|  |
| --- |
| Spring |
| Replaced-Method  public class ReplacedM implements MethodReplacer {  @Override  public Object reimplement(Object arg0, Method arg1, Object[] arg2) throws Throwable {    System.*out*.println(arg2);  return "hallo";  }  }  <bean id=*"T1000"* class=*"ru.javabegin.training.spring.impls.robot.ModelT1000"*  autowire=*"byType"* scope=*"prototype"*>  <replaced-method name=*"dance"* replacer=*"replacementTarget"*/>  </bean>  <bean id=*"replacementTarget"* class=*"ru.javabegin.training.spring.main.ReplacedM"*>  </bean> |
| Lookup-Method  public interface IRobotConveyer { //kann auch class oder abstract class sein    public Robot getRobot();  }  <bean id=*"RobotConveyer"* class=*"ru.javabegin.training.spring.main.IRobotConveyer"*>  <lookup-method name=*"getRobot"* bean=*"T1000"* />  </bean> |
| Autowired  <bean id=*"T1000"* class=*"ru.javabegin.training.spring.impls.robot.ModelT1000"*  autowire=*"byType"* scope=*"prototype"*>  //autowire versucht für alle in class *ModelT1000* befindlichen Properties entsprechnde bean im context zu finden.  <bean class=*"ru.javabegin.training.spring.impls.robot.ModelT1000"*  autowire-candidate=*"true"*>  <constructor-arg value=*"golded"* type=*"String"*/>  <constructor-arg value=*"2007"* type=*"int"*/>  <constructor-arg value=*"false"* type=*"boolean"*/>  </bean>  <bean class=*"ru.javabegin.training.spring.impls.robot.ModelT1000"*  autowire-candidate=*"true"*>  <constructor-arg value=*"gold"* />  <constructor-arg value=*"3219"* type=*"int"*/>  <constructor-arg value=*"true"* type=*"boolean"*/>  </bean>    <bean id=*"t1000Pool"* class=*"ru.javabegin.training.spring.impls.robot.ModelT1000"* autowire=*"byType"*/>  public void setRobotCollection(Collection<Robot> robotCollection) {  this.robotCollection = robotCollection;  }  So funktioniert es auch. Spring sieht Propertie Collection und sucht für diese Collection alle passende Beans. |
| Annotation  Required  <context:annotation-config></context:annotation-config>  @Required  public void setHand(Hand hand) {  this.hand = hand;  }  Autowired  @Autowired // выдает ошибкy если не нашел  private Hand hand;  @Autowired(required=false) // не выдает ошибки если не нашел  private Head head;  @Autowired  @Qualifier("SonyLeg")  private Leg leg;  @Autowired(required=false)  @Qualifier("SonyHeadGold")  private Head head;  <bean id=*"SonyHead"* class=*"ru.javabegin.training.spring.impls.sony.SonyHead"*  autowire-candidate=*"false"* >  <qualifier value=*"SonyHeadGold"*></qualifier>  </bean>  @Autowired  public void preper(Leg leg, Head head) {  this.head = head;  this.leg = leg;  System.*out*.println("Klapts");  }  @Autowired  public ModelT1000(Hand hand, Leg leg, Head head) {  super();  this.hand = hand;  this.leg = leg;  this.head = head;  }  public class MovieRecommender {  @Autowired  private MovieCatalog[] movieCatalogs;  *// ...*  }  public class MovieRecommender {  private Set<MovieCatalog> movieCatalogs;  @Autowired  public void setMovieCatalogs(Set<MovieCatalog> movieCatalogs) {  this.movieCatalogs = movieCatalogs;  }  *// ...*  }  Создание собственных аннотаций что то вроде группы и подгруппы  @Target({ElementType.FIELD, ElementType.PARAMETER})  @Retention(RetentionPolicy.RUNTIME)  @Qualifier  public @interface Genre {  String value();  }   * ElementType.ANNOTATION\_TYPE can be applied to an annotation type. * ElementType.CONSTRUCTOR can be applied to a constructor. * ElementType.FIELD can be applied to a field or property. * ElementType.LOCAL\_VARIABLE can be applied to a local variable. * ElementType.METHOD can be applied to a method-level annotation. * ElementType.PACKAGE can be applied to a package declaration. * ElementType.PARAMETER can be applied to the parameters of a method. * ElementType.TYPE can be applied to any element of a class.   Then you can provide the custom qualifier on autowired fields and parameters:  public class MovieRecommender {  @Autowired  @Genre("Action")  private MovieCatalog actionCatalog;  private MovieCatalog comedyCatalog;  @Autowired  public void setComedyCatalog(@Genre("Comedy") MovieCatalog comedyCatalog) {  this.comedyCatalog = comedyCatalog;  }  *// ...*  }  Next, provide the information for the candidate bean definitions. You can add <qualifier/> tags as sub-elements of the <bean/> tag and then specify the type and value to match your custom qualifier annotations. The type is matched against the fully-qualified class name of the annotation. Or, as a convenience if no risk of conflicting names exists, you can use the short class name. Both approaches are demonstrated in the following example.  <?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-3.0.xsd  http://www.springframework.org/schema/context  http://www.springframework.org/schema/context/spring-context-3.0.xsd">  <context:annotation-config/>  <bean class="example.SimpleMovieCatalog">  <qualifier type="Genre" value="Action"/>  <*!-- inject any dependencies required by this bean --*>  </bean>  <bean class="example.SimpleMovieCatalog">  <qualifier type="example.Genre" value="Comedy"/>  <*!-- inject any dependencies required by this bean --*>  </bean>  <bean id="movieRecommender" class="example.MovieRecommender"/>  </beans>  Oder so  @Target({ElementType.FIELD, ElementType.PARAMETER})  @Retention(RetentionPolicy.RUNTIME)  @Qualifier  public @interface MovieQualifier {  String genre();  Format format();  }  In this case Format is an enum:  public enum Format {  VHS, DVD, BLURAY  }  The fields to be autowired are annotated with the custom qualifier and include values for both attributes: genre and format.  public class MovieRecommender {  @Autowired  @MovieQualifier(format=Format.VHS, genre="Action")  private MovieCatalog actionVhsCatalog;  @Autowired  @MovieQualifier(format=Format.VHS, genre="Comedy")  private MovieCatalog comedyVhsCatalog;  @Autowired  @MovieQualifier(format=Format.DVD, genre="Action")  private MovieCatalog actionDvdCatalog;  @Autowired  @MovieQualifier(format=Format.BLURAY, genre="Comedy")  private MovieCatalog comedyBluRayCatalog;  *// ...*  }  Finally, the bean definitions should contain matching qualifier values. This example also demonstrates that bean *meta* attributes may be used instead of the <qualifier/> sub-elements. If available, the <qualifier/> and its attributes take precedence, but the autowiring mechanism falls back on the values provided within the <meta/> tags if no such qualifier is present, as in the last two bean definitions in the following example.  <?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-3.0.xsd  http://www.springframework.org/schema/context  http://www.springframework.org/schema/context/spring-context-3.0.xsd">  <context:annotation-config/>  <bean class="example.SimpleMovieCatalog">  <qualifier type="MovieQualifier">  <attribute key="format" value="VHS"/>  <attribute key="genre" value="Action"/>  </qualifier>  <*!-- inject any dependencies required by this bean --*>  </bean>  <bean class="example.SimpleMovieCatalog">  <qualifier type="MovieQualifier">  <attribute key="format" value="VHS"/>  <attribute key="genre" value="Comedy"/>  </qualifier>  <*!-- inject any dependencies required by this bean --*>  </bean>  <bean class="example.SimpleMovieCatalog">  <meta key="format" value="DVD"/>  <meta key="genre" value="Action"/>  <*!-- inject any dependencies required by this bean --*>  </bean>  <bean class="example.SimpleMovieCatalog">  <meta key="format" value="BLURAY"/>  <meta key="genre" value="Comedy"/>  <*!-- inject any dependencies required by this bean --*>  </bean>  </beans> |
| **Classpath scanning** <context:component-scan base-package="org.example"/>  @Service  public class SimpleMovieLister {  private MovieFinder movieFinder;  @Autowired  public SimpleMovieLister(MovieFinder movieFinder) {  this.movieFinder = movieFinder;  }  }  @Repository  public class JpaMovieFinder implements MovieFinder {  *// implementation elided for clarity*  }  @Component  public class FactoryMethodComponent {  private static int i;  @Bean @Qualifier("public")  public TestBean publicInstance() {  return new TestBean("publicInstance");  }  *// use of a custom qualifier and autowiring of method parameters*  @Bean  protected TestBean protectedInstance(@Qualifier("public") TestBean spouse,  @Value("#{privateInstance.age}") String country) {  TestBean tb = new TestBean("protectedInstance", 1);  tb.setSpouse(tb);  tb.setCountry(country);  return tb;  }  @Bean @Scope(BeanDefinition.SCOPE\_SINGLETON)  private TestBean privateInstance() {  return new TestBean("privateInstance", i++);  }  @Bean @Scope(value = WebApplicationContext.SCOPE\_SESSION,  proxyMode = ScopedProxyMode.TARGET\_CLASS)  public TestBean requestScopedInstance() {  return new TestBean("requestScopedInstance", 3);  }  }  @Component  public class FactoryMethodComponent {  private static int i;  @Bean @Qualifier("public")  public TestBean publicInstance() {  return new TestBean("publicInstance");  }  *// use of a custom qualifier and autowiring of method parameters*  @Bean  protected TestBean protectedInstance(@Qualifier("public") TestBean spouse,  @Value("#{privateInstance.age}") String country) {  TestBean tb = new TestBean("protectedInstance", 1);  tb.setSpouse(tb);  tb.setCountry(country);  return tb;  }  @Bean @Scope(BeanDefinition.SCOPE\_SINGLETON)  private TestBean privateInstance() {  return new TestBean("privateInstance", i++);  }  @Bean @Scope(value = WebApplicationContext.SCOPE\_SESSION,  proxyMode = ScopedProxyMode.TARGET\_CLASS)  public TestBean requestScopedInstance() {  return new TestBean("requestScopedInstance", 3);  }  }  1  @Component  public class FactoryMethodComponent {  private static int i;  @Bean @Qualifier("public")  public TestBean publicInstance() {  return new TestBean("publicInstance");  }  *// use of a custom qualifier and autowiring of method parameters*  @Bean  protected TestBean protectedInstance(@Qualifier("public") TestBean spouse,  @Value("#{privateInstance.age}") String country) {  TestBean tb = new TestBean("protectedInstance", 1);  tb.setSpouse(tb);  tb.setCountry(country);  return tb;  }  @Bean @Scope(BeanDefinition.SCOPE\_SINGLETON)  private TestBean privateInstance() {  return new TestBean("privateInstance", i++);  }  @Bean @Scope(value = WebApplicationContext.SCOPE\_SESSION,  proxyMode = ScopedProxyMode.TARGET\_CLASS)  public TestBean requestScopedInstance() {  return new TestBean("requestScopedInstance", 3);  }  }  @Service  public class ModelT1000 implements Robot {  private String color;  @Autowired  private Hand hand;  @Autowired  private Head head;  @Autowired  private Leg leg;  }  @Component oder @Service @Controll @Repository  public class SonyLeg implements Leg {    public void go(){  System.*out*.println("Go to Sony!");  }  }  public static void main(String[] args) {  ApplicationContext context = new ClassPathXmlApplicationContext("context.xml");    ModelT1000 t1000 = (ModelT1000) context.getBean("modelT1000");  } |
| @Bean  @Qualifier("test")  public ModelT1000 robot() {  return new ModelT1000("blau", 123, true);  }  @Bean  //@Qualifier("test") is autowired  private ModelT1000 tesRobot(@Qualifier("test") ModelT1000 mod) {  System.*out*.println("ich bins");  return mod;  }  public static void main(String[] args) {  ApplicationContext context = new ClassPathXmlApplicationContext("context.xml");    ModelT1000 robot20 =(ModelT1000) context.getBean("robot");  robot20.action();    } |
| Using JSR 330 Standard Annotations   |  |  |  | | --- | --- | --- | | @Resource | javax.annotation | Java | | @Inject | javax.inject | Java | | @Qualifier | javax.inject | Java | | @Autowired | org.springframework.bean.factory | Spring |  |  | | --- | | @Resource  **private** Party party; | | @Autowired  **private** Party party; |  |  | | --- | | @Inject  **private** Party party; | |  |  **Feine unterschiede** |
| private String color;  @Inject  private Hand hand;  @Inject  private Head head; @Inject @Named  public class SonyHand implements Hand{    public void catchSomething(){  System.*out*.println("Catched from Sony!!");  }  } |
|  |
|  |
|  |