百知教育 — Spring系列课程 — 工厂

第一章引言

1. EJB存在的问题

EJB (Enterprise Java Bean)

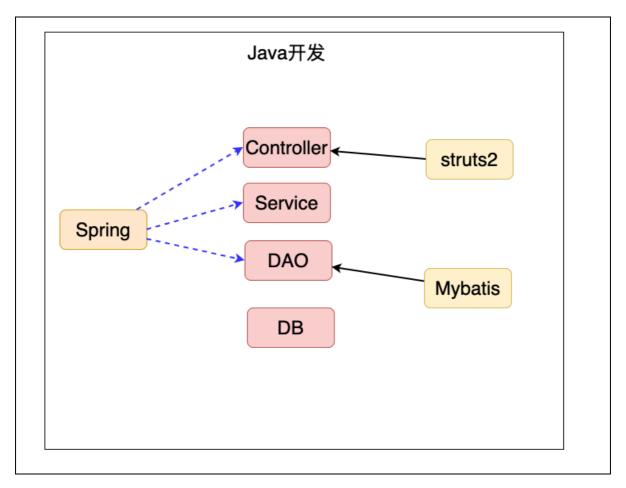
- 1. 运行环境苛刻
- 2. 代码移植性差

总结: EJB重量级的框架

2. 什么是Spring

- 1 Spring是一个轻量级的JavaEE解决方案,整合众多优秀的设计模式
 - 轻量级
 - 1 1. 对于运行环境是没有额外要求的
 - 2 开源 tomcat resion jetty3 收费 weblogic websphere

 - 4 2. 代码移植性高
 - 不需要实现额外接口
 - JavaEE的解决方案



• 整合设计模式

```
1 1. 工厂
2 2. 代理
3 3. 模板
4 4. 策略
```

3. 设计模式

```
    1.广义概念
    面向对象设计中,解决特定问题的经典代码
    2.狭义概念
    GOF4人帮定义的23种设计模式:工厂、适配器、装饰器、门面、代理、模板...
```

4. 工厂设计模式

4.1 什么是工厂设计模式

```
1. 概念: 通过工厂类,创建对象
User user = new User();
UserDAO userDAO = new UserDAOImpl();

2. 好处: 解耦合
精合: 指定是代码间的强关联关系,一方的改变会影响到另一方问题: 不利于代码维护
简单: 把接口的实现类,硬编码在程序中
UserService userService = new UserServiceImpl();
```

4.2 简单工厂的设计

```
package com.baizhiedu.basic;
 2
 3
     import java.io.IOException;
 4
     import java.io.InputStream;
 5
     import java.util.Properties;
 6
 7
     public class BeanFactory {
 8
         private static Properties env = new Properties();
9
10
         static{
             try {
11
12
                 //第一步 获得IO输入流
13
                 InputStream inputStream =
     BeanFactory.class.getResourceAsStream("/applicationContext.properties"
     );
                 //第二步 文件内容 封装 Properties集合中 key = userService
14
     value = com.baizhixx.UserServiceImpl
                 env.load(inputStream);
15
16
                 inputStream.close();
17
             } catch (IOException e) {
18
19
                 e.printStackTrace();
20
21
22
         }
23
24
25
         /*
26
             对象的创建方式:
27
                1. 直接调用构造方法 创建对象 UserService userService = new
     UserServiceImpl();
28
                2. 通过反射的形式 创建对象 解耦合
29
                    Class clazz =
     Class.forName("com.baizhiedu.basic.UserServiceImpl");
30
                    UserService userService =
     (UserService)clazz.newInstance();
31
          */
32
         public static UserService getUserService() {
33
34
             UserService userService = null;
35
             try {
36
     //com.baizhiedu.basic.UserServiceImpl
37
                 Class clazz =
     Class.forName(env.getProperty("userService"));
38
                 userService = (UserService) clazz.newInstance();
39
             } catch (ClassNotFoundException e) {
40
                 e.printStackTrace();
41
             } catch (InstantiationException e) {
                 e.printStackTrace();
42
             } catch (IllegalAccessException e) {
43
44
                 e.printStackTrace();
45
46
47
             return userService;
```

```
48
         }
49
50
51
52
         public static UserDAO getUserDAO(){
53
54
              UserDAO userDAO = null;
55
              try {
56
                  Class clazz = Class.forName(env.getProperty("userDAO"));
                  userDAO = (UserDAO) clazz.newInstance();
57
58
              } catch (ClassNotFoundException e) {
59
                  e.printStackTrace();
              } catch (InstantiationException e) {
60
                  e.printStackTrace();
              } catch (IllegalAccessException e) {
62
                  e.printStackTrace();
63
64
65
66
             return userDAO;
67
68
         }
69
```

4.3 通用工厂的设计

• 问题

简单工厂会存在大量的代码冗余

```
public static UserService getUserService() {
    UserService userService = null;
    try {
                                       //com.baizhiedu.basic.UserServiceImpl
        Class clazz = Class.forName(env.getProperty("userService"));
        userService = (UserService) clazz.newInstance();
    } catch (ClassNotFoundException e) {
        e.printStackTrace();
    } catch (InstantiationException e) {
        e.printStackTrace();
    } catch (IllegalAccessException e) {
        e.printStackTrace();
    return userService;
}
public static UserDAO getUserDAO(){
    UserDA0 userDA0 = null;
    try {
        Class clazz = Class.forName(env.getProperty("userDAO"));
userDAO = (UserDAO) clazz.newInstance();
    } catch (ClassNotFoundException e) {
        e.printStackTrace();
    } catch (InstantiationException e) {
        e.printStackTrace();
    } catch (IllegalAccessException e) {
        e.printStackTrace();
    return userDAO;
```

• 通用工厂的代码

```
创建一切想要的对象
 2
     public class BeanFactory{
 3
 4
         public static Object getBean(String key){
 5
              Object ret = null;
 6
              try {
 7
                  Class clazz = Class.forName(env.getProperty(key));
 8
                  ret = clazz.newInstance();
 9
              } catch (Exception e) {
                 e.printStackTrace();
10
11
12
              return ret;
13
          }
14
15
   }
```

4.4 通用工厂的使用方式

```
1 1. 定义类型 (类)
2 2. 通过配置文件的配置告知工厂(applicationContext.properties)
3 key = value
4 3. 通过工厂获得类的对象
5 Object ret = BeanFactory.getBean("key")
```

5.总结

```
1 Spring本质: 工厂 ApplicationContext (applicationContext.xml)
```

第二章、第一个Spring程序

1. 软件版本

```
1 1. JDK1.8+
2 2. Maven3.5+
3 3. IDEA2018+
4 4. SpringFramework 5.1.4
5 官方网站 www.spring.io
```

2. 环境搭建

• Spring的jar包

```
#设置pom 依赖

2  <!-- https://mvnrepository.com/artifact/org.springframework/spring-
    context -->
3    <dependency>
        <groupId>org.springframework</groupId>
        <artifactId>spring-context</artifactId>
        <version>5.1.4.RELEASE</version>
7    </dependency>
```

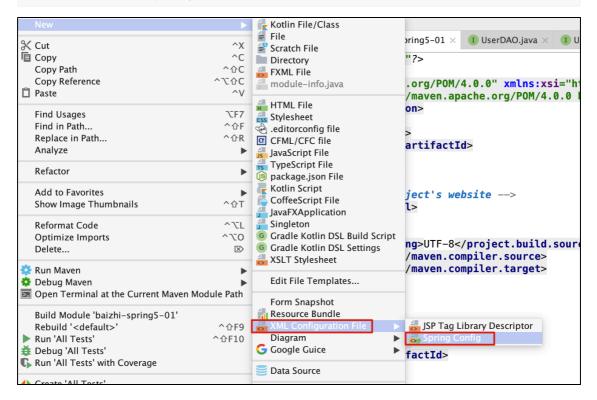
• Spring的配置文件

1 1. 配置文件的放置位置:任意位置 没有硬性要求

2 2. 配置文件的命名 : 没有硬性要求 建议: applicationContext.xml

3

4 思考:日后应用Spring框架时,需要进行配置文件路径的设置。



3. Spring的核心API

ApplicationContext

1 作用: Spring提供的ApplicationContext这个工厂,用于对象的创建

2 好处:解耦合

○ ApplicationContext接口类型

1 接口:屏蔽实现的差异

2 非web环境: ClassPathXmlApplicationContext (main junit)

3 web环境 : XmlWebApplicationContext

```
ApplicationContext (org.springframework.context)

    ConfigurableApplicationContext (org.springframework.context)

  AbstractApplicationContext (org.springframework.context.support)
     AbstractRefreshableApplicationContext (org.springframework.context.support)
        🔻 📵 🦫 AbstractRefreshableConfigApplicationContext (org.springframework.context.suppor
          AbstractXmlApplicationContext (org.springframework.context.support)
                © ← FileSystemXmlApplicationContext (org.springframework.context.support)
                ClassPathXmlApplicationContext (org.springframework.context.support)
          AbstractRefreshableWebApplicationContext (org.springframework.web.context.s
     GenericApplicationContext (org.springframework.context.support)
          GenericXmlApplicationContext (org.springframework.context.support)
        StaticApplicationContext (org.springframework.context.support)
          StaticWebApplicationContext (org.springframework.web.context.support)
        ► GenericWebApplicationContext (org.springframework.web.context.support)
          ☐ ☐ GenericGroovyApplicationContext (org.springframework.context.support)
           AnnotationConfigApplicationContext (org.springframework.context.annotation)

    ConfigurableWebApplicationContext (org.springframework.web.context)

WebApplicationContext (org.springframework.web.context)

    ConfigurableWebApplicationContext (org.springframework.web.context)

        GenericWebApplicationContext (org.springframework.web.context.support)
     StaticWebApplicationContext (org.springframework.web.context.support)

    © 1 AbstractRefreshableWebApplicationContext (org.springframework.web.context.support

      XmlWebApplicationContext (org.springframework.web.context.support)
        GroovyWebApplicationContext (org.springframework.web.context.support)
        AnnotationConfigWebApplicationContext (org.springframework.web.context.suppor
```

。 重量级资源

```
ApplicationContext工厂的对象占用大量内存。
不会频繁的创建对象: 一个应用只会创建一个工厂对象。
ApplicationContext工厂: 一定是线程安全的(多线程并发访问)
```

4. 程序开发

```
1
    1. 创建类型
2
    2. 配置文件的配置 applicationContext.xml
3
       <bean id="person" class="com.baizhiedu.basic.Person"/>
    3. 通过工厂类,获得对象
4
5
       ApplicationContext
              |- ClassPathXmlApplicationContext
6
       ApplicationContext ctx = new
    ClassPathXmlApplicationContext("/applicationContext.xml");
8
       Person person = (Person)ctx.getBean("person");
```

5. 细节分析

• 名词解释

```
1 Spring工厂创建的对象,叫做bean或者组件(componet)
```

• Spring工厂的相关的方法

```
//通过这种方式获得对象,就不需要强制类型转换
 2
     Person person = ctx.getBean("person", Person.class);
     System.out.println("person = " + person);
4
 5
 6
     //当前Spring的配置文件中 只能有一个<bear class是Person类型
7
    Person person = ctx.getBean(Person.class);
 8
    System.out.println("person = " + person);
9
10
     //获取的是 Spring工厂配置文件中所有bean标签的id值 person person1
11
12
    String[] beanDefinitionNames = ctx.getBeanDefinitionNames();
13
    for (String beanDefinitionName : beanDefinitionNames) {
      System.out.println("beanDefinitionName = " +
14
     beanDefinitionName);
15
    }
16
17
18
     //根据类型获得Spring配置文件中对应的id值
19
    String[] beanNamesForType = ctx.getBeanNamesForType(Person.class);
20
    for (String id : beanNamesForType) {
21
      System.out.println("id = " + id);
22
     }
23
24
25
    //用于判断是否存在指定id值得bean
26
    if (ctx.containsBeanDefinition("a")) {
     System.out.println("true = " + true);
27
28
     }else{
29
     System.out.println("false = " + false);
30
    }
31
32
33
    //用于判断是否存在指定id值得bean
34
   if (ctx.containsBean("person")) {
     System.out.println("true = " + true);
35
36
   }else{
      System.out.println("false = " + false);
37
38
```

• 配置文件中需要注意的细节

```
1. 只配置class属性
1
2
    <bean class="com.baizhiedu.basic.Person"/>
3
    a) 上述这种配置 有没有id值 com.baizhiedu.basic.Person#0
4
    b) 应用场景: 如果这个bean只需要使用一次,那么就可以省略id值
5
              如果这个bean会使用多次,或者被其他bean引用则需要设置id值
6
7
    2. name属性
8
9
    作用:用于在Spring的配置文件中,为bean对象定义别名(小名)
    相同:
10
      1. ctx.getBean("id|name")-->object
11
      2. <bean id="" class=""
12
13
         等效
```

```
<bean name="" class=""</pre>
14
15
    区别:
       1. 别名可以定义多个, 但是id属性只能有一个值
16
       2. XML的id属性的值, 命名要求: 必须以字母开头, 字母 数字 下划线 连字符 不
17
    能以特殊字符开头 /person
18
            name属性的值,命名没有要求 /person
          name属性会应用在特殊命名的场景下: /person (spring+struts1)
20
21
          XML发展到了今天: ID属性的限制, 不存在 /person
       3. 代码
22
23
            //用于判断是否存在指定id值得bean,不能判断name值
           if (ctx.containsBeanDefinition("person")) {
25
               System.out.println("true = " + true);
            }else{
               System.out.println("false = " + false);
27
28
29
30
            //用于判断是否存在指定id值得bean,也可以判断name值
           if (ctx.containsBean("p")) {
               System.out.println("true = " + true);
33
34
               System.out.println("false = " + false);
35
36
```

6. Spring工厂的底层实现原理(简易版)

Spring工厂是可以调用对象私有的构造方法创建对象

```
public class Account implements Serializable {
    private String password;
    public String password;
}

public String getName() {
    return name;
}

public void setName(String name) {
    this.name = name;
}

public String getPassword() {
    return password;
}

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

chean id="account" class="com.baizhiedu.injection.Account"/>

public class TestSpring) {
    public class TestSpring) {
        //maiz Tapis, ##ape@acception
        //maiz
```

7. 思考

- 1 问题:未来在开发过程中,是不是所有的对象,都会交给Spring工厂来创建呢?
- 2 回答:理论上 是的,但是有特例:实体对象(entity)是不会交给Spring创建,它是由持久层框架进行创建。

第三章、Spring5.x与日志框架的整合

- 1 Spring与日志框架进行整合,日志框架就可以在控制台中,输出Spring框架运行过程中的一些 重要的信息。
- 2 好处: 便于了解Spring框架的运行过程, 利于程序的调试
- Spring如何整合日志框架

```
1 默认
2 Spring1.2.3早期都是于commons-logging.jar
3 Spring5.x默认整合的日志框架 logback log4j2
4 
5 Spring5.x整合log4j
6 1. 引入log4j jar包
7 2. 引入log4.properties配置文件
```

o pom

```
1
    <dependency>
     <groupId>org.slf4j</groupId>
2
     <artifactId>slf4j-log4j12</artifactId>
3
4
     <version>1.7.25
   </dependency>
5
6
7
   <dependency>
    <groupId>log4j</groupId>
8
9
     <artifactId>log4j</artifactId>
10
      <version>1.2.17
11
    </dependency>
```

o log4j.properties

```
# resources文件夹根目录下
### 配置根
log4j.rootLogger = debug,console

### 日志輸出到控制台显示
log4j.appender.console=org.apache.log4j.ConsoleAppender
log4j.appender.console.Target=System.out
log4j.appender.console.layout=org.apache.log4j.PatternLayout
log4j.appender.console.layout.ConversionPattern=%d{yyyy-MM-dd
HH:mm:ss} %-5p %c{1}:%L - %m%n
```

第四章、注入(Injection)

1. 什么是注入

1 通过Spring工厂及配置文件,为所创建对象的成员变量赋值

1.1 为什么需要注入

通过编码的方式, 为成员变量进行赋值, 存在耦合

1.2 如何进行注入[开发步骤]

- 类的成员变量提供set get方法
- 配置spring的配置文件

1.3 注入好处

1 解耦合

2. Spring注入的原理分析(简易版)

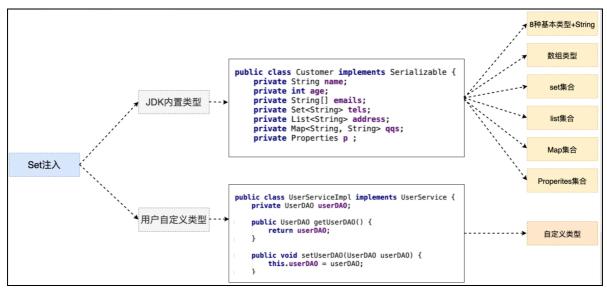
Spring通过底层调用对象属性对应的set方法,完成成员变量的赋值,这种方式我们也称之为set注入

```
public class Account implements Serializable {
      private String name;
private String password;
                                                                                                  <bean id="account" class="com.baizhiedu.injection.Account">
                                                                                                     <value>suns
      public String getName() {
                                                                               password

            return name;
      public void setName(String name) {
                                                                                                       </property>
                                                                                                 </bean>
            this.name = name;
      public String getPassword() {
            return password;
                                                                                               Spring注入的工作原理:
      public void setPassword(String password) {
            this.password = password;
                                                                                               1. <bean id="account" class="xxx.Account"
}
                                                                                               等效于 Account account = new Account();
                                                                                               2. cproperty name="name"
                                                                                               等效于 account.setName("suns")
                                                                                               3. cproperty name="password"
                                                                                                       <value>123456
                                                                                               等效于 account.setPassword("123456")
```

第五章、Set注入详解

```
1针对于不同类型的成员变量,在<property>标签,需要嵌套其他标签23<property>4xxxxx5</property>
```



1. JDK内置类型

1.1 String+8种基本类型

```
1 <value>suns</value>
```

1.2 数组

1.3 Set集合

1.4 List集合

1.5 Map集合

```
注意: map -- entry -- key有特定的标签 <key></key>
 2
                          值根据对应类型选择对应类型的标签
 3
   <map>
 4
     <entry>
      <key><value>suns</value></key><value>3434334343</value>
 5
 6
 7
     </entry>
     <entry>
 8
9
        <key><value>chenyn</value></key>
10
        <ref bean
    </entry>
11
12 </map>
```

1.6 Properites

1 Properties类型 特殊的Map key=String value=String

1.7 复杂的JDK类型 (Date)

需要程序员自定义类型转换器,处理。

2. 用户自定义类型

2.1 第一种方式

- 为成员变量提供set get方法
- 配置文件中进行注入(赋值)

2.2 第二种方式

• 第一种赋值方式存在的问题

```
1 1. 配置文件代码冗余
2 2. 被注入的对象(UserDAO),多次创建,浪费(JVM)内存资源
```

- 为成员变量提供set get方法
- 配置文件中进行配置

```
<bean id="userDA0" class="xxx.UserDA0Impl"/>
1
2
   <bean id="userService" class="xxx.UserServiceImpl">
3
     operty name="userDAO">
4
5
           <ref bean="userDA0"/>
    </property>
6
7
   </bean>
8
   #Spring4.x 废除了 <ref local=""/> 基本等效 <ref bean=""/>
9
```

3. Set注入的简化写法

3.1 基于属性简化

```
JDK类型注入
1
2
   operty name="name">
3
       <value>suns</value>
4
   </property>
5
    cproperty name="name" value="suns"/>
6
7
    注意: value属性 只能简化 8种基本类型+String 注入标签
8
9
   用户自定义类型
   property name="userDAO">
10
11
       <ref bean="userDAO"/>
12
   </property>
13
    property name="userDA0" ref="userDA0"/>
14
```

3.2 基于p命名空间简化

```
1
    JDK类型注入
2
   <bean id="person" class="xxxx.Person">
3
     operty name="name">
4
         <value>suns</value>
 5
     </property>
    </bean>
6
 7
    <bean id="person" class="xxx.Person" p:name="suns"/>
8
    注意: value属性 只能简化 8种基本类型+String 注入标签
9
10
11
   用户自定义类型
    <bean id="userService" class="xx.UserServiceImpl">
12
13
     property name="userDAO">
        <ref bean="userDA0"/>
14
       </property>
15
```

第六章、构造注入

```
1 注入: 通过Spring的配置文件,为成员变量赋值
2 Set注入: Spring调用Set方法 通过配置文件 为成员变量赋值
3 构造注入: Spring调用构造方法 通过配置文件 为成员变量赋值
```

1. 开发步骤

• 提供有参构造方法

```
public class Customer implements Serializable {
   private String name;
   private int age;

public Customer(String name, int age) {
     this.name = name;
     this.age = age;
}
```

• Spring的配置文件

2. 构造方法重载

2.1参数个数不同时

```
1 通过控制<constructor-arg>标签的数量进行区分
```

2.1 构造参数个数相同时

```
1 通过在标签引入 type属性 进行类型的区分 <constructor-arg type="">
```

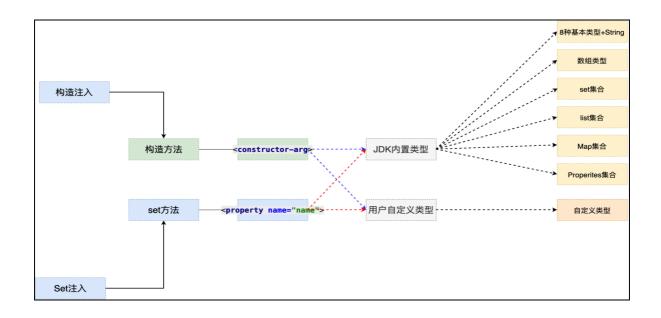
3. 注入的总结

```
      1
      未来的实战中,应用set注入还是构造注入?

      2
      答案: set注入更多

      3
      1. 构造注入麻烦 (重载)

      4
      2. Spring框架底层 大量应用了 set注入
```



第七章、反转控制与 依赖注入

1. 反转(转移)控制(IOC Inverse of Control)

```
1 控制:对于成员变量赋值的控制权
```

2 反转控制:把对于成员变量赋值的控制权,从代码中反转(转移)到Spring工厂和配置文件中完成

3 好处:解耦合

4 底层实现: 工厂设计模式

```
public class UserServiceImpl implements UserService {
   private UserDAO userDAO = new UserDAOImpl();
                                                              直接在代码中 完成对于成员变量的赋值
   @Override
public void login(String name, String password) {
       userDAO.queryUserByNameAndPassword(name, password);
                                                                 对成员变量赋值的控制权=代码
                                                                                                       耦合
   public void register(User user) {
   userDAO.save(user);
public class UserServiceImpl implements UserService {
    private UserDAO userDAO;
    public UserDAO getUserDAO() {
   return userDAO;
                                                            对于成员变量赋值的控制权 = Spring配置文件 + Spring工厂
    public void setUserDAO(UserDAO userDAO) {
   this.userDAO = userDAO;
                                                                        解耦合
<bean id="userDAO" class="com.baizhiedu.UserDAOImpl"/>
```

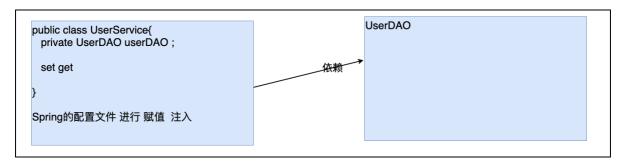
2. 依赖注入 (Dependency Injection DI)

1 注入:通过Spring的工厂及配置文件,为对象(bean,组件)的成员变量赋值

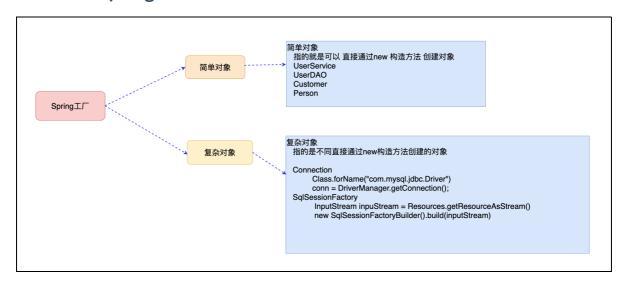
2

3 依赖注入: 当一个类需要另一个类时,就意味着依赖,一旦出现依赖,就可以把另一个类作为本 类的成员变量,最终通过Spring配置文件进行注入(赋值)。

4 好处:解耦合



第八章、Spring工厂创建复杂对象



1. 什么是复杂对象

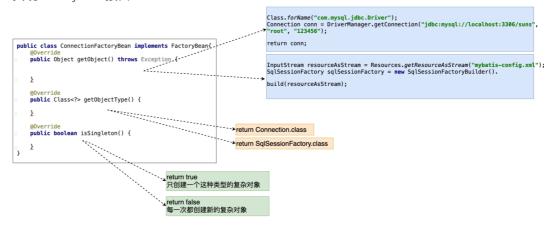
- 1 复杂对象:指的就是不能直接通过new构造方法创建的对象
- 2 Connection
- 3 SqlSessionFactory

2. Spring工厂创建复杂对象的3种方式

2.1 FactoryBean接口

• 开发步骤

o 实现FactoryBean接口



o Spring配置文件的配置

- 1 # 如果Class中指定的类型 是FactoryBean接口的实现类,那么通过id值获得的是 这个类所创建的复杂对象 Connection
- 2 <bean id="conn"
 class="com.baizhiedu.factorybean.ConnectionFactoryBean"/>

细节

- 如果就想获得FactoryBean类型的对象 ctx.getBean("&conn") 获得就是ConnectionFactoryBean对象
- isSingleton方法 返回 true 只会创建一个复杂对象
 返回 false 每一次都会创建新的对象 问题:根据这个对象的特点,决定是返回true (SqlSessionFactory) 还是 false (Connection)
- mysql高版本连接创建时,需要制定SSL证书,解决问题的方式

```
url = "jdbc:mysql://localhost:3306/suns?useSSL=false"
```

· 依赖注入的体会(DI)

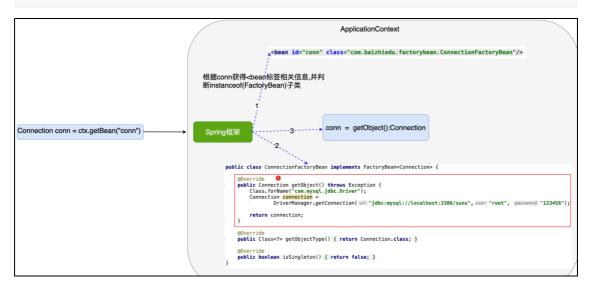
```
把ConnectionFactoryBean中依赖的4个字符串信息 , 进行配置文件的注入
1
2
  好处: 解耦合
  <br/>bean id="conn"
3
   class="com.baizhiedu.factorybean.ConnectionFactoryBean">
4
      roperty name="driverClassName"
    value="com.mysql.jdbc.Driver"/>
      cyroperty name="url" value="jdbc:mysql://localhost:3306/suns?
   useSSL=false"/>
      roperty name="username" value="root"/>
6
7
      roperty name="password" value="123456"/>
8
    </bean>
```

• FactoryBean的实现原理[简易版]

1 接口回调

4

- 2 1. 为什么Spring规定FactoryBean接口 实现 并且 getObject()?
- 3 2. ctx.getBean("conn") 获得是复杂对象 Connection 而没有 获得 ConnectionFactoryBean(&)
- 5 Spring内部运行流程
- 1. 通过conn获得 ConnectionFactoryBean类的对象 , 进而通过instanceof 判断 出是FactoryBean接口的实现类
- 7 2. Spring按照规定 getObject() ---> Connection
- 8 3. 返回Connection



- FactoryBean总结
 - Spring中用于创建复杂对象的一种方式,也是Spring原生提供的,后续讲解Spring整合 其他框架,大量应用FactoryBean

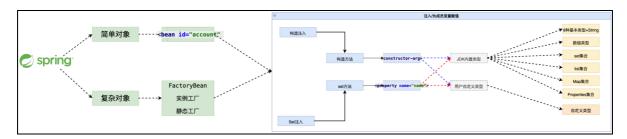
2.2 实例工厂

- 1 1. 避免Spring框架的侵入
- 2 2. 整合遗留系统
- 开发步骤

2.3 静态工厂

• 开发步骤

3. Spring工厂创建对象的总结



第九章、控制Spring工厂创建对象的次数

1. 如何控制简单对象的创建次数

2. 如何控制复杂对象的创建次数

```
1
FactoryBean{

2
isSingleton() {

3
return true 只会创建一次

4
return false 每一次都会创建新的

5
}

6

7
}

8
如没有isSingleton方法 还是通过scope属性 进行对象创建次数的控制
```

3. 为什么要控制对象的创建次数?

- 1 好处: 节省不别要的内存浪费
 - 什么样的对象只创建一次?

```
    SqlSessionFactory
    DAO
    Service
```

• 什么样的对象每一次都要创建新的?

```
    1. Connection
    2. SqlSession | Session
    3. Struts2 Action
```