当组件不好归类的时候，我们可以使用这个注解进行标注。
@Service	用于标注业务层组件
@Controller	用于标注控制层组件
@Repository	用于标注数据访问组件，即DAO组件。
@Value	普通数据类型赋值
@Autowired	默认按类型装配，如果我们想使用按名称装配，可以结合@Qualifier注解一起使用
@PropertySource	读取配置文件赋值
@Qualifier	如存在多个实例配合使用
@Primary	自动装配时当出现多个Bean候选者时，被注解为@Primary的Bean将作为首选者，否则将抛出异常
@Resource	默认按名称装配，当找不到与名称匹配的bean才会按类型装配。

4.4 注解编程的使用

我们需要将需要被IoC容器管理的类型通过`@Componenet`注解标注

```
@Component // 需要被IoC容器加载
public class UserController {

    @Autowired
    private IUserService service;

    public List<User> query(){

        return service.query();
    }
}
```

1.基于配置文件的方式

在配置文件中我们需要添加对应的扫描

```
<?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
       http://www.springframework.org/schema/beans/spring-beans.xsd
       http://www.springframework.org/schema/context
       http://www.springframework.org/schema/context/spring-context.xsd
">


    <!-- 添加扫描的路径 指定从哪些package下加载被 @Component标注的类型-->
    <!--<context:component-scan base-package="com.gupaoedu"/>-->
    <!--<context:component-scan base-package="com.gupaoedu.controller
    ,com.gupaoedu.service.impl
    ,com.gupaoedu.dao.impl"/>-->
    <context:component-scan base-package="com.gupaoedu.controller" />
    <context:component-scan base-package="com.gupaoedu.service.impl" />
    <context:component-scan base-package="com.gupaoedu.dao.impl" />
</beans>
```

显示的限制控制层只能用`@Controller`注解，业务逻辑层和持久层不能使用`@Controller`注解

```
<?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
       http://www.springframework.org/schema/beans/spring-beans.xsd
       http://www.springframework.org/schema/context
       http://www.springframework.org/schema/context/spring-context.xsd
">


    <!-- 添加扫描的路径 指定从哪些package下加载被 @Component标注的类型-->
    <!--<context:component-scan base-package="com.gupaoedu"/>-->
    <!--<context:component-scan base-package="com.gupaoedu.controller
    ,com.gupaoedu.service.impl
    ,com.gupaoedu.dao.impl"/>-->
    <!--
    use-default-filters="false" 表示不适用默认的过滤器
        默认过滤器会识别 @Componenet @Controller @Service @Repository
    -->
    <context:component-scan base-package="com.gupaoedu.controller" use-default-
filters="false">
        <context:include-filter type="annotation"
expression="org.springframework.stereotype.Controller"/>
    </context:component-scan>
    <context:component-scan base-
package="com.gupaoedu.service.impl,com.gupaoedu.dao.impl" use-default-
filters="true" >
        <!-- 排除掉某个注解 -->
        <context:exclude-filter type="annotation"
expression="org.springframework.stereotype.Controller" />
    </context:component-scan>

</beans>
```

@Autowired和@Resource的区别

@Autowired：默认只能根据类型来查找，可以结合@Qualifier("abc")注解来实现通过name查找

@Resource：默认同样是根据类型来查找，但是提供的有type和name属性类实现不同的查找方式

2.基于Java配置类的方式

我们需要在Java配置类中通过@ComponentScan注解来指定扫描的路径

默认的情况下扫描的是当前路径及其子路径下的所有的被@Componenet @Controller @Service @Repository标注的类型

```
package com.gupaoedu;

import com.gupaoedu.controller.UserController;
import com.gupaoedu.dao.IUserDao;
import com.gupaoedu.dao.impl.UserDaoImpl;
import com.gupaoedu.service.IUserService;
import com.gupaoedu.service.impl.UserServiceImpl;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.ComponentScan;
import org.springframework.context.annotation.ComponentScans;
import org.springframework.context.annotation.Configuration;
import org.springframework.stereotype.Controller;

/**
 * 让每一个人的职业生涯不留遗憾
 *  @ComponentScan 如果不去指定扫描的路基，默认是会扫描当前目录及其子目录下的所有的
 *                  被@Componenet @Controller @Service @Repository标注的类型
 * @author 波波老师【咕泡学院】
 */
@Configuration
/*@ComponentScan(value = {"com.gupaoedu.controller"}
,useDefaultFilters = false
        ,includeFilters = {@ComponentScan.Filter(Controller.class)})*/
@ComponentScans({
        @ComponentScan(value = {"com.gupaoedu.controller"}
                ,useDefaultFilters = false
                ,includeFilters = {@ComponentScan.Filter(Controller.class)})
        ,@ComponentScan(value = {"com.gupaoedu.service","com.gupaoedu.dao"}
                ,useDefaultFilters = true
                ,excludeFilters = {@ComponentScan.Filter(Controller.class)})
})
public class JavaConfig {

}
```

3.@Value注解介绍

@Value 帮助我们给数组动态的设值

```
package com.gupaoedu.pojo;

import org.springframework.beans.factory.annotation.Value;
import org.springframework.core.io.Resource;
import org.springframework.stereotype.Component;

/**
 * 让每一个人的职业生涯不留遗憾
 *
 * @author 波波老师【咕泡学院】
 */
@Component
public class User {

    @Value("bobo") // 注入普通的字符串
    private String userName ;

    @Value("#{systemProperties['os.name']}")
    private String systemPropertiesName; // 注入操作系统的信息

    @Value("#{T(java.lang.Math).random()*100}")
    private double randomNumber; // 注入表达式的结果

    @Value("#{person.personName}")
    private String fromPersonName; // 注入其他Bean的属性


    @Value("classpath:test.txt")
    private Resource resourceFile;

    @Value("http://www.baidu.com")
    private Resource baiduFile;

    public String getUserName() {
        return userName;
    }

    public void setUserName(String userName) {
        this.userName = userName;
    }

    public String getSystemPropertiesName() {
        return systemPropertiesName;
    }

    public void setSystemPropertiesName(String systemPropertiesName) {
        this.systemPropertiesName = systemPropertiesName;
    }

    public double getRandomNumber() {
        return randomNumber;
    }
```

```
    public void setRandomNumber(double randomNumber) {
        this.randomNumber = randomNumber;
    }

    public String getFromPersonName() {
        return fromPersonName;
    }

    public void setFromPersonName(String fromPersonName) {
        this.fromPersonName = fromPersonName;
    }

    public Resource getResourceFile() {
        return resourceFile;
    }

    public void setResourceFile(Resource resourceFile) {
        this.resourceFile = resourceFile;
    }

    public Resource getBaiduFile() {
        return baiduFile;
    }

    public void setBaiduFile(Resource baiduFile) {
        this.baiduFile = baiduFile;
    }

    @Override
    public String toString() {
        return "User{" +
                "userName='" + userName + '\'' +
                ", systemPropertiesName='" + systemPropertiesName + '\'' +
                ", randomNumber=" + randomNumber +
                ", fromPersonName='" + fromPersonName + '\'' +
                ", resourceFile=" + resourceFile +
                ", baiduFile=" + baiduFile +
                '}';
    }
}
```

```
package com.gupaoedu.pojo;

import org.springframework.beans.factory.annotation.Value;
import org.springframework.stereotype.Component;

/**
 * 让每一个人的职业生涯不留遗憾
 *
 * @author 波波老师【咕泡学院】
 */
@Component
public class Person {
```

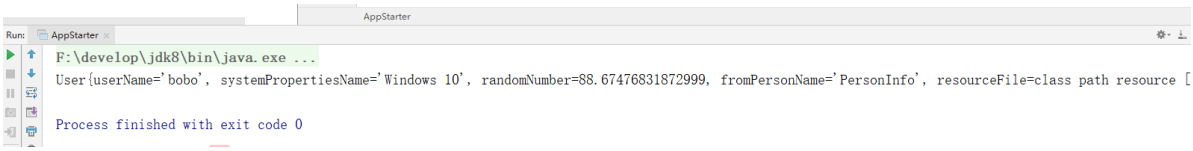
```
    @Value("PersonInfo")
    private String personName;

    public String getPersonName() {
        return personName;
    }

    public void setPersonName(String personName) {
        this.personName = personName;
    }
}
```

Java配置

```
package com.gupaoedu.config;

import org.springframework.context.annotation.ComponentScan;
import org.springframework.context.annotation.Configuration;

/**
 * 让每一个人的职业生涯不留遗憾
 *
 * @author 波波老师【咕泡学院】
 */
@Configuration
@ComponentScan("com.gupaoedu")
public class JavaConfig {

}
```

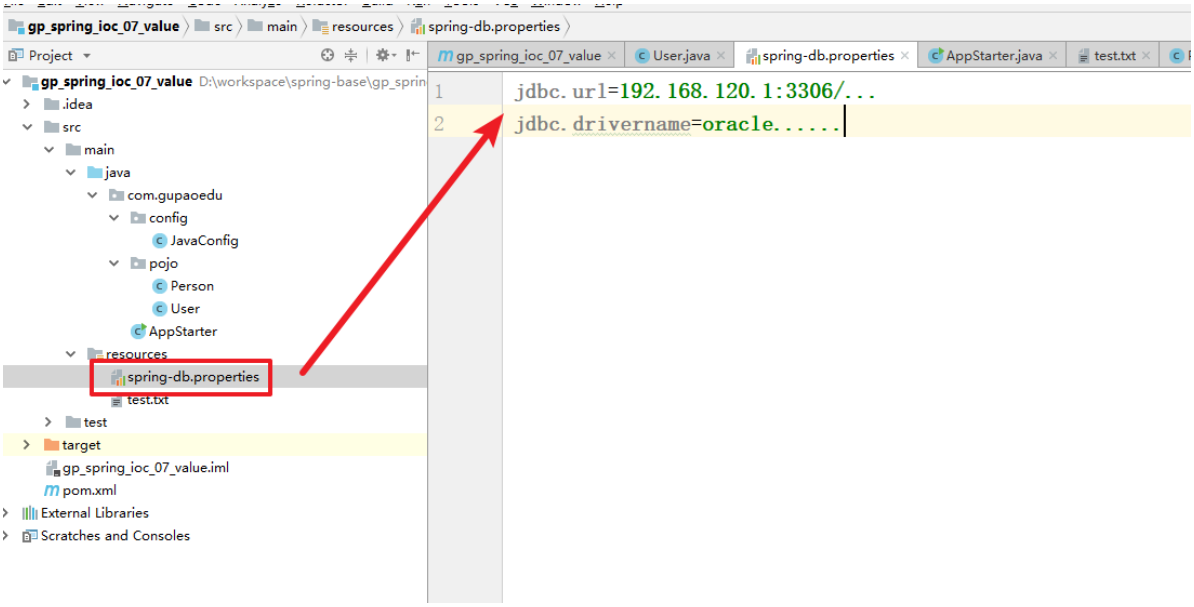
```
public class AppStarter {

    public static void main(String[] args) {
        ApplicationContext ac = new
AnnotationConfigApplicationContext(JavaConfig.class);
        System.out.println(ac.getBean(User.class));
        User user = ac.getBean(User.class);
    }
}
```

运行效果



读取第三方的Properties文件中的信息

创建第三方的属性文件



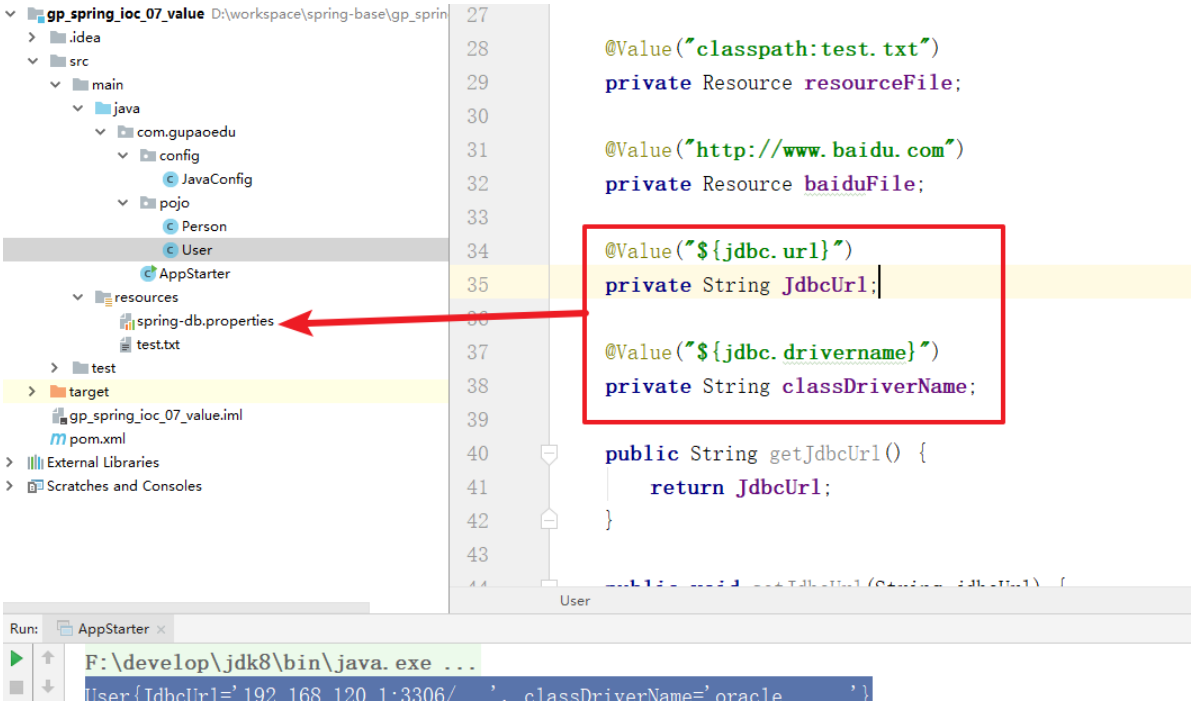
在需要引入属性文件中的信息的时候通过@Value注解实现

```
28      @Value("classpath:test.txt")
29      private Resource resourceFile;
30
31      @Value("http://www.baidu.com")
32      private Resource baiduFile;
33
34      @Value("${jdbc.url}")
35      private String JdbcUrl;
36
37      @Value("${jdbc.drivername}")
38      private String classDriverName;
39
40      public String getJdbcUrl() {
41          return JdbcUrl;
42      }
```

注意我们需要在Java配置类中通过@PropertySouce注解来显示的引入属性文件

```
@Configuration
@ComponentScan("com.gupaoedu")
// 显示的指定要加载的属性文件
@PropertySource({"classpath:spring-db.properties"})
public class JavaConfig {

}
```

效果



4.@PostConstruct @PreDestory @DependsOn

@PostConstruct 系统初始化完成后的回调方法

@PreDestory 系统销毁时的回调方法

@DependsOn指定实例化对象的先后顺序

```
@Component
@DependsOn({"user"}) // Person的实例化依赖于User对象的实例化，也就是User先于Person实例化
public class Person {

    public Person(){
        System.out.println("Person 构造方法执行了...");
    }
}
```

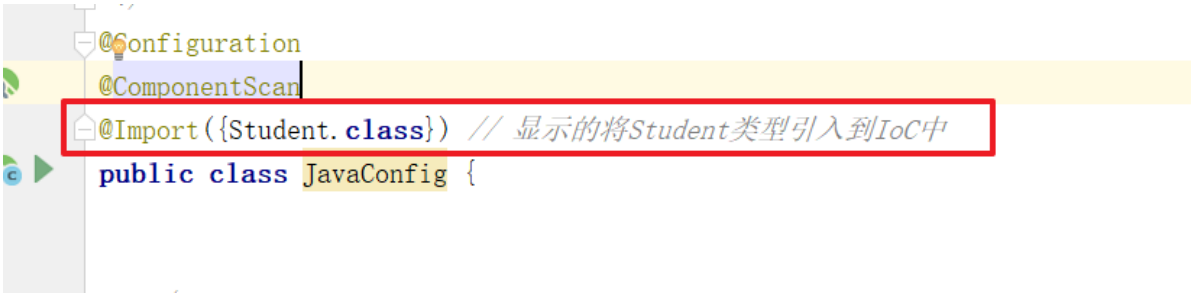
5.@Import注解

将类型加入到IoC容器中的方式有哪些?

> 1.基于XML文件的方式
>
> 2.基于XML文件的方式context:Component-Scan + @Component
>
> 3.基于Java配置类@Bean
>
> 4.基于Java配置类@ComponentScan + @Component
>
> 5.FactoryBean + getObject方法
>
> 6.@Import

5.1 静态使用

直接在@Import注解中直接定义要引入到IoC容器中的类型



缺点是：无法灵活的指定引入的类型

5.2 动态使用-ImportSelector

通过重写selectImports方法来达到灵活控制引入类型的目的

```
public class GpImportSelector implements ImportSelector {

    /**
     *
     * @param annotationMetadata
     * @return
     *    IoC 要加载的类型的全路径的字符串数组
     */
    public String[] selectImports(AnnotationMetadata annotationMetadata) {
        // 在此处实现不同的业务逻辑控制
        return new String[]
{LoggerService.class.getName(),CacheService.class.getName()};
    }
}
```

5.3 动态使用-ImportBeanDefinitionRegistrar

其实和5.2差不多

```
package com.gupaoedu.demo;

import org.springframework.beans.factory.support.BeanDefinitionRegistry;
import org.springframework.beans.factory.support.RootBeanDefinition;
import org.springframework.context.annotation.ImportBeanDefinitionRegistrar;
import org.springframework.core.type.AnnotationMetadata;

/**
 * 让每一个人的职业生涯不留遗憾
 *
 * @author 波波老师【咕泡学院】
 */
public class GpImportBeanDefinitionRegistrar implements
ImportBeanDefinitionRegistrar {
    /**
     *
     * @param annotationMetadata
```

```
     * @param beanDefinitionRegistry IoC容器中管理对象的一个注册器
     */
    public void registerBeanDefinitions(AnnotationMetadata annotationMetadata,
BeanDefinitionRegistry beanDefinitionRegistry) {
        // 需要将添加的对象包装为一个RootBeanDefinition对象
        RootBeanDefinition cache = new RootBeanDefinition(CacheService.class);
        beanDefinitionRegistry.registerBeanDefinition("cache",cache);

        RootBeanDefinition logger = new RootBeanDefinition(LoggerService.class);
        beanDefinitionRegistry.registerBeanDefinition("logger",logger);
    }
}
```

五、SpringBoot的基础条件之条件注解

@Conditional注解 通过重写matches方法来动态实现是否加载某类型

```
package com.gupaoedu.conditional;

import org.springframework.context.annotation.Condition;
import org.springframework.context.annotation.ConditionContext;
import org.springframework.core.type.AnnotatedTypeMetadata;

/**
 * 让每一个人的职业生涯不留遗憾
 *
 * @author 波波老师【咕泡学院】
 */
public class ConditionalOnBean implements Condition {
    /**
     *
     * @param conditionContext
     * @param annotatedTypeMetadata
     * @return
     *    true 表示IoC容器加载该类型
     *    false 表示IoC容器不加载该类型
     */
    public boolean matches(ConditionContext conditionContext,
AnnotatedTypeMetadata annotatedTypeMetadata) {
        return true;
    }
}
```

```java
package com.gupaoedu;

import com.gupaoedu.conditional.ConditionalOnBean;
import com.gupaoedu.pojo.User;
import org.springframework.context.ApplicationContext;
import
org.springframework.context.annotation.AnnotationConfigApplicationContext;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Conditional;
import org.springframework.context.annotation.Configuration;

/**
 * 让每一个人的职业生涯不留遗憾
 *
 * @author 波波老师【咕泡学院】
 */
@Configuration
public class JavaConfig {

    /**
     * @Conditional(ConditionalOnBean.class)
     *    是一个条件注解 表示如果ConditionalOnBean中的matches方法返回true就加载
     *        返回false就不加载
     * @return
     */
    @Bean
    @Conditional(ConditionalOnBean.class)
    public User user(){
        return new User();
    }

    public static void main(String[] args) {
        ApplicationContext ac = new
AnnotationConfigApplicationContext(JavaConfig.class);
        String[] beanDefinitionNames = ac.getBeanDefinitionNames();
        for (String beanName:beanDefinitionNames){
            System.out.println(beanName);
        }
    }
}
```

扩展：SpringBoot中的ConditionalXXX

@Conditional扩展注解	作用(判断是否满足当前指定条件)
@ConditionalOnJava	系统的Java版本是否符合要全
@ConditionalOnBean	容器中存在指定的Bean
@ConditionalOnMissingBean	容器中不存在指定的Bean
@ConditionalOnExpression	满足SpEL表达式
@ConditionalOnClass	系统中有指定的类
@ConditionalOnMissingClass	系统中没有指定的类
@ConditionalOnSingleCandidate	容器中只有一个指定的Bean，或者这个Bean是首选Bean
@ConditionalOnProperty	系统中指定的属性是否有指定的值
@ConditionalOnResource	类路径下是否存在指定的资源文件
@ConditionalOnWebApplication	当前是Web环境
@ConditionalOnNotWebApplication	当前不是Web环境
@ConditionalOnJndi	JNDI存在指定项

具体案例

```
public class ConditionalOnClass implements Condition {
    public boolean matches(ConditionContext conditionContext,
AnnotatedTypeMetadata annotatedTypeMetadata) {
        try {
            Class<?> aClass =
conditionContext.getClassLoader().loadClass("com.gupaoedu.test.Test1");
            return aClass==null?false:true;
        } catch (ClassNotFoundException e) {
            e.printStackTrace();
        }
        return false;
    }
}
```

```
package com.gupaoedu.conditional;

import org.springframework.context.annotation.Condition;
import org.springframework.context.annotation.ConditionContext;
import org.springframework.core.type.AnnotatedTypeMetadata;

/**
 * 让每一个人的职业生涯不留遗憾
 *
```

```
 * @author 波波老师【咕泡学院】
 */
public class ConditionalOnBean implements Condition {
    /**
     *  如果IoC容器中有Person对象就返回true 否在返回false
     *
     * @param conditionContext
     * @param annotatedTypeMetadata
     * @return
     *    true 表示IoC容器加载该类型
     *    false 表示IoC容器不加载该类型
     */
    public boolean matches(ConditionContext conditionContext,
AnnotatedTypeMetadata annotatedTypeMetadata) {
        boolean flag =
conditionContext.getRegistry().containsBeanDefinition("person");
        System.out.println(flag + " **** ");
        return flag;
    }
}
```

六、多环境下的解决方案之Profile

Profile本质就是Conditional的实现

```
package com.gupaoedu.pojo;

/**
 * 让每一个人的职业生涯不留遗憾
 *
 * @author 波波老师【咕泡学院】
 */
public class GpDataSource {

    private String username;

    private String password;

    private String url;

    public String getUsername() {
        return username;
    }

    public void setUsername(String username) {
        this.username = username;
    }

    public String getPassword() {
        return password;
    }
```

```
    public void setPassword(String password) {
        this.password = password;
    }

    public String getUrl() {
        return url;
    }

    public void setUrl(String url) {
        this.url = url;
    }

    public GpDataSource(String username, String password, String url) {
        this.username = username;
        this.password = password;
        this.url = url;
    }

    @Override
    public String toString() {
        return "GpDataSource{" +
                "username='" + username + '\'' +
                ", password='" + password + '\'' +
                ", url='" + url + '\'' +
                '}';
    }

    public GpDataSource() {
    }
}
```

```
package com.gupaoedu;

import com.gupaoedu.conditional.ConditionalOnBean;
import com.gupaoedu.conditional.ConditionalOnClass;
import com.gupaoedu.pojo.GpDataSource;
import com.gupaoedu.pojo.Person;
import com.gupaoedu.pojo.User;
import org.springframework.context.ApplicationContext;
import org.springframework.context.annotation.*;

/**
 * 让每一个人的职业生涯不留遗憾
 *
 * @author 波波老师【咕泡学院】
 */
@Configuration
public class JavaConfig {

    //@Bean
    public Person person(){
        return new Person();
    }
```

```
    /**
     * @Conditional(ConditionalOnBean.class)
     *     是一个条件注解 表示如果ConditionalOnBean中的matches方法返回true就加载
     *         返回false就不加载
     * @return
     */
    @Bean
    //@Conditional(ConditionalOnBean.class)
    @Conditional(ConditionalOnClass.class)
    public User user(){
        return new User();
    }

    @Bean
    @Profile("pro") // 其实Profile注解本质上就是Conditional的一种实现
    public GpDataSource proDataSource(){
        GpDataSource ds = new GpDataSource("root","123","192.168.11.190");
        return ds;
    }

    @Bean
    @Profile("dev")
    public GpDataSource devDataSource(){
        GpDataSource ds = new GpDataSource("admin","456","192.168.12.190");
        return ds;
    }

    public static void main(String[] args) {
        AnnotationConfigApplicationContext ac = new
AnnotationConfigApplicationContext();
        /*String[] beanDefinitionNames = ac.getBeanDefinitionNames();
        for (String beanName:beanDefinitionNames){
            System.out.println(beanName);
        }*/
        ac.getEnvironment().setActiveProfiles("dev");
        ac.register(JavaConfig.class);
        ac.refresh();
        System.out.println(ac.getBean(GpDataSource.class));
    }
}
```

七、Bean对象的作用域

作用域	说明
prototype	每次请求，都是一个新的Bean（java原型模式）
singleton	bean是单例的（Java单例模式）
request	在一次请求中，bean的声明周期和request同步
session	bean的生命周期和session同步

默认的情况是 `singleton`

```
@Bean
@Scope("prototype")
public Person person(){
    return new Person();
}
```

```
@Bean
@Scope("singleton")
public Person person(){
    return new Person();
}
```

八、SpringIoC源码浅析

晕车~~~！！！

分析的入口

8.1 IoC 初始源码分析

```
// IoC 容器的初始化
ApplicationContext ac = new
ClassPathXmlApplicationContext("applicationContext.xml");
```

```
public ClassPathXmlApplicationContext(String[] configLocations, boolean refresh,
@Nullable ApplicationContext parent) throws BeansException {
    super(parent);
    this.setConfigLocations(configLocations);
    if (refresh) {
        this.refresh();
    }

}
```

```
public void refresh() throws BeansException, IllegalStateException {
    Object var1 = this.startupShutdownMonitor;
    synchronized(this.startupShutdownMonitor) {
        this.prepareRefresh();
        // IoC 容器的初始化操作
        ConfigurableListableBeanFactory beanFactory =
this.obtainFreshBeanFactory();
        this.prepareBeanFactory(beanFactory);

        try {
            this.postProcessBeanFactory(beanFactory);
            this.invokeBeanFactoryPostProcessors(beanFactory);
            this.registerBeanPostProcessors(beanFactory);
            this.initMessageSource();
            this.initApplicationEventMulticaster();
            this.onRefresh();
            this.registerListeners();
            this.finishBeanFactoryInitialization(beanFactory);
            this.finishRefresh();
        } catch (BeansException var9) {
            if (this.logger.isWarnEnabled()) {
                this.logger.warn("Exception encountered during context
initialization - cancelling refresh attempt: " + var9);
            }

            this.destroyBeans();
            this.cancelRefresh(var9);
            throw var9;
        } finally {
            this.resetCommonCaches();
        }

    }
}
```

```
protected ConfigurableListableBeanFactory obtainFreshBeanFactory() {
    // 刷新IoC容器
    this.refreshBeanFactory();
    // 获取Bean Factory对象 --》 已经完成了IoC的初始化操作
    return this.getBeanFactory();
}
```

```
protected final void refreshBeanFactory() throws BeansException {
    if (this.hasBeanFactory()) {
        this.destroyBeans();
        this.closeBeanFactory();
```

```
    }

    try {
        // 创建Bean Factory对象 DefaultListableBeanFactory
        DefaultListableBeanFactory beanFactory = this.createBeanFactory();
        beanFactory.setSerializationId(this.getId());
        this.customizeBeanFactory(beanFactory);
        // 加载 BeanDefinition -> <bean>
        this.loadBeanDefinitions(beanFactory);
        this.beanFactory = beanFactory;
    } catch (IOException var2) {
        throw new ApplicationContextException("I/O error parsing bean definition
source for " + this.getDisplayName(), var2);
    }
}
```

```
protected void loadBeanDefinitions(DefaultListableBeanFactory beanFactory)
throws BeansException, IOException {
    XmlBeanDefinitionReader beanDefinitionReader = new
XmlBeanDefinitionReader(beanFactory);
    beanDefinitionReader.setEnvironment(this.getEnvironment());
    beanDefinitionReader.setResourceLoader(this);
    beanDefinitionReader.setEntityResolver(new ResourceEntityResolver(this));
    this.initBeanDefinitionReader(beanDefinitionReader);
    this.loadBeanDefinitions(beanDefinitionReader);
}
```

```
protected DefaultListableBeanFactory createBeanFactory() {
    return new DefaultListableBeanFactory(this.getInternalParentBeanFactory());
}
```

获取需要加载读取的配置文件

```
protected void loadBeanDefinitions(XmlBeanDefinitionReader reader) throws
BeansException, IOException {
    Resource[] configResources = this.getConfigResources();
    if (configResources != null) {
        reader.loadBeanDefinitions(configResources);
    }

    String[] configLocations = this.getConfigLocations();
    if (configLocations != null) {
        reader.loadBeanDefinitions(configLocations);
    }

}
```

```
public int loadBeanDefinitions(String... locations) throws
BeanDefinitionStoreException {
    Assert.notNull(locations, "Location array must not be null");
    int count = 0;
    String[] var3 = locations;
    int var4 = locations.length;
    // 循环加载每个配置文件
    for(int var5 = 0; var5 < var4; ++var5) {
        String location = var3[var5];
        count += this.loadBeanDefinitions(location);
    }

    return count;
}
```

```
public int loadBeanDefinitions(String location, @Nullable Set<Resource>
actualResources) throws BeanDefinitionStoreException {
    ResourceLoader resourceLoader = this.getResourceLoader();
    if (resourceLoader == null) {
        throw new BeanDefinitionStoreException("Cannot load bean definitions
from location [" + location + "]: no ResourceLoader available");
    } else {
        int count;
        if (resourceLoader instanceof ResourcePatternResolver) {
            try {
                Resource[] resources =
((ResourcePatternResolver)resourceLoader).getResources(location);
                // 进入
                count = this.loadBeanDefinitions(resources);
                if (actualResources != null) {
                    Collections.addAll(actualResources, resources);
                }

                if (this.logger.isTraceEnabled()) {
                    this.logger.trace("Loaded " + count + " bean definitions
from location pattern [" + location + "]");
                }

                return count;
            } catch (IOException var6) {
                throw new BeanDefinitionStoreException("Could not resolve bean
definition resource pattern [" + location + "]", var6);
            }
        } else {
            Resource resource = resourceLoader.getResource(location);
            count = this.loadBeanDefinitions((Resource)resource);
            if (actualResources != null) {
                actualResources.add(resource);
            }
```

```
            if (this.logger.isTraceEnabled()) {
                this.logger.trace("Loaded " + count + " bean definitions from
location [" + location + "]");
            }

            return count;
        }
    }
}
```

```
public int loadBeanDefinitions(EncodedResource encodedResource) throws
BeanDefinitionStoreException {
    Assert.notNull(encodedResource, "EncodedResource must not be null");
    if (this.logger.isTraceEnabled()) {
        this.logger.trace("Loading XML bean definitions from " +
encodedResource);
    }

    Set<EncodedResource> currentResources =
(Set)this.resourcesCurrentlyBeingLoaded.get();
    if (currentResources == null) {
        currentResources = new HashSet(4);
        this.resourcesCurrentlyBeingLoaded.set(currentResources);
    }

    if (!((Set)currentResources).add(encodedResource)) {
        throw new BeanDefinitionStoreException("Detected cyclic loading of " +
encodedResource + " - check your import definitions!");
    } else {
        int var5;
        try {
            // 获取需要读取的配置文件的字节输入流
            InputStream inputStream =
encodedResource.getResource().getInputStream();

            try {
                InputSource inputSource = new InputSource(inputStream);
                if (encodedResource.getEncoding() != null) {
                    inputSource.setEncoding(encodedResource.getEncoding());
                }
                // 进入
                var5 = this.doLoadBeanDefinitions(inputSource,
encodedResource.getResource());
            } finally {
                inputStream.close();
            }
        } catch (IOException var15) {
            throw new BeanDefinitionStoreException("IOException parsing XML
document from " + encodedResource.getResource(), var15);
        } finally {
            ((Set)currentResources).remove(encodedResource);
            if (((Set)currentResources).isEmpty()) {
```

```
                this.resourcesCurrentlyBeingLoaded.remove();
            }

        }

        return var5;
    }
}
```

```
protected int doLoadBeanDefinitions(InputSource inputSource, Resource resource)
throws BeanDefinitionStoreException {
    try {
        // 通过SAX加载配置文件 获取对于的Document对象
        Document doc = this.doLoadDocument(inputSource, resource);
        // 注册BeanDefinition
        int count = this.registerBeanDefinitions(doc, resource);
        if (this.logger.isDebugEnabled()) {
            this.logger.debug("Loaded " + count + " bean definitions from " +
resource);
        }

        return count;
    } catch (BeanDefinitionStoreException var5) {
        throw var5;
    } catch (SAXParseException var6) {
        throw new XmlBeanDefinitionStoreException(resource.getDescription(),
"Line " + var6.getLineNumber() + " in XML document from " + resource + " is
invalid", var6);
    } catch (SAXException var7) {
        throw new XmlBeanDefinitionStoreException(resource.getDescription(),
"XML document from " + resource + " is invalid", var7);
    } catch (ParserConfigurationException var8) {
        throw new BeanDefinitionStoreException(resource.getDescription(),
"Parser configuration exception parsing XML from " + resource, var8);
    } catch (IOException var9) {
        throw new BeanDefinitionStoreException(resource.getDescription(),
"IOException parsing XML document from " + resource, var9);
    } catch (Throwable var10) {
        throw new BeanDefinitionStoreException(resource.getDescription(),
"Unexpected exception parsing XML document from " + resource, var10);
    }
}
```

```
public int registerBeanDefinitions(Document doc, Resource resource) throws
BeanDefinitionStoreException {
    BeanDefinitionDocumentReader documentReader =
this.createBeanDefinitionDocumentReader();
    int countBefore = this.getRegistry().getBeanDefinitionCount();
    // 注册
    documentReader.registerBeanDefinitions(doc,
this.createReaderContext(resource));
    return this.getRegistry().getBeanDefinitionCount() - countBefore;
}
```

```
protected void doRegisterBeanDefinitions(Element root) {
    BeanDefinitionParserDelegate parent = this.delegate;
    this.delegate = this.createDelegate(this.getReaderContext(), root, parent);
    if (this.delegate.isDefaultNamespace(root)) {
        String profileSpec = root.getAttribute("profile");
        if (StringUtils.hasText(profileSpec)) {
            String[] specifiedProfiles =
StringUtils.tokenizeToStringArray(profileSpec, ",; ");
            if
(!this.getReaderContext().getEnvironment().acceptsProfiles(specifiedProfiles)) {
                if (this.logger.isDebugEnabled()) {
                    this.logger.debug("Skipped XML bean definition file due to
specified profiles [" + profileSpec + "] not matching: " +
this.getReaderContext().getResource());
                }

                return;
            }
        }
    }
    // 解析之前操作
    this.preProcessXml(root);
    this.parseBeanDefinitions(root, this.delegate);
    // 解析之后的操作
    this.postProcessXml(root);
    this.delegate = parent;
}
```

```
protected void parseBeanDefinitions(Element root, BeanDefinitionParserDelegate
delegate) {
    if (delegate.isDefaultNamespace(root)) {
        NodeList nl = root.getChildNodes();

        for(int i = 0; i < nl.getLength(); ++i) {
            Node node = nl.item(i);
            if (node instanceof Element) {
                Element ele = (Element)node;
```

```
                if (delegate.isDefaultNamespace(ele)) {
                    // 默认的元素解析 <bean> <import> ...
                    this.parseDefaultElement(ele, delegate);
                } else {
                    // <aop> <dubbo> ...
                    delegate.parseCustomElement(ele);
                }
            }
        }
    } else {
        delegate.parseCustomElement(root);
    }

}
```

```
private void parseDefaultElement(Element ele, BeanDefinitionParserDelegate
delegate) {
    if (delegate.nodeNameEquals(ele, "import")) {
        // <import>
        this.importBeanDefinitionResource(ele);
    } else if (delegate.nodeNameEquals(ele, "alias")) {
        // <alias>
        this.processAliasRegistration(ele);
    } else if (delegate.nodeNameEquals(ele, "bean")) {
        // <bean>
        this.processBeanDefinition(ele, delegate);
    } else if (delegate.nodeNameEquals(ele, "beans")) {
        // <beans>
        this.doRegisterBeanDefinitions(ele);
    }

}
```

```
protected void processBeanDefinition(Element ele, BeanDefinitionParserDelegate
delegate) {
    // 完成了<bean> 标签的解析
    BeanDefinitionHolder bdHolder = delegate.parseBeanDefinitionElement(ele);
    if (bdHolder != null) {
        bdHolder = delegate.decorateBeanDefinitionIfRequired(ele, bdHolder);

        try {
            // 注册
            BeanDefinitionReaderUtils.registerBeanDefinition(bdHolder,
this.getReaderContext().getRegistry());
        } catch (BeanDefinitionStoreException var5) {
            this.getReaderContext().error("Failed to register bean definition
with name '" + bdHolder.getBeanName() + "'", ele, var5);
        }
```

```
        this.getReaderContext().fireComponentRegistered(new
BeanComponentDefinition(bdHolder));
    }

}
```

```
@Nullable
public BeanDefinitionHolder parseBeanDefinitionElement(Element ele, @Nullable
BeanDefinition containingBean) {
    String id = ele.getAttribute("id");
    String nameAttr = ele.getAttribute("name");
    List<String> aliases = new ArrayList();
    if (StringUtils.hasLength(nameAttr)) {
        String[] nameArr = StringUtils.tokenizeToStringArray(nameAttr, ",; ");
        aliases.addAll(Arrays.asList(nameArr));
    }

    String beanName = id;
    if (!StringUtils.hasText(id) && !aliases.isEmpty()) {
        beanName = (String)aliases.remove(0);
        if (this.logger.isTraceEnabled()) {
            this.logger.trace("No XML 'id' specified - using '" + beanName + "'
as bean name and " + aliases + " as aliases");
        }
    }

    if (containingBean == null) {
        this.checkNameUniqueness(beanName, aliases, ele);
    }
    // 完成了<bean> --> BeanDefinition的转换
    AbstractBeanDefinition beanDefinition = this.parseBeanDefinitionElement(ele,
beanName, containingBean);
    if (beanDefinition != null) {
        if (!StringUtils.hasText(beanName)) {
            try {
                if (containingBean != null) {
                    beanName =
BeanDefinitionReaderUtils.generateBeanName(beanDefinition,
this.readerContext.getRegistry(), true);
                } else {
                    beanName =
this.readerContext.generateBeanName(beanDefinition);
                    String beanClassName = beanDefinition.getBeanClassName();
                    if (beanClassName != null &&
beanName.startsWith(beanClassName) && beanName.length() > beanClassName.length()
&& !this.readerContext.getRegistry().isBeanNameInUse(beanClassName)) {
                        aliases.add(beanClassName);
                    }
                }

                if (this.logger.isTraceEnabled()) {
                    this.logger.trace("Neither XML 'id' nor 'name' specified -
using generated bean name [" + beanName + "]");
```

```
                }
            } catch (Exception var9) {
                this.error(var9.getMessage(), ele);
                return null;
            }
        }

        String[] aliasesArray = StringUtils.toStringArray(aliases);
        return new BeanDefinitionHolder(beanDefinition, beanName, aliasesArray);
    } else {
        return null;
    }
}
```

```
public static void registerBeanDefinition(BeanDefinitionHolder definitionHolder,
BeanDefinitionRegistry registry) throws BeanDefinitionStoreException {
    String beanName = definitionHolder.getBeanName();
    registry.registerBeanDefinition(beanName,
definitionHolder.getBeanDefinition());
    String[] aliases = definitionHolder.getAliases();
    if (aliases != null) {
        String[] var4 = aliases;
        int var5 = aliases.length;

        for(int var6 = 0; var6 < var5; ++var6) {
            String alias = var4[var6];
            registry.registerAlias(beanName, alias);
        }
    }

}
```

```
this.beanDefinitionMap.put(beanName, beanDefinition);
```

8.2 DI过程

```
ApplicationContext ac = new
ClassPathXmlApplicationContext("applicationContext.xml");
        ac.getBean(Person.class);
```

```
@Nullable
private <T> T resolveBean(ResolvableType requiredType, @Nullable Object[] args,
boolean nonUniqueAsNull) {
    // 核心代码
    NamedBeanHolder<T> namedBean = this.resolveNamedBean(requiredType, args,
nonUniqueAsNull);
    if (namedBean != null) {
        return namedBean.getBeanInstance();
    } else {
        BeanFactory parent = this.getParentBeanFactory();
        if (parent instanceof DefaultListableBeanFactory) {
            return
((DefaultListableBeanFactory)parent).resolveBean(requiredType, args,
nonUniqueAsNull);
        } else if (parent != null) {
            ObjectProvider<T> parentProvider =
parent.getBeanProvider(requiredType);
            if (args != null) {
                return parentProvider.getObject(args);
            } else {
                return nonUniqueAsNull ? parentProvider.getIfUnique() :
parentProvider.getIfAvailable();
            }
        } else {
```

```
            return null;
        }
    }
}
```

```
@Nullable
private <T> NamedBeanHolder<T> resolveNamedBean(ResolvableType requiredType,
@Nullable Object[] args, boolean nonUniqueAsNull) throws BeansException {
    Assert.notNull(requiredType, "Required type must not be null");
    // 获取候选的类型的全路径字符串数组
    String[] candidateNames = this.getBeanNamesForType(requiredType);
    String[] var6;
    int var7;
    int var8;
    String beanName;
    if (candidateNames.length > 1) {
        List<String> autowireCandidates = new ArrayList(candidateNames.length);
        var6 = candidateNames;
        var7 = candidateNames.length;

        for(var8 = 0; var8 < var7; ++var8) {
            beanName = var6[var8];
            if (!this.containsBeanDefinition(beanName) ||
this.getBeanDefinition(beanName).isAutowireCandidate()) {
                autowireCandidates.add(beanName);
            }
        }

        if (!autowireCandidates.isEmpty()) {
            candidateNames = StringUtils.toStringArray(autowireCandidates);
        }
    }

    if (candidateNames.length == 1) {
        String beanName = candidateNames[0];
        return new NamedBeanHolder(beanName, this.getBean(beanName,
requiredType.toClass(), args));
    } else {
        if (candidateNames.length > 1) {
            Map<String, Object> candidates = new
LinkedHashMap(candidateNames.length);
            var6 = candidateNames;
            var7 = candidateNames.length;

            for(var8 = 0; var8 < var7; ++var8) {
                beanName = var6[var8];
                if (this.containsSingleton(beanName) && args == null) {
                    // 创建实例的方法
                    Object beanInstance = this.getBean(beanName);
                    candidates.put(beanName, beanInstance instanceof NullBean ?
null : beanInstance);
                } else {
                    candidates.put(beanName, this.getType(beanName));
```

```
                }
            }

            String candidateName = this.determinePrimaryCandidate(candidates,
requiredType.toClass());
            if (candidateName == null) {
                candidateName =
this.determineHighestPriorityCandidate(candidates, requiredType.toClass());
            }

            if (candidateName != null) {
                Object beanInstance = candidates.get(candidateName);
                if (beanInstance == null || beanInstance instanceof Class) {
                    beanInstance = this.getBean(candidateName,
requiredType.toClass(), args);
                }

                return new NamedBeanHolder(candidateName, beanInstance);
            }

            if (!nonUniqueAsNull) {
                throw new NoUniqueBeanDefinitionException(requiredType,
candidates.keySet());
            }
        }

        return null;
    }
}
```

```
public Object getBean(String name) throws BeansException {
    return this.doGetBean(name, (Class)null, (Object[])null, false);
}
```

```
protected <T> T doGetBean(String name, @Nullable Class<T> requiredType,
@Nullable Object[] args, boolean typeCheckOnly) throws BeansException {
    String beanName = this.transformedBeanName(name);
    Object sharedInstance = this.getSingleton(beanName);
    Object bean;
    if (sharedInstance != null && args == null) {
        if (this.logger.isTraceEnabled()) {
            if (this.isSingletonCurrentlyInCreation(beanName)) {
                this.logger.trace("Returning eagerly cached instance of
singleton bean '" + beanName + "' that is not fully initialized yet - a
consequence of a circular reference");
            } else {
                this.logger.trace("Returning cached instance of singleton bean
'" + beanName + "'");
            }
        }

        bean = this.getObjectForBeanInstance(sharedInstance, name, beanName,
(RootBeanDefinition)null);
```

```
    } else {
        if (this.isPrototypeCurrentlyInCreation(beanName)) {
            throw new BeanCurrentlyInCreationException(beanName);
        }

        BeanFactory parentBeanFactory = this.getParentBeanFactory();
        if (parentBeanFactory != null && !this.containsBeanDefinition(beanName))
{
            String nameToLookup = this.originalBeanName(name);
            if (parentBeanFactory instanceof AbstractBeanFactory) {
                return
((AbstractBeanFactory)parentBeanFactory).doGetBean(nameToLookup, requiredType,
args, typeCheckOnly);
            }

            if (args != null) {
                return parentBeanFactory.getBean(nameToLookup, args);
            }

            if (requiredType != null) {
                return parentBeanFactory.getBean(nameToLookup, requiredType);
            }

            return parentBeanFactory.getBean(nameToLookup);
        }

        if (!typeCheckOnly) {
            this.markBeanAsCreated(beanName);
        }

        try {
            RootBeanDefinition mbd =
this.getMergedLocalBeanDefinition(beanName);
            this.checkMergedBeanDefinition(mbd, beanName, args);
            String[] dependsOn = mbd.getDependsOn();
            String[] var11;
            if (dependsOn != null) {
                var11 = dependsOn;
                int var12 = dependsOn.length;

                for(int var13 = 0; var13 < var12; ++var13) {
                    String dep = var11[var13];
                    if (this.isDependent(beanName, dep)) {
                        throw new
BeanCreationException(mbd.getResourceDescription(), beanName, "Circular depends-
on relationship between '" + beanName + "' and '" + dep + "'");
                    }

                    this.registerDependentBean(dep, beanName);

                    try {
                        this.getBean(dep);
                    } catch (NoSuchBeanDefinitionException var24) {
                        throw new
BeanCreationException(mbd.getResourceDescription(), beanName, "'" + beanName +
"' depends on missing bean '" + dep + "'", var24);
                    }
                }
```

```
            }

            if (mbd.isSingleton()) {
                sharedInstance = this.getSingleton(beanName, () -> {
                    try {
                        // 进入核心方法
                        return this.createBean(beanName, mbd, args);
                    } catch (BeansException var5) {
                        this.destroySingleton(beanName);
                        throw var5;
                    }
                });
                bean = this.getObjectForBeanInstance(sharedInstance, name,
beanName, mbd);
            } else if (mbd.isPrototype()) {
                var11 = null;

                Object prototypeInstance;
                try {
                    this.beforePrototypeCreation(beanName);
                    prototypeInstance = this.createBean(beanName, mbd, args);
                } finally {
                    this.afterPrototypeCreation(beanName);
                }

                bean = this.getObjectForBeanInstance(prototypeInstance, name,
beanName, mbd);
            } else {
                String scopeName = mbd.getScope();
                if (!StringUtils.hasLength(scopeName)) {
                    throw new IllegalStateException("No scope name defined for
bean ´" + beanName + "'");
                }

                Scope scope = (Scope)this.scopes.get(scopeName);
                if (scope == null) {
                    throw new IllegalStateException("No Scope registered for
scope name '" + scopeName + "'");
                }

                try {
                    Object scopedInstance = scope.get(beanName, () -> {
                        this.beforePrototypeCreation(beanName);

                        Object var4;
                        try {
                            var4 = this.createBean(beanName, mbd, args);
                        } finally {
                            this.afterPrototypeCreation(beanName);
                        }

                        return var4;
                    });
                    bean = this.getObjectForBeanInstance(scopedInstance, name,
beanName, mbd);
                } catch (IllegalStateException var23) {
```

```
                    throw new BeanCreationException(beanName, "Scope '" +
scopeName + "' is not active for the current thread; consider defining a scoped
proxy for this bean if you intend to refer to it from a singleton", var23);
                }
            }
        } catch (BeansException var26) {
            this.cleanupAfterBeanCreationFailure(beanName);
            throw var26;
        }
    }

    if (requiredType != null && !requiredType.isInstance(bean)) {
        try {
            T convertedBean = this.getTypeConverter().convertIfNecessary(bean,
requiredType);
            if (convertedBean == null) {
                throw new BeanNotOfRequiredTypeException(name, requiredType,
bean.getClass());
            } else {
                return convertedBean;
            }
        } catch (TypeMismatchException var25) {
            if (this.logger.isTraceEnabled()) {
                this.logger.trace("Failed to convert bean '" + name + "' to
required type '" + ClassUtils.getQualifiedName(requiredType) + "'", var25);
            }

            throw new BeanNotOfRequiredTypeException(name, requiredType,
bean.getClass());
        }
    } else {
        return bean;
    }
}
```

```
protected Object createBean(String beanName, RootBeanDefinition mbd, @Nullable
Object[] args) throws BeanCreationException {
    if (this.logger.isTraceEnabled()) {
        this.logger.trace("Creating instance of bean '" + beanName + "'");
    }

    RootBeanDefinition mbdToUse = mbd;
    Class<?> resolvedClass = this.resolveBeanClass(mbd, beanName, new Class[0]);
    if (resolvedClass != null && !mbd.hasBeanClass() && mbd.getBeanClassName()
!= null) {
        mbdToUse = new RootBeanDefinition(mbd);
        mbdToUse.setBeanClass(resolvedClass);
    }

    try {
        mbdToUse.prepareMethodOverrides();
    } catch (BeanDefinitionValidationException var9) {
```

```
            throw new
BeanDefinitionStoreException(mbdToUse.getResourceDescription(), beanName,
"Validation of method overrides failed", var9);
        }

        Object beanInstance;
        try {
            beanInstance = this.resolveBeforeInstantiation(beanName, mbdToUse);
            if (beanInstance != null) {
                return beanInstance;
            }
        } catch (Throwable var10) {
            throw new BeanCreationException(mbdToUse.getResourceDescription(),
beanName, "BeanPostProcessor before instantiation of bean failed", var10);
        }

        try {
            beanInstance = this.doCreateBean(beanName, mbdToUse, args);
            if (this.logger.isTraceEnabled()) {
                this.logger.trace("Finished creating instance of bean '" + beanName
+ "'");
            }

            return beanInstance;
        } catch (ImplicitlyAppearedSingletonException | BeanCreationException var7)
{
            throw var7;
        } catch (Throwable var8) {
            throw new BeanCreationException(mbdToUse.getResourceDescription(),
beanName, "Unexpected exception during bean creation", var8);
        }
    }
```

```
    protected Object doCreateBean(String beanName, RootBeanDefinition mbd,
@Nullable Object[] args) throws BeanCreationException {
            BeanWrapper instanceWrapper = null;
            if (mbd.isSingleton()) {
                instanceWrapper =
(BeanWrapper)this.factoryBeanInstanceCache.remove(beanName);
            }

            if (instanceWrapper == null) {
                // 创建实例的方法
                instanceWrapper = this.createBeanInstance(beanName, mbd, args);
            }

            Object bean = instanceWrapper.getWrappedInstance();
            Class<?> beanType = instanceWrapper.getWrappedClass();
            if (beanType != NullBean.class) {
                mbd.resolvedTargetType = beanType;
            }
```

```
        Object var7 = mbd.postProcessingLock;
        synchronized(mbd.postProcessingLock) {
            if (!mbd.postProcessed) {
                try {
                    this.applyMergedBeanDefinitionPostProcessors(mbd, beanType,
beanName);
                } catch (Throwable var17) {
                    throw new
BeanCreationException(mbd.getResourceDescription(), beanName, "Post-processing
of merged bean definition failed", var17);
                }

                mbd.postProcessed = true;
            }
        }

        boolean earlySingletonExposure = mbd.isSingleton() &&
this.allowCircularReferences && this.isSingletonCurrentlyInCreation(beanName);
        if (earlySingletonExposure) {
            if (this.logger.isTraceEnabled()) {
                this.logger.trace("Eagerly caching bean '" + beanName + "' to
allow for resolving potential circular references");
            }

            this.addSingletonFactory(beanName, () -> {
                return this.getEarlyBeanReference(beanName, mbd, bean);
            });
        }

        Object exposedObject = bean;

        try {
            this.populateBean(beanName, mbd, instanceWrapper);
            exposedObject = this.initializeBean(beanName, exposedObject, mbd);
        } catch (Throwable var18) {
            if (var18 instanceof BeanCreationException &&
beanName.equals(((BeanCreationException)var18).getBeanName())) {
                throw (BeanCreationException)var18;
            }

            throw new BeanCreationException(mbd.getResourceDescription(),
beanName, "Initialization of bean failed", var18);
        }

        if (earlySingletonExposure) {
            Object earlySingletonReference = this.getSingleton(beanName, false);
            if (earlySingletonReference != null) {
                if (exposedObject == bean) {
                    exposedObject = earlySingletonReference;
                } else if (!this.allowRawInjectionDespiteWrapping &&
this.hasDependentBean(beanName)) {
                    String[] dependentBeans = this.getDependentBeans(beanName);
                    Set<String> actualDependentBeans = new
LinkedHashSet(dependentBeans.length);
                    String[] var12 = dependentBeans;
                    int var13 = dependentBeans.length;

                    for(int var14 = 0; var14 < var13; ++var14) {
```

```
                        String dependentBean = var12[var14];
                        if
(!this.removeSingletonIfCreatedForTypeCheckOnly(dependentBean)) {
                            actualDependentBeans.add(dependentBean);
                        }
                    }

                    if (!actualDependentBeans.isEmpty()) {
                        throw new BeanCurrentlyInCreationException(beanName,
"Bean with name '" + beanName + "' has been injected into other beans [" +
StringUtils.collectionToCommaDelimitedString(actualDependentBeans) + "] in its
raw version as part of a circular reference, but has eventually been wrapped.
This means that said other beans do not use the final version of the bean. This
is often the result of over-eager type matching - consider using
'getBeanNamesForType' with the 'allowEagerInit' flag turned off, for example.");
                    }
                }
            }
        }

        try {
            this.registerDisposableBeanIfNecessary(beanName, bean, mbd);
            return exposedObject;
        } catch (BeanDefinitionValidationException var16) {
            throw new BeanCreationException(mbd.getResourceDescription(),
beanName, "Invalid destruction signature", var16);
        }
    }
```

可以看到最终是以反射的形式创建出来的对象

```
ıtiationException {

getDeclaringClass()) ? BeanUtils.KotlinDelegate.instantiateClass(ctor, args)   ctor.newInstance(args);

4):
```