# spring简介

|  |
| --- |
| 1.分离项目组件之间的依赖关系  2.简化企业开发  核心概念  IOC(inversion of control)：控制反转  DI(Dependency injection)：依赖注入  AOP(aspect orientend programming)：面向切面编程 |

# 使用log4j

|  |
| --- |
|  |

# 装配bean的三种方式

|  |
| --- |
| 自动装配  在xml中进行显示配置  在java中进行显示配置  隐式的bean发现机制和自动装配  组件扫描：  @Component:表示这个类需要在应用程式中被创建  @ComponentScan:自动发现应用程序中创建的类  自动装配  @Autowired: 自动满足bean之间的依赖  定义配置类  @Configuration: 表示当前类是一个配置类 |

# 使用单元测试

|  |
| --- |
| 引入Spring单元测试模块  maven: junit、 spring-test  @RunWith(SpringJUnit4ClassRunner.class)  加载配置类  @ContextConfiguration(classes=AppConfig.class) |

# @Autowired

|  |
| --- |
| 用在构造函数上(多个依赖的情况)  用在成员变量上  用在setter方法上  用在任意方法上  @Autowired(required=false)  表示注入的对象是可选的 |

# 处理自动装配的歧义性

|  |
| --- |
| 首选bean  在声明类的时候使用@Primary  只能定义一个@Primary  使用限定符  在声明的时候和装配的时候分别使用@Qualifier  使用限定符和bean id  在声明的时候指定bean的id(默认的id是首写字母小写的类名)  在装配的时候使用@Qualifier |

# 总结

|  |
| --- |
| 定义配置  @Configuration  @ComponentScan  分层架构中定义组件  @Controller  @Service  @Repostory  定义组件  @Component  @AutoWired  @AutoWired(required=false)  Spring测试环境  @RunWith  @ContextConfiguration  自定义装配歧义性  @Primary  @Qualifier  @Resource  @使用xml启用组件扫描  <context:component-scan base-package=""></context:component-scan> |

# 处理自动装配的歧义性

|  |
| --- |
| 首选bean  在声明类时候使用@Primary  只能定义一个@Primary  使用限定符  在声明的时候和装配的时候分别使用@Qualifier  使用限定符和bean id  在声明的时候指定bean的id(默认的id是首写字母小写的类名)  在装配的时候使用@Qualifier |

# 总结

|  |
| --- |
| JavaConfig  @Configuration  @Bean  依赖注入  构造函数  setter  任意方法  装配过程的歧义性 |

# 注解初始化spring容器(AnnotationConfigApplicationContext)

## pom.xml

|  |
| --- |
| *<?*xml version="1.0" encoding="UTF-8"*?>* <project xmlns="http://maven.apache.org/POM/4.0.0"  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">  <modelVersion>4.0.0</modelVersion>   <groupId>com.qfeda</groupId>  <artifactId>spring01quickstart</artifactId>  <version>1.0-SNAPSHOT</version>   <dependencies>  <dependency>  <groupId>org.springframework</groupId>  <artifactId>spring-context</artifactId>  <version>4.3.13.RELEASE</version>  </dependency>  </dependencies>  </project> |

## MessageService

|  |
| --- |
| package hello;  import org.springframework.stereotype.Component;  @Component public class MessageService {   public MessageService() {  super();  System.***out***.println("MessageService");  }   public String getMeaasge() {  return "Hello World";  } } |

## MessagePrinter

|  |
| --- |
| package hello;  import org.springframework.beans.factory.annotation.Autowired; import org.springframework.stereotype.Component;  @Component public class MessagePrinter {  public MessagePrinter() {  super();  System.***out***.println("MessagePrinter");  }   private MessageService service;   @Autowired  public void setService(MessageService service) {  this.service = service;  }   public void printerMeaasge() {  System.***out***.println(this.service.getMeaasge());  } } |

## Application

|  |
| --- |
| package hello;  public class Application {  public static void main(String[] args) {  System.***out***.println("application");   MessagePrinter printer = new MessagePrinter();   MessageService service = new MessageService();   printer.setService(service);   printer.printerMeaasge();  } } |

## ApplicationSpring

|  |
| --- |
| package hello;  import org.springframework.context.ApplicationContext; import org.springframework.context.annotation.AnnotationConfigApplicationContext; import org.springframework.context.annotation.ComponentScan;  // 扫描component注解 @ComponentScan public class ApplicationSpring {  public static void main(String[] args) {  System.***out***.println("applicationSpring");   // 初始化spring容器  ApplicationContext context = new AnnotationConfigApplicationContext(ApplicationSpring.class);  // 根据类名获取bean  MessagePrinter printer = context.getBean(MessagePrinter.class);  System.***out***.println(printer);  printer.printerMeaasge();  } } |

# xml配置初始化spring容器(ClassPathXmlApplicationContext)

## pom.xml

|  |
| --- |
| *<?*xml version="1.0" encoding="UTF-8"*?>* <project xmlns="http://maven.apache.org/POM/4.0.0"  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">  <modelVersion>4.0.0</modelVersion>   <groupId>com.qfeda</groupId>  <artifactId>spring02quickstart\_xml</artifactId>  <version>1.0-SNAPSHOT</version>   <dependencies>  <dependency>  <groupId>org.springframework</groupId>  <artifactId>spring-context</artifactId>  <version>4.3.13.RELEASE</version>  </dependency>   <dependency>  <groupId>log4j</groupId>  <artifactId>log4j</artifactId>  <version>1.2.17</version>  </dependency>  </dependencies> </project> |

## applicationContext.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">   <bean id="service" class="hello.MessageService" />   <bean id="printer" class="hello.MessagePrinter">  <property name="service" ref="service"></property>  </bean> </beans> |

## log4j.properties

|  |
| --- |
| log4j.rootLogger=INFO,CONSOLE,project log4j.appender.CONSOLE=org.apache.log4j.ConsoleAppender log4j.appender.CONSOLE.layout=org.apache.log4j.PatternLayout log4j.appender.CONSOLE.layout.ConversionPattern=%d [%t] %-5p %c(%L) - %m%n log4j.appender.project=org.apache.log4j.RollingFileAppender log4j.appender.project.file={xxxWeb.root}\\WEB-INF\\logs\\project\_ log4j.appender.DayRollingFile.DatePattern=yyyyMMdd'.log' log4j.appender.project.layout=org.apache.log4j.PatternLayout log4j.appender.project.layout.conversionPattern=%d [%t] %-5p %c(%L) - %m%n log4j.appender.project.MaxFileSize=1000KB log4j.appender.project.MaxBackupIndex=10 log4j.logger.org.springframework=INFO,CONSOLE,project |

## MessageService

|  |
| --- |
| package hello;  public class MessageService {   public MessageService() {  super();  System.***out***.println("MessageService");  }   public String getMeaasge() {  return "Hello World";  } } |

## MessagePrinter

|  |
| --- |
| package hello;  public class MessagePrinter {  public MessagePrinter() {  super();  System.***out***.println("MessagePrinter");  }   private MessageService service;   public void setService(MessageService service) {  this.service = service;  }   public void printerMeaasge() {  System.***out***.println(this.service.getMeaasge());  } } |

## ApplicationSpring

|  |
| --- |
| package hello;  import org.springframework.context.ApplicationContext; import org.springframework.context.support.ClassPathXmlApplicationContext;  public class ApplicationSpring {  public static void main(String[] args) {  System.***out***.println("applicationSpring");   // 初始化spring容器 xml配置形式  // ApplicationContext context = new AnnotationConfigApplicationContext(ApplicationSpring.class);  ApplicationContext context = new ClassPathXmlApplicationContext("applicationContext.xml");  MessagePrinter printer = context.getBean(MessagePrinter.class);  // MessageService service = context.getBean(MessageService.class);  System.***out***.println(printer);  // System.out.println(service);  // printer.setService(service);  printer.printerMeaasge();  } } |

# 配置类初始化spring容器

## pom.xml

|  |
| --- |
| *<?*xml version="1.0" encoding="UTF-8"*?>* <project xmlns="http://maven.apache.org/POM/4.0.0"  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">  <modelVersion>4.0.0</modelVersion>   <groupId>com.qfeda</groupId>  <artifactId>spring03quickstart\_auto</artifactId>  <version>1.0-SNAPSHOT</version>   <dependencies>  <dependency>  <groupId>org.springframework</groupId>  <artifactId>spring-context</artifactId>  <version>4.3.13.RELEASE</version>  </dependency>   <dependency>  <groupId>org.springframework</groupId>  <artifactId>spring-test</artifactId>  <version>4.3.13.RELEASE</version>  </dependency>   <dependency>  <groupId>log4j</groupId>  <artifactId>log4j</artifactId>  <version>1.2.17</version>  </dependency>   <dependency>  <groupId>junit</groupId>  <artifactId>junit</artifactId>  <version>4.12</version>  <scope>test</scope>  </dependency>  </dependencies> </project> |

## applicationContext.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">  </beans> |

## log4j.properties

|  |
| --- |
| log4j.rootLogger=INFO,CONSOLE,project log4j.appender.CONSOLE=org.apache.log4j.ConsoleAppender log4j.appender.CONSOLE.layout=org.apache.log4j.PatternLayout log4j.appender.CONSOLE.layout.ConversionPattern=%d [%t] %-5p %c(%L) - %m%n log4j.appender.project=org.apache.log4j.RollingFileAppender log4j.appender.project.file={xxxWeb.root}\\WEB-INF\\logs\\project\_ log4j.appender.DayRollingFile.DatePattern=yyyyMMdd'.log' log4j.appender.project.layout=org.apache.log4j.PatternLayout log4j.appender.project.layout.conversionPattern=%d [%t] %-5p %c(%L) - %m%n log4j.appender.project.MaxFileSize=1000KB log4j.appender.project.MaxBackupIndex=10 log4j.logger.org.springframework=INFO,CONSOLE,project |

## CompactDisc

|  |
| --- |
| package soundsystem;  import org.springframework.stereotype.Component;  @Component public class CompactDisc {   public CompactDisc() {  super();  System.***out***.println("CompactDisc构造函数无参");  }   public void play() {  System.***out***.println("正在播放音乐....");  } } |

## Power

|  |
| --- |
| package soundsystem;  import org.springframework.stereotype.Component;  @Component public class Power {  public Power() {  super();  }   public void supply() {  System.***out***.println("电源供电中.....");  } } |

## CDPlayer(@Autowired用法)

|  |
| --- |
| package soundsystem;  import org.springframework.beans.factory.annotation.Autowired; import org.springframework.stereotype.Component;  */\* \* 1.用在构造函数上，多个依赖的情况 \* 2.用在成员变量上 \* 3.用在setter方法上 \* 4.用在任意方法上 \* \*/* @Component public class CDPlayer {   // 用在成员变量上  @Autowired(required = false)  private CompactDisc cd;   @Autowired  private Power power;   */\*  // 用在setter方法上  @Autowired  public void setCd(CompactDisc cd) {  this.cd = cd;  System.out.println("调用setCd");  }   @Autowired  public void setPower(Power power) {  this.power = power;  System.out.println("调用setPower");   }\*/* public CDPlayer() {  super();  System.***out***.println("CDPlayer无参构造函数");  }   */\*  // 用在构造函数上，多个依赖的情况  @Autowired  public CDPlayer (CompactDisc cd) {  this.cd = cd;  System.out.println("CDPlayer有参构造函数");  }   // 用在构造函数上，多个依赖的情况  @Autowired  public CDPlayer(CompactDisc cd, Power power) {  this.cd = cd;  this.power = power;  System.out.println("CDPlayer多参构造函数");   }   // 用在任意方法上，多个依赖的情况  @Autowired  public void prepare(CompactDisc cd, Power power) {  this.cd = cd;  this.power = power;  System.out.println("调用prepare");  }\*/* public void play() {  power.supply();  if(cd != null) {  cd.play();  }  } } |

## AppConfig(配置类)

|  |
| --- |
| package soundsystem;  import org.springframework.context.annotation.ComponentScan; import org.springframework.context.annotation.Configuration;  @Configuration @ComponentScan public class AppConfig { } |

## App

|  |
| --- |
| package soundsystem;  import org.springframework.context.ApplicationContext; import org.springframework.context.annotation.AnnotationConfigApplicationContext;  public class App {  public static void main(String[] args) {  // 配置类初始化spring容器  ApplicationContext context = new AnnotationConfigApplicationContext(AppConfig.class);   CDPlayer player = context.getBean(CDPlayer.class);  player.play();  } } |

## AppTest

|  |
| --- |
| package soundsystem;  import org.junit.Test; import org.junit.internal.Classes; import org.junit.runner.RunWith; import org.springframework.beans.factory.annotation.Autowired; import org.springframework.context.ApplicationContext; import org.springframework.context.annotation.AnnotationConfigApplicationContext; import org.springframework.test.context.ContextConfiguration; import org.springframework.test.context.junit4.SpringJUnit4ClassRunner;  */\* \* 引入Spring单元测试模块 \* maven: junit、 spring-test \* @RunWith(SpringJUnit4ClassRunner.class) \* 加载配置类 \* @ContextConfiguration(classes=AppConfig.class) \* \*/* @RunWith(SpringJUnit4ClassRunner.class) @ContextConfiguration(classes=AppConfig.class) public class AppTest {   @Autowired  private CDPlayer player;   @Test  public void testPlay() {  // ApplicationContext context = new AnnotationConfigApplicationContext(AppConfig.class);  // CDPlayer player = context.getBean(CDPlayer.class);  player.play();  } } |

# 自动装配歧义性

|  |
| --- |
| 1.首选bean  在声明类的时候使用@Primary，只能定义一个@Primary  2.使用限定符  在声明的时候和装配的时候分别使用@Qualifier  3.使用限定符和bean id  在声明的时候知道bean的id(默认的id是首写字母小写的类名)  在装配的时候使用@Qualifier |

## pom.xml

|  |
| --- |
| *<?*xml version="1.0" encoding="UTF-8"*?>* <project xmlns="http://maven.apache.org/POM/4.0.0"  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">  <modelVersion>4.0.0</modelVersion>   <groupId>com.qfeda</groupId>  <artifactId>spring03quickstart\_interface</artifactId>  <version>1.0-SNAPSHOT</version>   <dependencies>  <dependency>  <groupId>org.springframework</groupId>  <artifactId>spring-context</artifactId>  <version>4.3.13.RELEASE</version>  </dependency>   <dependency>  <groupId>org.springframework</groupId>  <artifactId>spring-test</artifactId>  <version>4.3.13.RELEASE</version>  </dependency>   <dependency>  <groupId>log4j</groupId>  <artifactId>log4j</artifactId>  <version>1.2.17</version>  </dependency>   <dependency>  <groupId>junit</groupId>  <artifactId>junit</artifactId>  <version>4.12</version>  <scope>test</scope>  </dependency>  </dependencies> </project> |

## applicationContext.xml(通过xml启用组件扫描包)

|  |
| --- |
| *<?*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:context="http://www.springframework.org/schema/context"  xmlns:c="http://www.springframework.org/schema/c"  xmlns:util="http://www.springframework.org/schema/util"  xsi:schemaLocation="http://www.springframework.org/schema/beans  http://www.springframework.org/schema/beans/spring-beans.xsd  http://www.springframework.org/schema/util  http://www.springframework.org/schema/util/spring-util.xsd  http://www.springframework.org/schema/context  http://www.springframework.org/schema/context/spring-context.xsd">   <context:component-scan base-package="com.qfedu.demo"></context:component-scan>  <!--  或  <context:component-scan base-package="com.qfedu.demo.web"></context:component-scan>  <context:component-scan base-package="com.qfedu.demo.dao"></context:component-scan>  <context:component-scan base-package="com.qfedu.demo.service"></context:component-scan>  --> </beans> |

## log4j.properties

|  |
| --- |
| log4j.rootLogger=INFO,CONSOLE,project log4j.appender.CONSOLE=org.apache.log4j.ConsoleAppender log4j.appender.CONSOLE.layout=org.apache.log4j.PatternLayout log4j.appender.CONSOLE.layout.ConversionPattern=%d [%t] %-5p %c(%L) - %m%n log4j.appender.project=org.apache.log4j.RollingFileAppender log4j.appender.project.file={xxxWeb.root}\\WEB-INF\\logs\\project\_ log4j.appender.DayRollingFile.DatePattern=yyyyMMdd'.log' log4j.appender.project.layout=org.apache.log4j.PatternLayout log4j.appender.project.layout.conversionPattern=%d [%t] %-5p %c(%L) - %m%n log4j.appender.project.MaxFileSize=1000KB log4j.appender.project.MaxBackupIndex=10 log4j.logger.org.springframework=INFO,CONSOLE,project |

## UserDao

|  |
| --- |
| package com.qfedu.demo.dao;  public interface UserDao {  void add(); } |

## UserDaoNormal

|  |
| --- |
| package com.qfedu.demo.dao.impl;  import com.qfedu.demo.dao.UserDao; import org.springframework.stereotype.Component; import org.springframework.stereotype.Repository;  // @Component @Repository public class UserDaoNormal implements UserDao {  public void add() {  System.***out***.println("添加用户数据");  } } |

## UserService

|  |
| --- |
| package com.qfedu.demo.service;  public interface UserService {  void add(); } |

## UserController

|  |
| --- |
| package com.qfedu.demo.web;  import com.qfedu.demo.service.UserService; import org.springframework.beans.factory.annotation.Autowired; import org.springframework.beans.factory.annotation.Qualifier; import org.springframework.stereotype.Component; import org.springframework.stereotype.Controller;  // @Component @Controller public class UserController {  @Autowired  @Qualifier("userServiceNormal")  private UserService userService;   public void add() {  userService.add();  } } |

## UserServiceNormal(解决自动装配歧义性)

|  |
| --- |
| package com.qfedu.demo.service.impl;  import com.qfedu.demo.dao.UserDao; import com.qfedu.demo.service.UserService; import org.springframework.beans.factory.annotation.Autowired; import org.springframework.beans.factory.annotation.Qualifier; import org.springframework.stereotype.Component; import org.springframework.stereotype.Service;   @Service // @Component // @Component("normal") // @Qualifier("normal") public class UserServiceNormal implements UserService {   @Autowired  private UserDao userDao;   public void add() {  System.***out***.println("添加用户");  userDao.add();  } } |

## UserServiceFestival(解决自动装配歧义性)

|  |
| --- |
| package com.qfedu.demo.service.impl;  import com.qfedu.demo.service.UserService; import org.springframework.beans.factory.annotation.Qualifier; import org.springframework.context.annotation.Primary; import org.springframework.stereotype.Component;   @Component // @Component("festival") // @Primary // 首选bean // @Qualifier("festival")使用限定符 public class UserServiceFestival implements UserService {  public void add() {  System.***out***.println("注册用户发送优惠券");  } } |

## AppConfig(设置组件扫描的基础包)

|  |
| --- |
| package com.qfedu.demo.config;  import com.qfedu.demo.dao.UserDao; import com.qfedu.demo.service.UserService; import com.qfedu.demo.web.UserController; import org.springframework.context.annotation.ComponentScan; import org.springframework.context.annotation.Configuration;   @Configuration // @ComponentScan("com.qfedu.demo") // @ComponentScan(basePackages = {"com.qfedu.demo.web","com.qfedu.demo.dao","com.qfedu.demo.service"}) @ComponentScan(basePackageClasses = {UserController.class, UserService.class, UserDao.class}) public class AppConfig { } |

## UserServiceTest

|  |
| --- |
| package com.qfedu.demo.service;  import com.qfedu.demo.config.AppConfig; import org.junit.Test; import org.junit.runner.RunWith; import org.springframework.test.context.ContextConfiguration; import org.springframework.test.context.junit4.SpringJUnit4ClassRunner;  import javax.annotation.Resource;   @RunWith(SpringJUnit4ClassRunner.class) @ContextConfiguration(classes= AppConfig.class) public class UserServiceTest {   */\*  \*@Resource = @Autowired + @Qualifier  \*@Autowired  \*@Qualifier("userServiceFestival")  \*/* @Resource(name = "userServiceFestival")  private UserService userService;   // 自动装配歧义性解决：具体类型UserServiceNormal  // private UserServiceNormal userService;   @Test  public void testAdd() {  userService.add();  } } |

## UserControllerTest(@ContextConfiguration使用)

|  |
| --- |
| package com.qfedu.demo.web;  import org.junit.Test; import org.junit.runner.RunWith; import org.springframework.beans.factory.annotation.Autowired; import org.springframework.test.context.ContextConfiguration; import org.springframework.test.context.junit4.SpringJUnit4ClassRunner;  @RunWith(SpringJUnit4ClassRunner.class) // @ContextConfiguration(classes= AppConfig.class) @ContextConfiguration("classpath:applicationContext.xml") public class UserControllerTest {   @Autowired  private UserController userController;   @Test  public void testAdd() {  userController.add();  } } |

# 在javaconfig中配置bean对象

## pom.xml

|  |
| --- |
| *<?*xml version="1.0" encoding="UTF-8"*?>* <project xmlns="http://maven.apache.org/POM/4.0.0"  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">  <modelVersion>4.0.0</modelVersion>   <groupId>com.qfeda</groupId>  <artifactId>spring05javaconfig</artifactId>  <version>1.0-SNAPSHOT</version>  <dependencies>  <dependency>  <groupId>org.springframework</groupId>  <artifactId>spring-context</artifactId>  <version>4.3.13.RELEASE</version>  </dependency>   <dependency>  <groupId>org.springframework</groupId>  <artifactId>spring-test</artifactId>  <version>4.3.13.RELEASE</version>  </dependency>   <dependency>  <groupId>log4j</groupId>  <artifactId>log4j</artifactId>  <version>1.2.17</version>  </dependency>   <dependency>  <groupId>junit</groupId>  <artifactId>junit</artifactId>  <version>4.12</version>  <scope>test</scope>  </dependency>  </dependencies>  </project> |

## applicationContext.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 http://www.springframework.org/schema/beans/spring-beans.xsd">   </beans> |

## log4j.properties

|  |
| --- |
| log4j.rootLogger=INFO,CONSOLE,project log4j.appender.CONSOLE=org.apache.log4j.ConsoleAppender log4j.appender.CONSOLE.layout=org.apache.log4j.PatternLayout log4j.appender.CONSOLE.layout.ConversionPattern=%d [%t] %-5p %c(%L) - %m%n log4j.appender.project=org.apache.log4j.RollingFileAppender log4j.appender.project.file={xxxWeb.root}\\WEB-INF\\logs\\project\_ log4j.appender.DayRollingFile.DatePattern=yyyyMMdd'.log' log4j.appender.project.layout=org.apache.log4j.PatternLayout log4j.appender.project.layout.conversionPattern=%d [%t] %-5p %c(%L) - %m%n log4j.appender.project.MaxFileSize=1000KB log4j.appender.project.MaxBackupIndex=10 log4j.logger.org.springframework=INFO,CONSOLE,project |

## UserDao

|  |
| --- |
| package com.qfedu.demo.dao;  public interface UserDao {  void add(); } |

## UserDaoNormal

|  |
| --- |
| package com.qfedu.demo.dao.impl;  import com.qfedu.demo.dao.UserDao; import org.springframework.stereotype.Repository;  // @Repository public class UserDaoNormal implements UserDao {  public void add() {  System.***out***.println("添加用户到数据库中.........");  } } |

## UserDaoCache

|  |
| --- |
| package com.qfedu.demo.dao.impl;  import com.qfedu.demo.dao.UserDao;  public class UserDaoCache implements UserDao {  public void add() {  System.***out***.println("添加用户dao到缓存中.........");  } } |

## UserService

|  |
| --- |
| package com.qfedu.demo.service;  public interface UserService {  void add(); } |

## UserServiceNormal

|  |
| --- |
| package com.qfedu.demo.service.impl;  import com.qfedu.demo.dao.UserDao; import com.qfedu.demo.service.UserService;  public class UserServiceNormal implements UserService {   private UserDao userDao;    // 无参构造函数  public UserServiceNormal() {  super();  }    // 有参构造函数  public UserServiceNormal(UserDao userDao) {  this.userDao = userDao;  System.***out***.println("UserServiceNormal构造函数");  }    // setter方法  public void setUserDao(UserDao userDao) {  this.userDao = userDao;  System.***out***.println("setUserDao");  }    // 普通函数  public void prepare(UserDao userDao) {  this.userDao = userDao;  System.***out***.println("prepare");  }   public void add() {  userDao.add();  } } |

## AppConfig

|  |
| --- |
| package com.qfedu.demo.config;  import com.qfedu.demo.dao.UserDao; import com.qfedu.demo.dao.impl.UserDaoCache; import com.qfedu.demo.dao.impl.UserDaoNormal; import com.qfedu.demo.service.UserService; import com.qfedu.demo.service.impl.UserServiceNormal; import org.springframework.beans.factory.annotation.Qualifier; import org.springframework.context.annotation.Bean; import org.springframework.context.annotation.ComponentScan; import org.springframework.context.annotation.Configuration; import org.springframework.context.annotation.Primary;  @Configuration // @ComponentScan // 不用扫描 public class AppConfig {   @Bean  // @Primary  // @Qualifier("normal")  // @Bean("normal")  public UserDao userDaoNormal() {  System.***out***.println("创建userDaoNormal对象");  return new UserDaoNormal();  }   @Bean  // @Primary  // @Bean("cache")  // @Qualifier("cache")  public UserDao userDaoCache() {  System.***out***.println("创建userDaoCache对象");  return new UserDaoCache();  }   @Bean  public UserService userServiceNormal( @Qualifier("userDaoCache")UserDao userDao) {  System.***out***.println("创建UserService对象");  // UserDao userDao = userDaoNormal();  // return new UserServiceNormal(userDao);   UserServiceNormal userService = new UserServiceNormal();  // userService.setUserDao(userDao);  userService.prepare(userDao);  return userService;  } } |

## UserDaoTest

|  |
| --- |
| package com.qfedu.demo.dao;  import com.qfedu.demo.config.AppConfig; import org.junit.Test; import org.junit.runner.RunWith; import org.springframework.beans.factory.annotation.Autowired; import org.springframework.test.context.ContextConfiguration; import org.springframework.test.context.junit4.SpringJUnit4ClassRunner;   @RunWith(SpringJUnit4ClassRunner.class) @ContextConfiguration(classes= AppConfig.class) public class UserDaoTest {  @Autowired  private UserDao userDao;   @Test  public void testAdd() {  userDao.add();  } } |

## UserServiceTest

|  |
| --- |
| package com.qfedu.demo.service;  import com.qfedu.demo.config.AppConfig; import org.junit.Test; import org.junit.runner.RunWith; import org.springframework.beans.factory.annotation.Autowired; import org.springframework.test.context.ContextConfiguration; import org.springframework.test.context.junit4.SpringJUnit4ClassRunner;   @RunWith(SpringJUnit4ClassRunner.class) @ContextConfiguration(classes= AppConfig.class) public class UserServiceTest {   @Autowired  private UserService userService;   @Test  public void testAdd() {  userService.add();  } } |

# applicationContext.xml配置规范

## 获取bean方式

|  |
| --- |
| ClassPathXmlApplicationContext context = new ClassPathXmlApplicationContext("applicationContext.xml");  // 单个bean时获取 CompactDisc cd = context.getBean(CompactDisc.class)  // 多个bean获取，没有id和name时，根据全类名加上#0获取 CompactDisc cd = (CompactDisc) context.getBean("com.qfedu.demo.soundsystem.CompactDisc#0")  // 多个bean获取，根据id或name获取 CompactDisc cd = (CompactDisc) context.getBean("compactDisc1"); cd.play();  <bean class="com.qfedu.demo.soundsystem.CompactDisc" /> <bean class="com.qfedu.demo.soundsystem.CompactDisc" />  <bean id="compactDisc1" class="com.qfedu.demo.soundsystem.CompactDisc" /> <bean id="compactDisc2" class="com.qfedu.demo.soundsystem.CompactDisc" />  name和id区别：name可以有多个，name属性可以用空格、逗号、分号隔开 <bean name="compactDisc1 compactDisc12" class="com.qfedu.demo.soundsystem.CompactDisc" /> <bean name="compactDisc2" class="com.qfedu.demo.soundsystem.CompactDisc" /> |

## 构造函数注入

|  |
| --- |
| <bean id="cdPlayer1" class="com.qfedu.demo.soundsystem.CDPlayer">  <constructor-arg ref="compactDisc1" /> </bean>  <bean id="compactDisc2"  class="com.qfedu.demo.soundsystem.CompactDisc"  c:title="I Do"  c:artist="李四"/>  <bean id="compactDisc3"  class="com.qfedu.demo.soundsystem.CompactDisc"  c:\_0="I Do"  c:\_1="王五"/> |

## 构造函数c命名空间注入

|  |
| --- |
| <bean id="cdPlayer2"  class="com.qfedu.demo.soundsystem.CDPlayer"  c:cd-ref="compactDisc2"/>  或  <bean id="cdPlayer3"  class="com.qfedu.demo.soundsystem.CDPlayer"  c:\_0-ref="compactDisc2"/> |

## 构造函数注入简单类型

|  |
| --- |
| <bean id="compactDisc1" class="com.qfedu.demo.soundsystem.CompactDisc">  <constructor-arg name="title" value="I Do"/>  <constructor-arg name="artist" value="张三"/>  <!--或-->  <constructor-arg index="0" value="I Do"/>  <constructor-arg index="1" value="张三"/>  <!--或-->  <constructor-arg type="java.lang.String" value="I Do"/>  <constructor-arg type="java.lang.String" value="张三"/> </bean> |

## 构造函数注入list、set、map、数组类型

|  |
| --- |
| <bean id="music1" class="com.qfedu.demo.soundsystem.Music">  <constructor-arg value="I Do 1"/>  <constructor-arg value="270" /> </bean> <bean id="music2" class="com.qfedu.demo.soundsystem.Music">  <constructor-arg value="I Do 2"/>  <constructor-arg value="280"/> </bean> <bean id="music3" class="com.qfedu.demo.soundsystem.Music">  <constructor-arg value="I Do 3"/>  <constructor-arg value="290"/> </bean>  <bean id="compactDisc1" class="com.qfedu.demo.soundsystem.CompactDisc">  <constructor-arg name="title" value="I Do"/>  <constructor-arg name="artist" value="张三"/>  <constructor-arg name="tracks">  <list>  <value>I Do 1</value>  <value>I Do 2</value>  <value>I Do 3</value>  </list>  <list>  <ref bean="music1"/>  <ref bean="music2"/>  <ref bean="music3"/>  </list>  <set>  <ref bean="music2"/>  <ref bean="music1"/>  <ref bean="music3"/>  <ref bean="music3"/>  </set>  <map>  <!--<entry key="" value="" />-->  <entry key="m1" value-ref="music1" />  <entry key="m2" value-ref="music2" />  <entry key="m3" value-ref="music3" />  </map>  <array>  <!--<value>I Do</value>-->  <ref bean="music1"/>  <ref bean="music2"/>  <ref bean="music3"/>  </array>  </constructor-arg>  </bean> |

## setter方法注入属性

|  |
| --- |
| <bean id="music1" class="com.qfedu.demo.soundsystem.Music">  <property name="title" value="告白气球" />  <property name="duration" value="215" /> </bean>  <bean id="music2"  class="com.qfedu.demo.soundsystem.Music"  p:title="爱情废柴"  p:duration="305">  <!--<property name="title" value="爱情废柴" />  <property name="duration" value="305" />--> </bean> |

## setter方法注入属性注入数组和列表

|  |
| --- |
| <util:list id="trackList">  <ref bean="music1"/>  <ref bean="music2"/> </util:list>  <util:map></util:map> <util:set></util:set>  <bean id="compactDisc1"  class="com.qfedu.demo.soundsystem.CompactDisc"  p:title="爱情买卖12"  p:artist="周杰伦"  p:tracks-ref="trackList">  <!-- <property name="title" value="爱情买卖" />  <property name="artist" value="周杰伦" />  <property name="tracks">  <array>  <ref bean="music1"/>  <ref bean="music2"/>  </array>  </property>--> </bean>  <!--属性注入中注入对象的引用--> <bean id="CDPlayre1"  class="com.qfedu.demo.soundsystem.CDPlayer"  p:cd-ref="compactDisc1">  <!--<property name="cd" ref="compactDisc1" />--> </bean> |

## setter方法注入对象引用

|  |
| --- |
| <!--属性注入中注入对象的引用--> <bean id="CDPlayre1"  class="com.qfedu.demo.soundsystem.CDPlayer"  p:cd-ref="compactDisc1">  <!--<property name="cd" ref="compactDisc1" />--> </bean> |

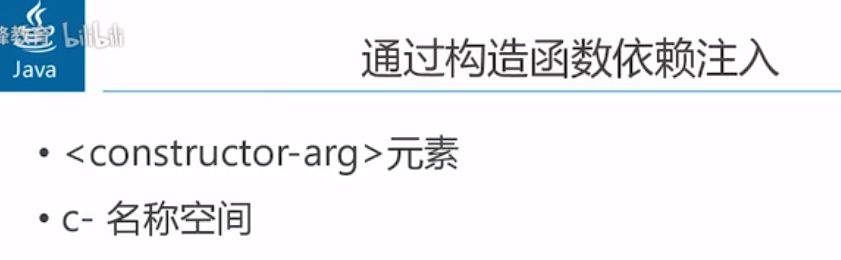
## p名称空间注入

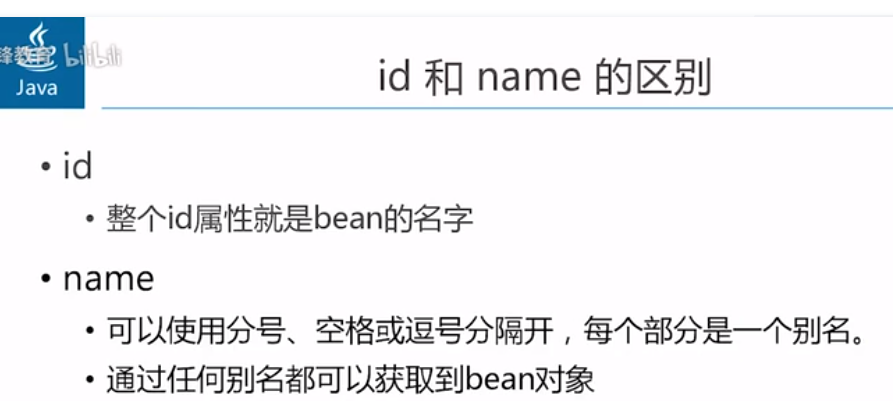
|  |
| --- |
| <bean id="music2"  class="com.qfedu.demo.soundsystem.Music"  p:title="爱情废柴"  p:duration="305">  <!--<property name="title" value="爱情废柴" />  <property name="duration" value="305" />--> </bean> |

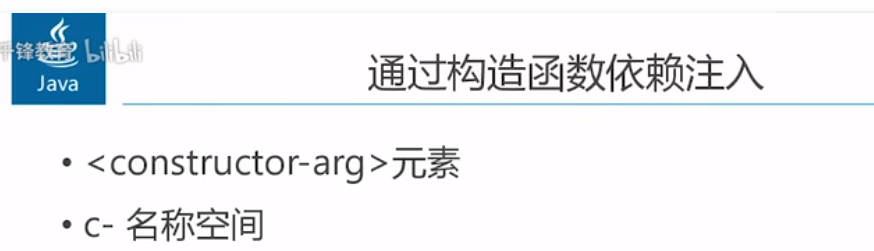
## util名称空间注入

|  |
| --- |
| <util:list id="trackList">  <ref bean="music1"/>  <ref bean="music2"/> </util:list>  <util:map></util:map> <util:set></util:set> |

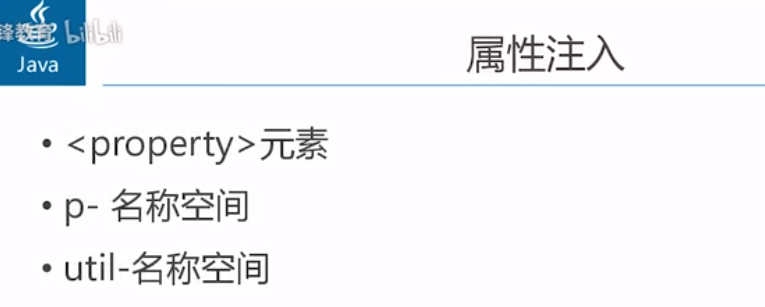














## pom.xml

|  |
| --- |
| *<?*xml version="1.0" encoding="UTF-8"*?>* <project xmlns="http://maven.apache.org/POM/4.0.0"  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">  <modelVersion>4.0.0</modelVersion>   <groupId>com.qfeda</groupId>  <artifactId>spring06xml</artifactId>  <version>1.0-SNAPSHOT</version>   <dependencies>  <dependency>  <groupId>org.springframework</groupId>  <artifactId>spring-context</artifactId>  <version>4.3.13.RELEASE</version>  </dependency>   <dependency>  <groupId>org.springframework</groupId>  <artifactId>spring-test</artifactId>  <version>4.3.13.RELEASE</version>  </dependency>   <dependency>  <groupId>log4j</groupId>  <artifactId>log4j</artifactId>  <version>1.2.17</version>  </dependency>   <dependency>  <groupId>junit</groupId>  <artifactId>junit</artifactId>  <version>4.12</version>  <scope>test</scope>  </dependency>  </dependencies> </project> |

## log4j.properties

|  |
| --- |
| log4j.rootLogger=INFO,CONSOLE,project log4j.appender.CONSOLE=org.apache.log4j.ConsoleAppender log4j.appender.CONSOLE.layout=org.apache.log4j.PatternLayout log4j.appender.CONSOLE.layout.ConversionPattern=%d [%t] %-5p %c(%L) - %m%n log4j.appender.project=org.apache.log4j.RollingFileAppender log4j.appender.project.file={xxxWeb.root}\\WEB-INF\\logs\\project\_ log4j.appender.DayRollingFile.DatePattern=yyyyMMdd'.log' log4j.appender.project.layout=org.apache.log4j.PatternLayout log4j.appender.project.layout.conversionPattern=%d [%t] %-5p %c(%L) - %m%n log4j.appender.project.MaxFileSize=1000KB log4j.appender.project.MaxBackupIndex=10 log4j.logger.org.springframework=INFO,CONSOLE,project |

## CompactDisc

|  |
| --- |
| package com.qfedu.demo.soundsystem;  import java.util.List; import java.util.Map;  public class CompactDisc {   private String title;  private String artist;  // private List<String> tracks;  // private List<Music> tracks;  // private Map<String, Music> tracks;  private Music[] tracks;   public String getTitle() {  return title;  }   public void setTitle(String title) {  this.title = title;  System.***out***.println("--在" + this.toString() + "中注入title");  }   public String getArtist() {  return artist;  }   public void setArtist(String artist) {  this.artist = artist;  System.***out***.println("--在" + this.toString() + "中注入artist");  }   public Music[] getTracks() {  return tracks;  }   public void setTracks(Music[] tracks) {  this.tracks = tracks;  System.***out***.println("--在" + this.toString() + "中注入setTracks");  }   public CompactDisc() {  super();  System.***out***.println("CompactDisc构造函数..." + this.toString());  }   public CompactDisc(String title, String artist) {  this.title = title;  this.artist = artist;  System.***out***.println("CompactDisc有2参构造函数..." + this.toString());   }   */\* public CompactDisc(String title, String artist, List<String> tracks) {  this.title = title;  this.artist = artist;  this.tracks = tracks;  System.out.println("CompactDisc有3参构造函数..." + this.toString());  }\*/   /\*public CompactDisc(String title, String artist, List<Music> tracks) {  this.title = title;  this.artist = artist;  this.tracks = tracks;  System.out.println("CompactDisc有3参构造函数..." + this.toString());  }\*/   /\* public CompactDisc(String title, String artist, Map<String, Music> tracks) {  this.title = title;  this.artist = artist;  this.tracks = tracks;  System.out.println("CompactDisc有3参构造函数..." + this.toString());  }\*/* public CompactDisc(String title, String artist, Music[] tracks) {  this.title = title;  this.artist = artist;  this.tracks = tracks;  System.***out***.println("CompactDisc有3参构造函数..." + this.toString());  }   public void play() {  System.***out***.println("正在播放音乐...." + this.toString() +" " + this.title + " by " + this.artist);  */\* for (String track: this.tracks) {  System.out.println("音乐：" + track);  }   for (Music track: this.tracks) {  System.out.println("音乐：" + track.getTitle() + ".时长：" + track.getDuration());  }\*/   /\*for (String key: this.tracks.keySet()) {  System.out.println("key：" + key);  Music music = this.tracks.get(key);  System.out.println("音乐：" + music.getTitle() + ".时长：" + music.getDuration());  }\*/* for (Music track: this.tracks) {  System.***out***.println("音乐：" + track.getTitle() + ".时长：" + track.getDuration());  }  }  } |

## applicationContext.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"  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">  <bean id="compactDisc" class="com.qfedu.demo.soundsystem.CompactDisc" />  </beans> |

## ApplicationSpring

|  |
| --- |
| import com.qfedu.demo.soundsystem.CompactDisc; import org.springframework.context.support.ClassPathXmlApplicationContext;    public class ApplicationSpring {  public static void main(String[] args) {  System.***out***.println("ApplicationSpring is running......");  ClassPathXmlApplicationContext context = new ClassPathXmlApplicationContext("applicationContext-property.xml");  // CompactDisc cd = context.getBean(CompactDisc.class);  // CompactDisc cd = (CompactDisc) context.getBean("com.qfedu.demo.soundsystem.CompactDisc#0");  // CompactDisc cd = (CompactDisc) context.getBean("compactDisc1");  CompactDisc cd = (CompactDisc) context.getBean("compactDisc12");  cd.play();  } } |

# 工厂方法创建bean对象

## pom.xml

|  |
| --- |
| *<?*xml version="1.0" encoding="UTF-8"*?>* <project xmlns="http://maven.apache.org/POM/4.0.0"  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">  <modelVersion>4.0.0</modelVersion>   <groupId>com.qfeda</groupId>  <artifactId>spring07factory</artifactId>  <version>1.0-SNAPSHOT</version>   <dependencies>  <dependency>  <groupId>org.springframework</groupId>  <artifactId>spring-context</artifactId>  <version>4.3.13.RELEASE</version>  </dependency>   <dependency>  <groupId>org.springframework</groupId>  <artifactId>spring-test</artifactId>  <version>4.3.13.RELEASE</version>  </dependency>   <dependency>  <groupId>log4j</groupId>  <artifactId>log4j</artifactId>  <version>1.2.17</version>  </dependency>   <dependency>  <groupId>junit</groupId>  <artifactId>junit</artifactId>  <version>4.12</version>  <scope>test</scope>  </dependency>  </dependencies> </project> |

## applicationContext.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:p="http://www.springframework.org/schema/p"  xmlns:context="http://www.springframework.org/schema/context"  xmlns:c="http://www.springframework.org/schema/c"  xmlns:util="http://www.springframework.org/schema/util"  xsi:schemaLocation="http://www.springframework.org/schema/beans  http://www.springframework.org/schema/beans/spring-beans.xsd  http://www.springframework.org/schema/util  http://www.springframework.org/schema/util/spring-util.xsd  http://www.springframework.org/schema/context  http://www.springframework.org/schema/context/spring-context.xsd">    <!-- 静态工厂-->  <bean id="person1"  class="com.qfedu.demo.PersonFactory"  factory-method="createPerson1" />  <!--实例工厂-->  <bean id="personFactory"  class="com.qfedu.demo.PersonFactory"/>   <bean id="person2"  factory-bean="personFactory"  factory-method="createPerson2" /> </beans> |

## log4j.properties

|  |
| --- |
| log4j.rootLogger=INFO,CONSOLE,project log4j.appender.CONSOLE=org.apache.log4j.ConsoleAppender log4j.appender.CONSOLE.layout=org.apache.log4j.PatternLayout log4j.appender.CONSOLE.layout.ConversionPattern=%d [%t] %-5p %c(%L) - %m%n log4j.appender.project=org.apache.log4j.RollingFileAppender log4j.appender.project.file={xxxWeb.root}\\WEB-INF\\logs\\project\_ log4j.appender.DayRollingFile.DatePattern=yyyyMMdd'.log' log4j.appender.project.layout=org.apache.log4j.PatternLayout log4j.appender.project.layout.conversionPattern=%d [%t] %-5p %c(%L) - %m%n log4j.appender.project.MaxFileSize=1000KB log4j.appender.project.MaxBackupIndex=10 log4j.logger.org.springframework=INFO,CONSOLE,project |

## Person

|  |
| --- |
| package com.qfedu.demo;  public class Person { } |

## PersonFactory

|  |
| --- |
| package com.qfedu.demo;  */\* \* 静态工厂方法 \* \*/* public class PersonFactory {  public static Person createPerson1() {  System.***out***.println("静态工厂创建Person");  return new Person();  }   public Person createPerson2() {  System.***out***.println("实例工厂创建Person");  return new Person();  } } |

## PersonFactoryTest

|  |
| --- |
| package com.qfedu.demo;  import org.junit.Test; import org.junit.runner.RunWith; import org.springframework.beans.factory.annotation.Autowired; import org.springframework.test.context.ContextConfiguration; import org.springframework.test.context.junit4.SpringJUnit4ClassRunner;   @RunWith(SpringJUnit4ClassRunner.class) @ContextConfiguration("classpath:applicationContext.xml") public class PersonFactoryTest {  @Autowired  Person person1;  @Autowired  Person person2;   @Test  public void tetst01() {  System.***out***.println(person1);  System.***out***.println(person2);  } } |

# idea创建spring的maven项目

|  |
| --- |
| 1.    2.    3.    4.引入  *<?*xml version="1.0" encoding="UTF-8"*?>* <project xmlns="http://maven.apache.org/POM/4.0.0"  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0  http://maven.apache.org/xsd/maven-4.0.0.xsd">  <modelVersion>4.0.0</modelVersion>   <groupId>com.shipeng</groupId>  <artifactId>spring-demo</artifactId>  <version>1.0-SNAPSHOT</version>  <dependencies>  <dependency>  <groupId>org.springframework</groupId>  <artifactId>spring-context</artifactId>  <version>4.3.13.RELEASE</version>  </dependency>   <dependency>  <groupId>org.springframework</groupId>  <artifactId>spring-test</artifactId>  <version>4.3.13.RELEASE</version>  </dependency>   <dependency>  <groupId>log4j</groupId>  <artifactId>log4j</artifactId>  <version>1.2.17</version>  </dependency>   <dependency>  <groupId>junit</groupId>  <artifactId>junit</artifactId>  <version>4.12</version>  <scope>test</scope>  </dependency>  </dependencies> </project> |

# ApplicationContext和BeanFactory

|  |
| --- |
|  |

# 获取IOC对象

# bean的依赖注入

## setter方法注入

|  |
| --- |
| 先调用无参构造函数，再调用属性的setUser方法赋值 属性注入是实际应用中最常用的注入方式  1.属性注入即通过 setter 方法注入bean 的属性值或依赖的对象  2.属性注入使用 <property> 元素, 使用 name 属性指定 bean 的属性名称，  value 属性或 <value> 子节点指定属性值  id: bean的名称，在 IOC 容器中必须是唯一的,  若 id 没有指定，Spring 自动将权限定性类名作为 bean的名字。  id 可以指定多个名字，名字之间可用逗号、分号、或空格分隔 class: 通过全类名方式配置bean class="com.atguigu.spring.helloworld.HelloWorld" |

### Car

|  |
| --- |
| package com.shipeng.spring;  public class Car {  private String brand;  private String corp;  private double price;  private int maxSpeed;   public String getBrand() {  return brand;  }   public void setBrand(String brand) {  this.brand = brand;  System.***out***.println("setBrand方法...");  }   public void setCorp(String corp) {  this.corp = corp;  System.***out***.println("setCorp方法...");  }   public void setPrice(double price) {  this.price = price;  System.***out***.println("setPrice方法...");  }   public void setMaxSpeed(int maxSpeed) {  this.maxSpeed = maxSpeed;  System.***out***.println("setMaxSpeed方法...");  }   public Car() {  super();  System.***out***.println("无参构造函数");  }   @Override  public String toString() {  return "Car{" +  "brand='" + brand + '\'' +  ", corp='" + corp + '\'' +  ", price=" + price +  ", maxSpeed=" + maxSpeed +  '}';  } } |

### applicationContext.xml

|  |
| --- |
| <bean id="car" class="com.shipeng.spring.beans.Car">  <property name="brand" value="大众"></property>  <property name="corp" value="奥迪"></property>  <property name="price" value="5000"></property>  <property name="maxSpeed" value="6000"></property> </bean> |

### App

|  |
| --- |
| package com.shipeng;  import com.shipeng.spring.beans.Car; import org.springframework.context.ApplicationContext; import org.springframework.context.support.ClassPathXmlApplicationContext;  public class App {  public static void main(String[] args) {  ApplicationContext context = new ClassPathXmlApplicationContext("applicationContext.xml");   Car car = (Car) context.getBean("car");   System.***out***.println(car);  }  } |

## 构造函数注入

|  |
| --- |
| 1.通过构造方法注入bean 的属性值或依赖的对象，它保证了bean实例在实例化后就可以使用。  2.构造器注入在 <constructor-arg> 元素里声明属性,<constructor-arg> 中是没有 name 属性的  若一个 bean 有多个构造器, 如何通过构造器来为 bean 的属性赋值？  使用构造函数注入属性值时，可以指定参数的位置(index)和类型(type)，以区分重载的构造函数 |

### Car

|  |
| --- |
| package com.shipeng.spring;  public class Car {  private String brand;  private String corp;  private double price;  private int maxSpeed;   public Car(String brand, String corp, double price, int maxSpeed) {  this.brand = brand;  this.corp = corp;  this.price = price;  this.maxSpeed = maxSpeed;  System.***out***.println("price构造函数...");  }   @Override  public String toString() {  return "Car{" +  "brand='" + brand + '\'' +  ", corp='" + corp + '\'' +  ", price=" + price +  ", maxSpeed=" + maxSpeed +  '}';  } } |

### applicationContext.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:p="http://www.springframework.org/schema/p"  xmlns:context="http://www.springframework.org/schema/context"  xmlns:c="http://www.springframework.org/schema/c"  xmlns:util="http://www.springframework.org/schema/util"  xsi:schemaLocation="http://www.springframework.org/schema/beans  http://www.springframework.org/schema/beans/spring-beans.xsd  http://www.springframework.org/schema/util  http://www.springframework.org/schema/util/spring-util.xsd  http://www.springframework.org/schema/context  http://www.springframework.org/schema/context/spring-context.xsd">   <bean id="car" class="com.shipeng.spring.beans.Car">  <!--根据顺序匹配-->  <constructor-arg value="大众"></constructor-arg>  <constructor-arg value="奥迪"></constructor-arg>  <constructor-arg value="5000"></constructor-arg>  <constructor-arg value="6000"></constructor-arg>   <!--按索引匹配-->  <constructor-arg value="大众" index="0"></constructor-arg>  <constructor-arg value="奥迪" index="1"></constructor-arg>  <constructor-arg value="5000" index="2"></constructor-arg>  <constructor-arg value="6000" index="3"></constructor-arg>   <!-- 按类型匹配-->  <constructor-arg value="大众" type="java.lang.String"></constructor-arg>  <constructor-arg value="奥迪" type="java.lang.String"></constructor-arg>  <constructor-arg value="5000" type="int"></constructor-arg>  <constructor-arg value="6000" type="double"></constructor-arg>   <!--混合使用-->  <constructor-arg value="大众" index="0"></constructor-arg>  <constructor-arg value="奥迪" index="1"></constructor-arg>  <constructor-arg value="250000" type="int"></constructor-arg>  <constructor-arg value="6000" type="double"></constructor-arg>   <!--其他，若字面值中包含特殊字符, 则可以使用 DCDATA 来进行赋值-->  <constructor-arg type="java.lang.String">  <value><![CDATA[Shanghai^]]></value>  </constructor-arg>   <!-- 属性值也可以使用value配置-->  <constructor-arg type="double">  <value>6000</value>  </constructor-arg>   </bean> </beans> |

### App

|  |
| --- |
| package com.shipeng;  import com.shipeng.spring.beans.Car; import org.springframework.context.ApplicationContext; import org.springframework.context.support.ClassPathXmlApplicationContext;  public class App {  public static void main(String[] args) {  ApplicationContext context = new ClassPathXmlApplicationContext("applicationContext.xml");   Car car = (Car) context.getBean("car");   System.***out***.println(car);  } } |

## 工厂方法创建bean对象

### Car

|  |
| --- |
| package com.shipeng.spring;  public class Car {  private String brand;  private double price;    public void setBrand(String brand) {  this.brand = brand;  }   public void setPrice(double price) {  this.price = price;  }   @Override  public String toString() {  return "Car{" +  "brand='" + brand + '\'' +  ", price=" + price +  '}';  }   public Car() {  System.***out***.println("Car无参构造函数");  }   public Car(String brand, double price) {  this.brand = brand;  this.price = price;  System.***out***.println("Car有参构造函数");  } } |

### applicationContext.xml

|  |
| --- |
| <!--  通过静态工厂方法来配置bean,注意不是配置静态工厂方法实例，而是配置bean实例   class属性：指向静态工厂方法的全类名  factory-method: 指向静态工厂方法的名字  constructor-arg：如果工厂方法需要传入参数，则使用constructor-arg来配置参数 --> <bean id="car1"  class="com.shipeng.spring.StaticCarFactory"  factory-method="getCar">  <constructor-arg value="audi"></constructor-arg> </bean>   <!--配置实例工厂方法来配置bean--> <!--  factory-bean属性：指向实例工厂方法的bean  factory-method: 指向静态工厂方法的名字  constructor-arg：如果工厂方法需要传入参数，则使用constructor-arg来配置参数 -->  <!--配置工厂的实例--> <bean id="carFactory"  class="com.shipeng.spring.InstantceCarFactory"> </bean>  <bean id="car2"  factory-bean="carFactory"  factory-method="getCar">  <constructor-arg value="ford"></constructor-arg> </bean> |

### StaticCarFactory(静态工厂)

|  |
| --- |
| package com.shipeng.spring;  import java.util.HashMap; import java.util.Map;  public class StaticCarFactory {  private static Map<String, Car> cars = new HashMap<String, Car>();   static {  cars.put("audi", new Car("audi", 300000));  cars.put("ford", new Car("ford", 400000));  }   // 静态工厂方法  public static Car getCar(String name) {  System.***out***.println("getCar静态工厂方法");  return cars.get(name);  } } |

### InstantceCarFactory(实例工厂)

|  |
| --- |
| package com.shipeng.spring;  import java.util.HashMap; import java.util.Map;  */\*  \* 实例工厂方法：实例工厂的方法，即现需要创建工厂本身，再调用工厂的实例方法来返回bena的实例  \* \*/* public class InstantceCarFactory {  private Map<String, Car> cars = null;   public InstantceCarFactory() {  cars = new HashMap<String, Car>();  cars.put("audi", new Car("audi", 300000));  cars.put("ford", new Car("ford", 400000));  }   // 实例工厂方法  public Car getCar(String brand) {  System.***out***.println("getCar实例工厂方法");  return cars.get(brand);  } } |

### App

|  |
| --- |
| package com.shipeng;  import com.shipeng.spring.Car; import org.springframework.context.ApplicationContext; import org.springframework.context.support.ClassPathXmlApplicationContext;  public class App {  public static void main(String[] args) {  ApplicationContext context = new ClassPathXmlApplicationContext("applicationContext.xml");   Car car1 = (Car) context.getBean("car1");  System.***out***.println(car1);   Car car2 = (Car) context.getBean("car2");  System.***out***.println(car2);  } } |

# bean的配置方式

## xml

|  |
| --- |
| 在 xml 文件中通过 bean 节点来配置 bean  id：Bean 的名称，在 IOC 容器中必须是唯一的。  若 id 没有指定，Spring 自动将权限定性类名作为 Bean 的名字  id 可以指定多个名字，名字之间可用逗号、分号、或空格分隔  class: bean的全类名，通过反射的方式在IOC容器中创建Bean,所  以要求bean要有无参构造器，先调用无参构造函数，再调用  <!--通过全类名配置bean--> <bean id="car" class="com.shipeng.spring.Car"></bean> |

## 注解

|  |
| --- |
|  |

# bean的自动装配

|  |
| --- |
| Spring IOC 容器可以自动装配 Bean. 需要做的仅仅是在 <bean> 的 autowire 属性里指定自动装配的模式  byType(根据类型自动装配): 若 IOC 容器中有多个与目标 Bean 类型一致的 Bean.  在这种情况下, Spring 将无法判定哪个 Bean 最合适该属性, 所以不能执行自动装配.  byName(根据名称自动装配): 必须将目标 Bean 的名称和属性名设置的完全相同.  constructor(通过构造器自动装配): 当 Bean 中存在多个构造器时, 此种自动装配方式将会很复杂. 不推荐使用 |

## Car

|  |
| --- |
| package com.shipeng.spring;  public class Car {  private String brand;  private double price;   public void setBrand(String brand) {  this.brand = brand;  }   public void setPrice(double price) {  this.price = price;  }   @Override  public String toString() {  return "Car{" +  "brand='" + brand + '\'' +  ", price=" + price +  '}';  }   public Car() {  System.***out***.println("Car无参构造函数");  } } |

## applicationContext.xml

|  |
| --- |
| <!-- 使用autowire属性指定自动装配的方式， byName 根据bean的名字和当前bean 的setter风格的属性进行自动装配，若有匹配的，则进行自动装配，若不匹配，则不装配，  byType 根据bean的类型和当前bean 的属性的类型进行自动装配，若IOC容器中有1个以上的类型匹配bean，则抛异常 --> <bean id="car" class="com.shipeng.spring.Car" autowire="byName">  <property name="price" value="20" />  <property name="brand" value="大众" /> </bean> |

## App

|  |
| --- |
| package com.shipeng;  import com.shipeng.spring.Car; import org.springframework.context.ApplicationContext; import org.springframework.context.support.ClassPathXmlApplicationContext;  public class App {  public static void main(String[] args) {  ApplicationContext context = new ClassPathXmlApplicationContext("applicationContext.xml");   Car car = (Car) context.getBean("car");  System.***out***.println(car);  } } |

# bean的关系

## 依赖

|  |
| --- |
| Spring 允许用户通过 depends-on 属性设定 Bean 前置依赖的Bean，前置依赖的 Bean 会在本 Bean 实例化之前创建好   如果前置依赖于多个 Bean，则可以通过逗号，空格或的方式配置 Bean 的名称 |

## 继承

|  |
| --- |
| Spring 允许继承 bean 的配置, 被继承的 bean 称为父 bean. 继承这个父 Bean 的 Bean 称为子 Bean  子 Bean 从父 Bean 中继承配置, 包括 Bean 的属性配置   子 Bean 也可以覆盖从父 Bean 继承过来的配置   父 Bean 可以作为配置模板, 也可以作为 Bean 实例. 若只想把父 Bean 作为模板, 可以设置 <bean> 的abstract 属性为 true, 这样 Spring 将不会实例化这个 Bean   并不是 <bean> 元素里的所有属性都会被继承. 比如: autowire, abstract 等.   也可以忽略父 Bean 的 class 属性, 让子 Bean 指定自己的类, 而共享相同的属性配置. 但此时 abstract 必须设为 true |

# bean的作用域

|  |
| --- |
| 在 Spring 中, 可以在 <bean> 元素的 scope 属性里设置 Bean 的作用域.  默认情况下, Spring 只为每个在 IOC 容器里声明的 Bean 创建唯一一个实例, 整个 IOC 容器范围内都能共享该实例：所有后续的 getBean() 调用和 Bean 引用都将返回这个唯一的 Bean 实例.该作用域被称为 singleton, 它是所有 Bean 的默认作用域. |

## xml

### Car

|  |
| --- |
| package com.shipeng.spring;  public class Car {  private String brand;  private double price;   public void setBrand(String brand) {  this.brand = brand;  }   public void setPrice(double price) {  this.price = price;  }   @Override  public String toString() {  return "Car{" +  "brand='" + brand + '\'' +  ", price=" + price +  '}';  }   public Car() {  System.***out***.println("Car无参构造函数");  } } |

### applicationContext.xml

|  |
| --- |
| <!-- 使用bean的scope属性来配置bean的作用域 singleton：默认值，容器初始化时创建bean实例，在整个容器的生命周期内只创建一个bean，单例的。 prototype：原型的，容器初始化时不创建bean的实例，而在每次请求时都创建一个新的bean实例，并返回。 --> <bean id="car"  class="com.shipeng.spring.Car"  scope="singleton">  <property name="brand" value="Audi"></property>  <property name="price" value="300000"></property> </bean> |

### App

|  |
| --- |
| package com.shipeng;  import com.shipeng.spring.Car; import org.springframework.context.ApplicationContext; import org.springframework.context.support.ClassPathXmlApplicationContext;  public class App {  public static void main(String[] args) {  ApplicationContext context = new ClassPathXmlApplicationContext("applicationContext.xml");   Car car1 = (Car) context.getBean("car");  Car car2 = (Car) context.getBean("car");  System.***out***.println(car1 == car2);  } } |

## 注解

|  |
| --- |
|  |

# bean的范围

|  |
| --- |
|  |

# bean的延迟加载

## xml

|  |
| --- |
|  |

## 注解

|  |
| --- |
|  |

# bean的歧义性

|  |
| --- |
|  |

# bean的生命周期

|  |
| --- |
|  |

# p命名空间注入属性

## Car

|  |
| --- |
| package com.shipeng.spring;  public class Car {  private String brand;  private double price;   public void setBrand(String brand) {  this.brand = brand;  }   public void setPrice(double price) {  this.price = price;  }   @Override  public String toString() {  return "Car{" +  "brand='" + brand + '\'' +  ", price=" + price +  '}';  }   public Car() {  System.***out***.println("Car无参构造函数");  } } |

## Person

|  |
| --- |
| package com.shipeng.spring;  public class Person {  private String name;  private Car car;   public void setName(String name) {  this.name = name;  }   public void setCar(Car car) {  this.car = car;  }   public Person() {  super();  System.***out***.println("Person无参构造函数");  }   @Override  public String toString() {  return "Person{" +  "name='" + name + '\'' +  ", car=" + car +  '}';  } } |

## applicationContext.xml

|  |
| --- |
| 1.添加 xmlns:p="http://www.springframework.org/schema/p"  <bean id="car"  class="com.shipeng.spring.Car"  p:brand="大众"  p:price="200"> </bean>  <!--注入属性或对象--> <bean id="person"  class="com.shipeng.spring.Person"  p:name="张三"  p:car-ref="car"> </bean> |

## App

|  |
| --- |
| package com.shipeng;  import com.shipeng.spring.Person; import org.springframework.context.ApplicationContext; import org.springframework.context.support.ClassPathXmlApplicationContext;  public class App {  public static void main(String[] args) {  ApplicationContext context = new ClassPathXmlApplicationContext("applicationContext.xml");   Person person = (Person) context.getBean("person");  System.***out***.println(person);  } } |

# Util命名空间注入属性

|  |
| --- |
| 使用基本的集合标签定义集合时, 不能将集合作为独立的 Bean 定义, 导致其他 Bean 无法引用该集合, 所以无法在不同 Bean 之间共享集合.  可以使用 util schema 里的集合标签定义独立的集合 Bean. 需要注意的是, 必须在 <beans> 根元素里添加 util schema 定义  xmlns:util="http://www.springframework.org/schema/util"  xsi:schemaLocation="http://www.springframework.org/schema/util http://www.springframework.org/schema/util/spring-util.xsd |

## Car

|  |
| --- |
| package com.shipeng.spring;  public class Car {  private String brand;  private double price;   public void setBrand(String brand) {  this.brand = brand;  }   public void setPrice(double price) {  this.price = price;  }   @Override  public String toString() {  return "Car{" +  "brand='" + brand + '\'' +  ", price=" + price +  '}';  }   public Car() {  System.***out***.println("Car无参构造函数");  } } |

## Person

|  |
| --- |
| package com.shipeng.spring;  import java.util.List;  public class Person {  private String name;  private List<Car> cars;   public void setName(String name) {  this.name = name;  }   public void setCars(List<Car> cars) {  this.cars = cars;  }   public Person() {  super();  System.***out***.println("Person无参构造函数");  }   @Override  public String toString() {  return "Person{" +  "name='" + name + '\'' +  ", cars=" + cars +  '}';  } } |

## applicationContext.xml

|  |
| --- |
| <bean id="car1"  class="com.shipeng.spring.Car"  p:brand="大众"  p:price="200"> </bean>  <bean id="car2"  class="com.shipeng.spring.Car"  p:brand="奥迪"  p:price="300"> </bean>  <util:list id="cars">  <ref bean="car1"/>  <ref bean="car2"/> </util:list>  <util:map></util:map> <util:set></util:set>  <!--注入属性或对象--> <bean id="person"  class="com.shipeng.spring.Person"  p:name="张三"  p:cars-ref="cars">  <!--  <property name="name" value="李四"></property>  <property name="cars" ref="cars"></property>--> </bean> |

## App

|  |
| --- |
| package com.shipeng;  import com.shipeng.spring.Person; import org.springframework.context.ApplicationContext; import org.springframework.context.support.ClassPathXmlApplicationContext;   public class App {  public static void main(String[] args) {  ApplicationContext context = new ClassPathXmlApplicationContext("applicationContext.xml");   Person person = (Person) context.getBean("person");  System.***out***.println(person);  } } |

# 配置集合属性：list、set、map、数组

|  |
| --- |
| 1.如果使用构造函数注入，可以把下面的<property />节点换成<constructor-arg />节点  如：  <property name="cars">  <list>  <ref bean="car1"/>  <ref bean="car2"/>  </list> </property> 换成  <constructor-arg name="cars">  <list>  <ref bean="car1"/>  <ref bean="car2"/>  </list> </constructor-arg>  2.可以使用util命名空间，配置单例的集合bean，以供多个bean进行引用，需要导入util命名空间  <util:list id="cars">  <ref bean="car1"/>  <ref bean="car2"/> </util:list>  <util:map></util:map> <util:set></util:set> |

## Car不变

|  |
| --- |
| package com.shipeng.spring;  public class Car {  private String brand;  private double price;   public void setBrand(String brand) {  this.brand = brand;  }   public void setPrice(double price) {  this.price = price;  }   @Override  public String toString() {  return "Car{" +  "brand='" + brand + '\'' +  ", price=" + price +  '}';  }   public Car() {  System.***out***.println("Car无参构造函数");  } } |

## App不变

|  |
| --- |
| package com.shipeng;  import com.shipeng.spring.Person; import org.springframework.context.ApplicationContext; import org.springframework.context.support.ClassPathXmlApplicationContext;  public class App {  public static void main(String[] args) {  ApplicationContext context = new ClassPathXmlApplicationContext("applicationContext.xml");   Person person = (Person) context.getBean("person");  System.***out***.println(person);  } } |

## list字符串

|  |
| --- |
| Person  package com.shipeng.spring;  import java.util.List;  public class Person {  private String name;  private List<String> cars;   public void setName(String name) {  this.name = name;  }   public void setCars(List<String> cars) {  this.cars = cars;  }public Person() {  super();  System.***out***.println("Person无参构造函数");  }   @Override  public String toString() {  return "Person{" +  "name='" + name + '\'' +  ", cars=" + cars +  '}';  } }  // ==========================================  applicationContext.xml  <bean id="person"  class="com.shipeng.spring.Person">  <property name="name" value="张三"></property>  <property name="cars">  <list>  <value>大众</value>  <value>奥迪</value>  </list>  </property> </bean> |

## list对象

|  |
| --- |
| Person  package com.shipeng.spring;  import java.util.List;  public class Person {  private String name;  private List<Car> cars;   public void setName(String name) {  this.name = name;  }   public void setCars(List<Car> cars) {  this.cars = cars;  }    public Person() {  super();  System.***out***.println("Person无参构造函数");  }   @Override  public String toString() {  return "Person{" +  "name='" + name + '\'' +  ", cars=" + cars +  '}';  } }  // ==================================  applicationContext.xml  <bean id="car1"  class="com.shipeng.spring.Car"  p:brand="大众"  p:price="200"> </bean>  <bean id="car2"  class="com.shipeng.spring.Car"  p:brand="奥迪"  p:price="300"> </bean>  <!--使用list节点为List类型的属性赋值--> <bean id="person"  class="com.shipeng.spring.Person">  <property name="name" value="张三"></property>  <property name="cars">  <list>  <ref bean="car1"/>  <ref bean="car2"/>  </list>  </property> </bean>  <!--set-->  <bean id="person"  class="com.shipeng.spring.Person">  <property name="name" value="张三"></property>  <property name="cars">  <set>  <ref bean="car1"/>  <ref bean="car2"/>  </set>  </property> </bean>  list |

## set集合

|  |
| --- |
| Person  package com.shipeng.spring;  import java.util.Set;  public class Person {  private String name;  private Set<Car> cars;   public void setName(String name) {  this.name = name;  }   public void setCars(Set<Car> cars) {  this.cars = cars;  }   public Person() {  super();  System.***out***.println("Person无参构造函数");  }   @Override  public String toString() {  return "Person{" +  "name='" + name + '\'' +  ", cars=" + cars +  '}';  } }  // ==============================  applicationContext.xml  <bean id="car1"  class="com.shipeng.spring.Car"  p:brand="大众"  p:price="200"> </bean>  <bean id="car2"  class="com.shipeng.spring.Car"  p:brand="奥迪"  p:price="300"> </bean>  <bean id="person"  class="com.shipeng.spring.Person">  <property name="name" value="张三"></property>  <property name="cars">  <set>  <ref bean="car1"/>  <ref bean="car2"/>  </set>  </property> </bean> |

## 数组

|  |
| --- |
| Person  package com.shipeng.spring;  import java.util.Arrays;  public class Person {  private String name;  private Car[] cars;   public void setName(String name) {  this.name = name;  }   public void setCars(Car[] cars) {  this.cars = cars;  }   public Person() {  super();  System.***out***.println("Person无参构造函数");  }   @Override  public String toString() {  return "Person{" +  "name='" + name + '\'' +  ", cars=" + Arrays.toString(cars) +  '}';  } }  // =================================  applicationContext.xml  <bean id="car1"  class="com.shipeng.spring.Car"  p:brand="大众"  p:price="200"> </bean>  <bean id="car2"  class="com.shipeng.spring.Car"  p:brand="奥迪"  p:price="300"> </bean>  <bean id="person"  class="com.shipeng.spring.Person">  <property name="name" value="张三"></property>  <property name="cars">  <array>  <ref bean="car1"/>  <ref bean="car2"/>  </array>  </property> </bean> |

## map集合

|  |
| --- |
| package com.shipeng.spring;  import java.util.Map;  public class Person {  private String name;  private Map<String, Car> cars;   public void setName(String name) {  this.name = name;  }   public void setCars(Map<String, Car> cars) {  this.cars = cars;  }   public Person() {  super();  System.***out***.println("Person无参构造函数");  }   @Override  public String toString() {  return "Person{" +  "name='" + name + '\'' +  ", cars=" + cars +  '}';  } }  // ======================= applicationContext.xml  <bean id="car1"  class="com.shipeng.spring.Car"  p:brand="大众"  p:price="200">  </bean>   <bean id="car2"  class="com.shipeng.spring.Car"  p:brand="奥迪"  p:price="300">  </bean>  <!--使用map节点及map的entry子节点配置Map类型的成员变量-->  <bean id="person"  class="com.shipeng.spring.Person">  <property name="name" value="张三"></property>  <property name="cars">  <map>  <!--<entry key="" value="" />-->  <entry key="c1" value-ref="car1" />  <entry key="c2" value-ref="car2" />  </map>  </property>  </bean> |

## Properties属性值

|  |
| --- |
| <!--配置Properties属性值--> <bean id="dataSource" class="com.atguigu.spring.beans.collections.DateSource">  <property name="properties">  <props>  <prop key="user">user</prop>  <prop key="password">password</prop>  <prop key="jdbcUrl">jdbcUrl</prop>  <prop key="dirverClass">com.mysql.jdbc.Driver</prop>  </props>  </property> </bean> |

# 自动扫描包

|  |
| --- |
| <!-- 配置自动扫描的包: 需要加入 aop 对应的 jar 包 --> <context:component-scan base-package="com.atguigu.spring.annotation.generic"></context:component-scan> |

# 从IOC容器中获取bean的方式

|  |
| --- |
| //2. 从IOC容器中获取bean的实例, 利用id定位到IOC容器中的bean HelloWorld helloWorld = (HelloWorld) ctx.getBean("helloWorld3");  // 或 // HelloWorld.class 根据类型获取 bean 的实例: 要求在 IOC 容器中只有一个与之类型匹配的 bean, // 若有多个则会抛出异常. // 一般情况下, 该方法可用, 因为一般情况下, 在一个 IOC 容器中一个类型对应的 bean 也只有一个. HelloWorld helloWorld = ctx.getBean(HelloWorld.class); |

# 使用外部属性文件

## db.properties

|  |
| --- |
| user=root password=root driverclass=com.mysql.jdbc.Driver jdbcurl=jdbc:mysql:///test |

## applicationContext.xml

|  |
| --- |
| <!--导入属性文件--> <context:property-placeholder location="classpath:db.properties"/>  <bean id="dataSource"  class="com.mchange.v2.c3p0.ComboPooledDataSource">  <property name="user" value="${user}"></property>  <property name="password" value="${password}"></property>  <property name="driverClass" value="${driverclass}"></property>  <property name="jdbcUrl" value="${jdbcUrl}"></property> </bean> |

## App

|  |
| --- |
| import org.springframework.context.ApplicationContext; import org.springframework.context.support.ClassPathXmlApplicationContext;  import javax.sql.DataSource; import java.sql.SQLException;  public class main {  public static void main(String[] args) throws SQLException {  ApplicationContext ctx = new ClassPathXmlApplicationContext("applicationContext.xml");  DataSource dataSource = (DataSource) ctx.getBean("dataSource");  System.***out***.println(dataSource.getConnection());  } } |

# 测试类中获取bean的方式

|  |
| --- |
|  |

# 字面值

|  |
| --- |
| 1.字面值：可用字符串表示的值，可以通过 <value> 元素标签或 value 属性进行注入。  2.基本数据类型及其封装类、String 等类型都可以采取字面值注入的方式  3.若字面值中包含特殊字符，可以使用 <![CDATA[]]> 把字面值包裹起来。  <bean id="car2" class="com.atguigu.spring.helloworld.Car">  <constructor-arg value="ChangAnMazda"></constructor-arg>  <!-- 若字面值中包含特殊字符, 则可以使用 DCDATA 来进行赋值. (了解) -->  <constructor-arg>  <value><![CDATA[<广州^>]]></value>  </constructor-arg>  <constructor-arg value="180" type="int"></constructor-arg> </bean> |

# null 值和级联属性

|  |
| --- |
| 使用 <null/> 元素标签为 Bean 的字符串或其它对象类型的属性注入 null 值  <bean id="dao" class="com.atguigu.spring.ref.Dao">  <!--若某一个 bean 的属性值不是 null, 使用时需要为其设置为 null(了解) -->   <property name="dataSource"><null/></property> </bean>  ==========================================  和 Struts、Hiberante 等框架一样，Spring 支持级联属性的配置。  <bean id="action" class="com.atguigu.spring.ref.Action">  <property name="service" ref="service"></property>  <!-- 设置级联属性(了解) ：查找id为service的ben，获取引用属性dao，再获取dataSource属性-->  <property name="service.dao.dataSource" value="DBCP2"></property> </bean>  ==================================================  <bean id="person2" class="com.atguigu.spring.beans.Person">  <constructor-arg value="Jerry"></constructor-arg>  <constructor-arg value="25"></constructor-arg>  <constructor-arg ref="car"></constructor-arg>  <!--测试赋值null  <constructor-arg><null/></constructor-arg>  -->  <!--为级联属性赋值。注意：属性需要先初始化后才可以为级联属性赋值，否则会有异常，和struts2不同-->  <!--car必须有对应的settr方法-->  <property name="car.maxSpeed" value="250"></property> </bean> |

# SPEL

|  |
| --- |
|  |

# AOP

|  |
| --- |
|  |

# 事务管理

|  |
| --- |
|  |