

* 学习目标

* 能够理解Spring的概述

- * 解决业务层与其他层松耦合问题

- * 2003兴起 , IOC , AOP , ...

* 能够理解Spring 架构图

- * IOC,AOP,Test,JDBC,ORM,WEB,WebSocket,集成其他框架

* 能够理解Spring的优点

- *

* 能够掌握自定义IOC

- * BeanFactory--->Map--->配置--->getBean

- * dom4j,反射

* 能够掌握Spring的IOC的XML开发方式

- * 快速开发

- * 细节 : SpringEL表达式

* 能够掌握Spring的IOC注解的开发方式

- * @Component , @Repository , @Service,@Controller

- * @Value , @Autowired @Qualifier,@Resource,@Scope

- * @Configuration,@ComponentScan,@Bean,@PropertySource,@Import

* 回顾

* MyBatis

* 动态SQL

- * JDBC:拼接SQL语句

- * MyBatis提供动态SQL标签

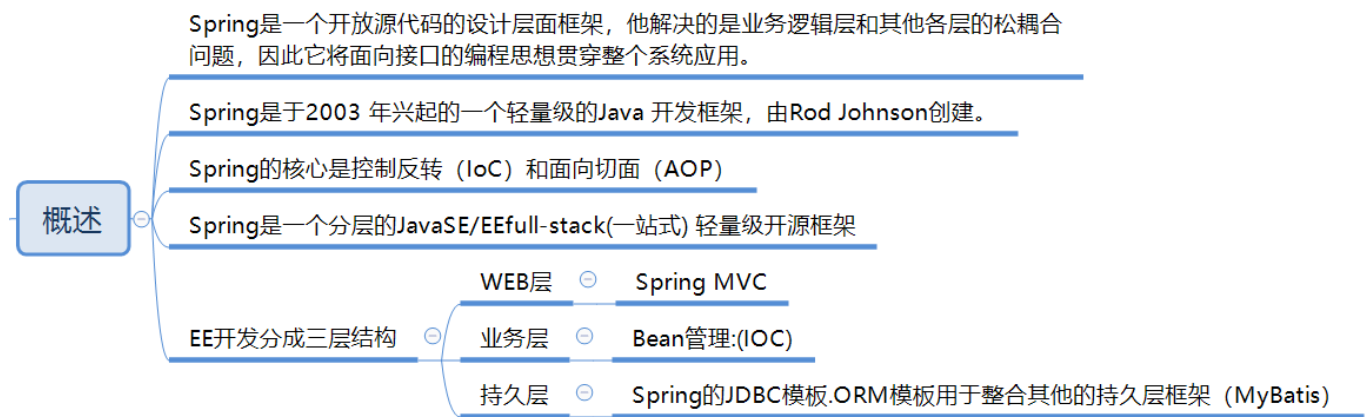
- * if , where , set , foreach,choosewhenotherwise , trim , sql,...

* 缓存

- * 一级缓存 , 二级缓存

* 调用存储过程,...

* 能够理解Spring的概述

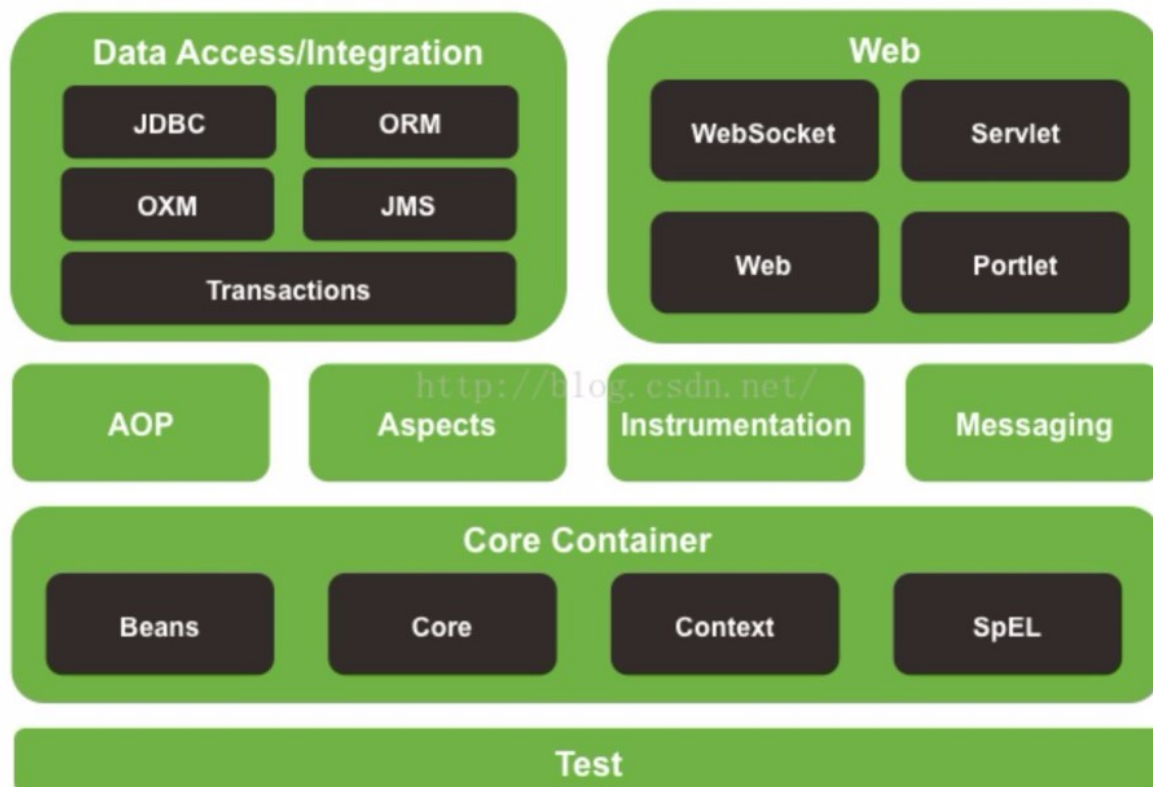


- 1 Spring 介绍:
- 2 <https://baike.baidu.com/item/spring/85061?fr=aladdin>
- 3 Rod Johnson 介绍:
- 4 <https://baike.baidu.com/item/Rod%20Johnson/1423612?fr=aladdin>
- 5 Spring 家族:
- 6 <http://spring.io/projects>

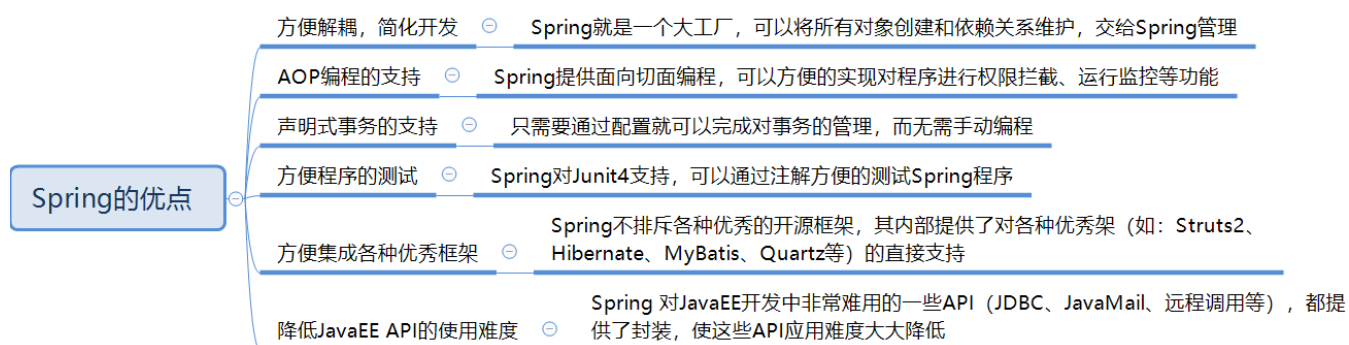
* 能够理解Spring 架构图



Spring Framework Runtime

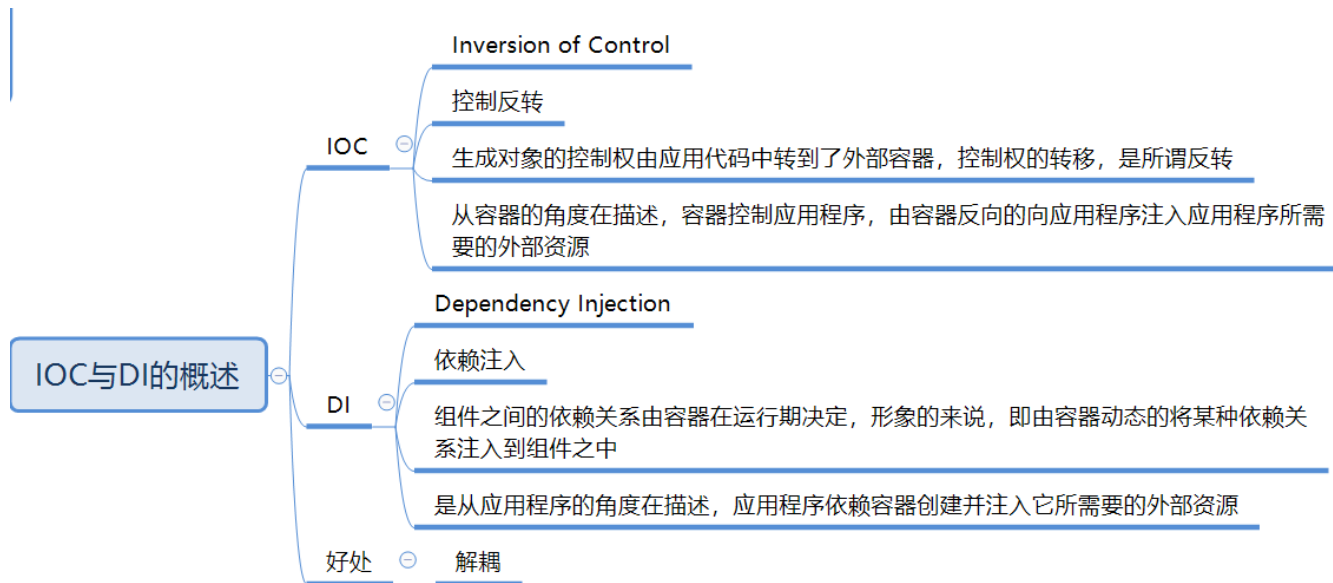


* 能够理解Spring的优点



* 能够掌握自定义IOC

* IOC&DI的概述



* 自定义IOC

BeanFactory：加载配置文件，生成bean对象，提供方法（通过id可以获得对象）

配置文件：xml

```
<beans>
<bean id="userDao" class=".."/>
<bean id="userService" class="...">
<property name="userDao" ref="userDao">
</property>
</bean>
</beans>
```

解析文件：dom4j

反射：产生对象，设置属性
放进容器

提供方法，从容器获取对象

```
1 * 自定义IOC
2 * 前期准备
3 @Data
4 @AllArgsConstructor
5 @NoArgsConstructor
6 public class User implements Serializable {
7     private int id;
8     private String username;
9     private String psw;
10    private String sex;
11 }
```

```

12 public interface UserDao {
13     public void addUser(User user);
14 }
15 public interface UserService {
16     public void reg(User user);
17 }
18 public class UserServiceImpl implements UserService {
19     private UserDao userDao;
20     public UserDao getUserDao() {
21         return userDao;
22     }
23     public void setUserDao(UserDao userDao) {
24         this.userDao = userDao;
25     }
26     @Override
27     public void reg(User user) {
28         userDao.addUser(user);
29     }
30 }
31 * 配置文件
32 <?xml version="1.0" encoding="utf-8"?>
33 <beans>
34     <bean id="userDao" class="com.lg.dao.impl.UserDaoImpl"></bean>
35     <bean id="userService" class="com.lg.service.impl.UserServiceImpl">
36         <property name="userDao" ref="userDao"></property>
37     </bean>
38 </beans>
39 * IOC工厂
40 public class BeanFactory {
41     /**
42      * 存储对象
43      * key: 对象别名（其实就是id）
44      * value: 具体的对象
45      */
46     private static HashMap<String, Object> beans=new HashMap<String, Object>();
47
48     public BeanFactory(String path) {
49         InputStream is = BeanFactory.class.getClassLoader().getResourceAsStream(
50             "applicationContext.xml");
51         SAXReader reader = new SAXReader();
52         try {

```

```

52     Document doc = reader.read(is);
53     // 获得根节点
54     Element root = doc.getRootElement();
55     // 获取所有bean节点
56     List<Element> beanElements = root.elements();
57     for (Element beanElement : beanElements) {
58         // 对象别名
59         String id = beanElement.attributeValue("id");
60         String className = beanElement.attributeValue("class");
61         // 通过反射产生对象
62         Object obj = Class.forName(className).newInstance();
63         // 存放对象
64         beans.put(id, obj);
65     }
66     // 处理: property 标签
67     for (Element beanElement : beanElements) {
68         List<Element> propertyElements = beanElement.elements("property");
69         if (propertyElements != null) {
70             // 处理多个properties标签
71             // 从容器获得当前的对象
72             Object obj = beans.get(beanElement.attributeValue("id"));
73             for (Element propertyElement : propertyElements) {
74                 String name = propertyElement.attributeValue("name");
75                 String ref = propertyElement.attributeValue("ref");
76                 // 从容器获取需要被赋值的对象
77                 Object refObj = beans.get(ref);
78                 // 获得当前的对象的setter方法并把需要被赋值的对象设置进去
79                 // 构建setter的方法
80                 // setUserDao
81                 String setter = "set" + name.substring(0, 1).toUpperCase();
82                 Method method = obj.getClass().getDeclaredMethod(setter, obj.getClass().getClass().getDeclaredMethod(ref));
83                 // 调用setter的方法
84                 method.invoke(obj, refObj);
85             }
86         }
87     }
88 } catch (Exception e) {
89     e.printStackTrace();
90 }
91 }

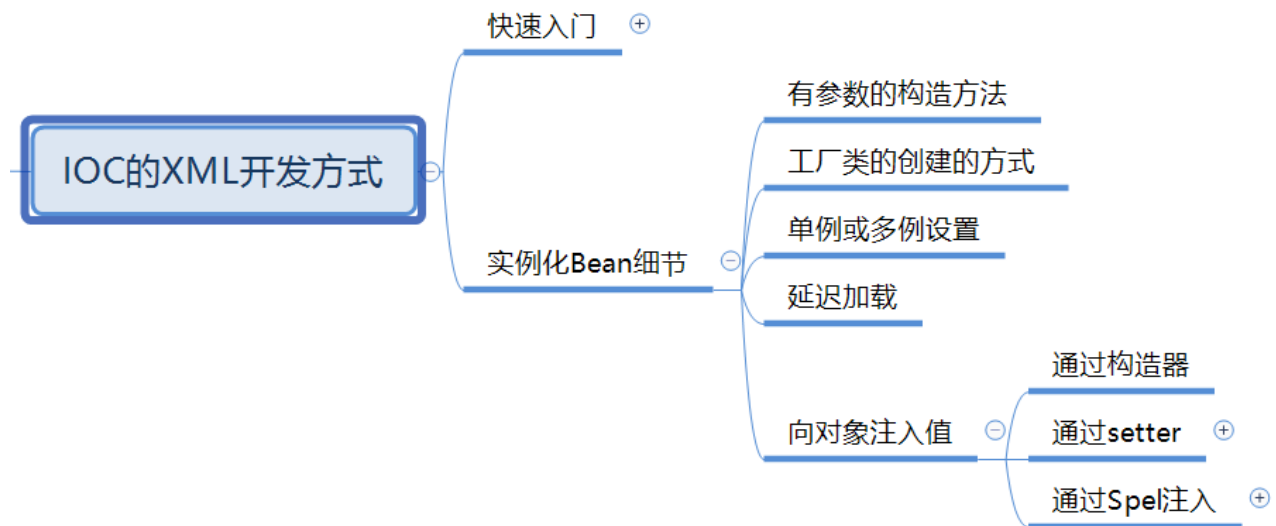
```

```

92     /**
93      * 通过别名获取对象
94      * @param name
95      * @return
96      */
97     public Object getBeans(String name){
98         return beans.get(name);
99     }
100 }
101 * 单元测试
102 @Test
103 public void test1(){
104     String path="beans.xml";
105     BeanFactory beanFactory=new BeanFactory(path);
106     UserService userService = (UserService) beanFactory.getBeans("userService"
107     userService.reg(new User());
108 }

```

* 能够掌握Spring的IOC的XML开发方式



* 快速入门

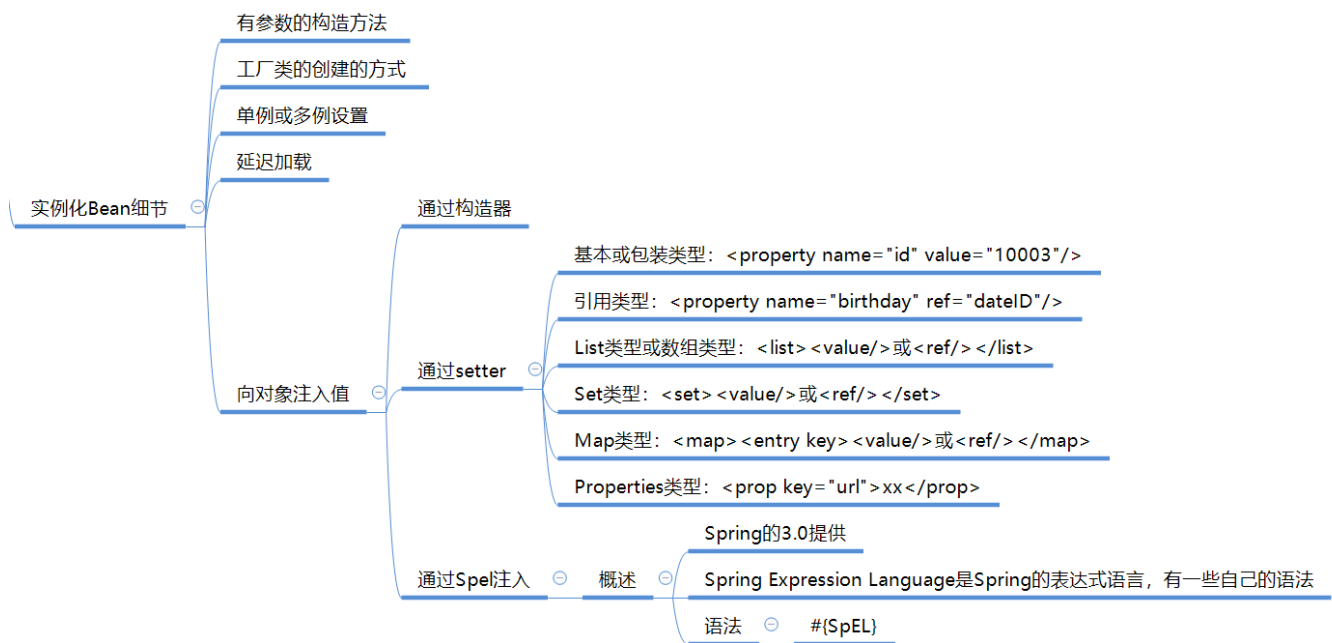


```
1 * 案例一（快速入门）
2 * 前期的准备：
3   * 实体Bean: User, UserDao, UserDaoImpl
4 * 添加依赖
5 <dependency>
6     <groupId>org.springframework</groupId>
7     <artifactId>spring-context</artifactId>
8     <version>5.2.2.RELEASE</version>
9 </dependency>
10 * 温馨提醒：观察依赖，发现aop, beans, core, expression依赖也引进来了
11 * 添加配置文件
12 <?xml version="1.0" encoding="UTF-8"?>
13 <beans xmlns="http://www.springframework.org/schema/beans"
14     xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
15     xsi:schemaLocation="http://www.springframework.org/schema/beans
16         http://www.springframework.org/schema/beans/spring-beans.xsd">
17     <bean id="userDao" class="com.lg.dao.impl.UserDaoImpl"></bean>
18 </beans>
19
20 * 单元测试
21 @Test
22 public void test1(){
23     //了解：早期的做法
24     BeanFactory beanFactory=new XmlBeanFactory(new ClassPathResource("applicatio
25     UserDao userDao = (UserDao) beanFactory.getBean("userDao");
26     userDao.addUser(new User());
```



```
27     }
28     @Test
29     public void test2(){
30         ApplicationContext context=new ClassPathXmlApplicationContext("applicationCo
31         UserDao userDao = (UserDao) context.getBean("userDao");
32         userDao.addUser(new User());
33     }
```

* 实现Bean的细节



```
1 * 案例一（有参数的构造器）
2 * 配置
3 * 方式一
4 <bean id="user" class="com.lg.bean.User">
5     <constructor-arg name="id" value="1"/>
6     <constructor-arg name="username" value="xiaoming"/>
7     <constructor-arg name="psw" value="123"/>
8     <constructor-arg name="sex" value="男"/>
9 </bean>
10 * 方式二
11 <bean id="user" class="com.lg.bean.User">
12     <constructor-arg value="2"/>
13     <constructor-arg value="xiaohei"/>
14     <constructor-arg value="456"/>
15     <constructor-arg value="女"/>
```

```

16 </bean>
17 * 单元测试
18 @Test
19 public void test3(){
20     ApplicationContext context=new ClassPathXmlApplicationContext("applicationC
21     User user = (User) context.getBean("user");
22     System.out.println(user);
23 }
24
25 * 案例二（工厂类创建对象形式）
26 public class UserFactory {
27     public User getInstance(){
28         return new User(10002,"小黑","123123","男");
29     }
30     public static User getStaticInstance(){
31         return new User(10003,"小白","456789","女");
32     }
33 }
34 * 配置
35 <bean id="userFactory" class="com.lg.factory.UserFactory"/>
36 <bean id="u1" factory-bean="userFactory" factory-method="getInstance"/>
37 <bean id="u2" class="com.lg.factory.UserFactory" factory-method="getStaticInsta
38 * 单元测试
39
40 * 案例三（单例或者多例）
41 * 温馨提醒：默认是单例的
42 * 配置
43 <bean id="u3" class="com.lg.bean.User" scope="singleton|prototype"></bean>
44 * 单元测试
45 @Test
46 public void test3(){
47     ApplicationContext context=new ClassPathXmlApplicationContext("applicati
48     User user = (User) context.getBean("u3");
49     User user1 = (User) context.getBean("u3");
50     System.out.println(user==user1);
51 }
52 * 案例四：（延迟加载）
53 * 对于多例来说，都是延迟加载的
54 * 只对单例有效
55 * lazy-init 为false：工厂加载xml之后，会产生对象

```

```

56 * lazy-init 为true: 工厂加载xml之后, 不会产生对象
57 * 代码
58 @Data
59 @AllArgsConstructor
60 public class User implements Serializable {
61     private int id;
62     private String username;
63     private String psw;
64     private String sex;
65     public User(){
66         System.out.println("User 初始化");
67     }
68 }
69 * 配置
70 <bean id="u5" class="com.lg.bean.User" scope="singleton" lazy-init="true"></bean>
71 * 单元测试
72 @Test
73 public void test5(){
74     ApplicationContext context=new ClassPathXmlApplicationContext("applicationContext.xml");
75 }
76
77 * 案例五: 通过setter注入
78 @Data
79 public class TestBean {
80     /**注入Integer类型参数*/
81     private Integer id;
82     /**注入String类型参数*/
83     private String name;
84     /**注入实体Bean*/
85     private User user;
86     /**注入数组*/
87     private Object[] array;
88     /**注入List集合*/
89     private List<Object> list;
90     /**注入Set集合*/
91     private Set<Object> set;
92     /**注入Map键值对*/
93     private Map<Object, Object> map;
94     /**注入properties类型*/
95     private Properties properties;

```

```
96  /**注入空字符串*/
97  private String emptyValue;
98  /**注入null值*/
99  private String nullValue = "";
100 /**检测注入的属性是否全部正确*/
101 public Boolean checkAttr() {
102     if(id == null) {
103         return false;
104     } else {
105         System.out.println("id:" + id);
106     }
107     System.out.println("-----");
108     if(name == null) {
109         return false;
110     } else {
111         System.out.println("name:" + name);
112     }
113     System.out.println("-----");
114     if(user == null) {
115         return false;
116     } else {
117         System.out.println("Bean:" + user.getId() + "|" +
118             user.getUsername()+ "|" + user.getPsw());
119     }
120     System.out.println("-----");
121     if(array == null) {
122         return false;
123     } else {
124         System.out.println("array:");
125         for (Object object : array) {
126             System.out.println(object.toString());
127         }
128     }
129     System.out.println("-----");
130     if(list == null) {
131         return false;
132     } else {
133         System.out.println("list:");
134         for (Object object : list) {
135             System.out.println(object.toString());
```

```
136     }
137 }
138 System.out.println("-----");
139 if(set == null) {
140     return false;
141 } else {
142     System.out.println("set:");
143     for (Object object : set) {
144         System.out.println(object.toString());
145     }
146 }
147 System.out.println("-----");
148 if(map == null) {
149     return false;
150 } else {
151     Set<Map.Entry<Object, Object>> set = map.entrySet();
152     System.out.println("map:");
153     for (Map.Entry<Object, Object> entry : set) {
154         System.out.println(entry.getKey() + "|" + entry.getValue());
155     }
156 }
157 System.out.println("-----");
158 if(properties == null) {
159     return false;
160 } else {
161     Set<Map.Entry<Object, Object>> set = properties.entrySet();
162     System.out.println("properties:");
163     for (Map.Entry<Object, Object> entry : set) {
164         System.out.println(entry.getKey() + "|" + entry.getValue());
165     }
166 }
167 System.out.println("-----");
168 if(!"".equals(emptyValue)) {
169     return false;
170 }
171 System.out.println("-----");
172 if(!(null == nullValue)) {
173     return false;
174 }
175 System.out.println("-----");
```

```
176     System.out.println("全部正确!!!");
177     return true;
178 }
179 }
180
181 * 配置
182 <bean id="test1" class="com.lg.bean.TestBean">
183     <!-- 注入id属性 -->
184     <property name="id" value="1"/>
185     <!-- 使用<![CDATA[]]>标记处理XML特殊字符 -->
186     <property name="name">
187         <!-- 也可以使用P&G -->
188         <value><![CDATA[P&G]]</value>
189     </property>
190     <!-- 定义内部Bean注入 -->
191     <property name="user">
192         <bean class="com.lg.bean.User">
193             <property name="id" value="1"/>
194             <property name="username" value="xiaohei"/>
195             <property name="psw" value="123123"/>
196         </bean>
197     </property>
198     <!-- 注入数组类型 -->
199     <property name="array">
200         <array>
201             <!-- 定义数组元素 -->
202             <value>array01</value>
203             <value>array02</value>
204             <value>array03</value>
205         </array>
206     </property>
207     <!-- 注入List类型 -->
208     <property name="list">
209         <list>
210             <!-- 定义list中元素 -->
211             <value>list01</value>
212             <value>list02</value>
213             <value>list03</value>
214         </list>
215     </property>
```

```
216 <!-- 注入Set类型 -->
217 <property name="set">
218     <set>
219         <!-- 定义set中元素 -->
220         <value>set01</value>
221         <value>set02</value>
222         <value>set03</value>
223     </set>
224 </property>
225 <!-- 注入Map类型 -->
226 <property name="map">
227     <map>
228         <!-- 定义map中的键值对 -->
229         <entry>
230             <key>
231                 <value>mapKey01</value>
232             </key>
233             <value>mapValue01</value>
234         </entry>
235         <entry>
236             <key>
237                 <value>mapKey02</value>
238             </key>
239             <value>mapValue02</value>
240         </entry>
241     </map>
242 </property>
243 <!-- 注入properties类型 -->
244 <property name="properties">
245     <props>
246         <!-- 定义properties中的键值对 -->
247         <prop key="propKey1">propValue1</prop>
248         <prop key="propKey2">propValue2</prop>
249     </props>
250 </property>
251 <!-- 注入空字符串 -->
252 <property name="emptyValue">
253     <value></value>
254 </property>
255 <!-- 注入null值 -->
```

```

256         <property name="nullValue">
257             <null/>
258         </property>
259     </bean>
260 * 单元测试
261 @Test
262 public void test6() {
263     ApplicationContext context = new ClassPathXmlApplicationContext("applicatio
264     TestBean testBean= (TestBean) context.getBean("test1");
265     testBean.checkAttr();
266 }
267
268 * 案例六: Spel的注入
269 public class Person {
270     public String say(String value){
271         System.out.println(value);
272         return value;
273     }
274     public String getName(){
275         return "xiaohei";
276     }
277 }
278 @Data
279 public class SpringEL {
280     private Integer num;
281     private User user;
282     private String sayWhat;
283     private String name;
284     private int rand;
285     private boolean flag;
286 }
287 * 配置
288 <bean id="person" class="com.lg.bean.Person"/>
289 <bean id="el1" class="com.lg.bean.SpringEL">
290     <property name="num" value="#{3*5}"></property>
291     <property name="sayWhat" value="#{person.say('HelloWorld')}"></property>
292     <property name="name" value="#{person.getName()}"></property>
293     <property name="user" value="#{u5}"></property>
294     <property name="rand" value="#{T(Math).random()*100}"></property>
295     <property name="flag" value="#{u5.id==10003}"></property>

```

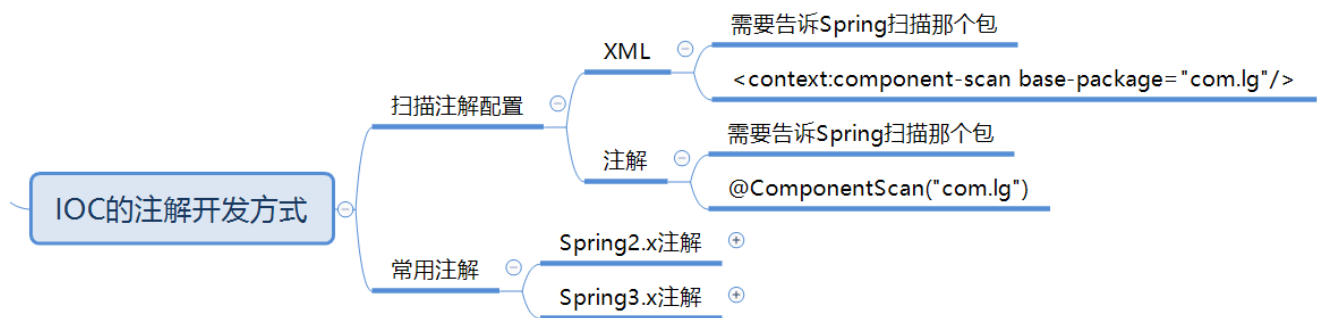


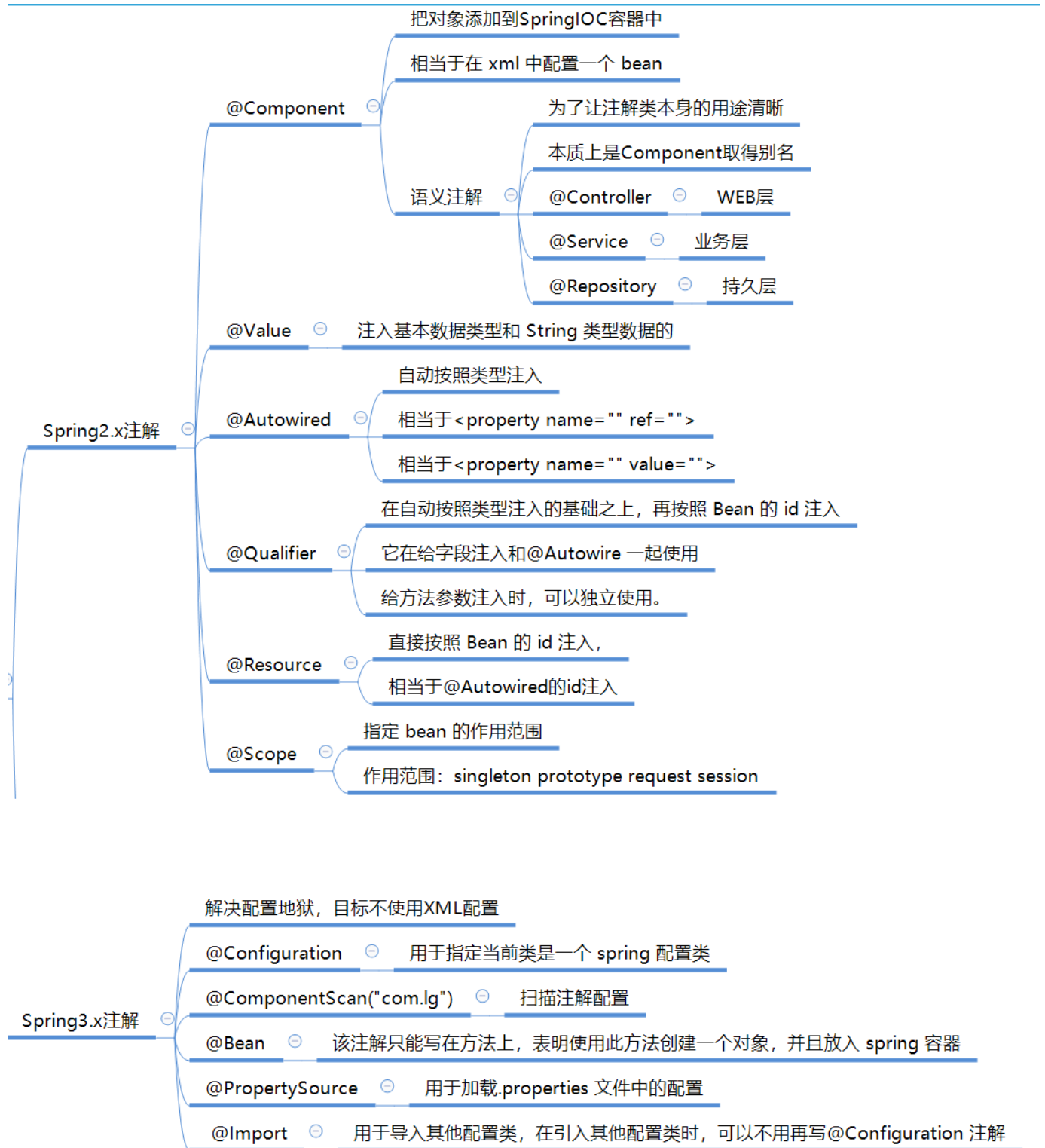
```

296 </bean>
297
298 * 单元测试
299 @Test
300 public void test8() {
301     ApplicationContext context = new ClassPathXmlApplicationContext("application
302     SpringEL el= (SpringEL) context.getBean("el1");
303     System.out.println(el);
304 }
305

```

* 能够掌握Spring的IOC注解的开发方式





```
1 * 案例一 (@Component,@Repository,Service,@Controller) :
2 * 添加配置
3 <beans xmlns="http://www.springframework.org/schema/beans"
4     xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
5     xmlns:context="http://www.springframework.org/schema/context"
6     xsi:schemaLocation="http://www.springframework.org/schema/beans
7     http://www.springframework.org/schema/beans/spring-beans.xsd
8     http://www.springframework.org/schema/context
```

```

9      http://www.springframework.org/schema/context/spring-context.xsd">
10    <context:component-scan base-package="com.lg.*"/>
11    * 代码
12    public interface EmployeeDao {
13        void addEmployee(Employee employee);
14    }
15    @Component("employeeDao")
16    public class EmployeeDaoImpl implements EmployeeDao {
17        @Override
18        public void addEmployee(Employee employee) {
19            System.out.println("addEmployee...");
20        }
21    }
22    * 单元测试
23    @Test
24    public void test8(){
25        ApplicationContext context = new ClassPathXmlApplicationContext("applicationContext.xml");
26        EmployeeDao employeeDao = (EmployeeDao) context.getBean("employeeDao");
27        employeeDao.addEmployee(new Employee());
28    }
29    * 温馨提醒
30    * @Component替换成@Controller或者@Service或者@Repository效果一样
31    它们只是Component取的别名，表达不同的层
32
33    案例二：（@Value, @Autowired, @Qualifier, @Resource）
34    @Repository("employeeDao1")
35    public class EmployeeDaoImpl implements EmployeeDao {
36        @Override
37        public void addEmployee(Employee employee) {
38            System.out.println("addEmployee...");
39        }
40    }
41    @Repository("employeeDao2")
42    public class EmployeeDaoImpl2 implements EmployeeDao {
43        @Override
44        public void addEmployee(Employee employee) {
45            System.out.println("addEmployee2...");
46        }
47    }
48    public interface EmployeeService {

```

```

49     void saveEmployee(Employee employee);
50 }
51 @Service("employeeService")
52 public class EmployeeServiceImpl implements EmployeeService {
53     @Value("5")
54     private int num;
55     @Autowired
56     @Qualifier("employeeDao2")
57     // @Resource(name = "employeeDao2")
58     private EmployeeDao employeeDao;
59     @Override
60     public void saveEmployee(Employee employee) {
61         System.out.println(num);
62         employeeDao.addEmployee(employee);
63     }
64 }
65
66 @Test
67 public void test9(){
68     ApplicationContext context = new ClassPathXmlApplicationContext("applicationContext.xml");
69     EmployeeService employeeService = (EmployeeService) context.getBean("employeeService");
70     employeeService.saveEmployee(new Employee());
71 }
72
73 * 案例三 (@Scope)
74 @Scope("singleton|prototype")
75 public class EmployeeServiceImpl implements EmployeeService {
76     // 单元测试
77     @Test
78     public void test10(){
79         ApplicationContext context = new ClassPathXmlApplicationContext("applicationContext.xml");
80         EmployeeService employeeService = (EmployeeService) context.getBean("employeeService");
81         EmployeeService employeeService1 = (EmployeeService) context.getBean("employeeService");
82         System.out.println(employeeService==employeeService1);
83     }
84     * 温馨提醒：分别测试多例和单例的效果
85
86     * 案例四：（无xml配置，使用注解）
87     * 基本替换xml配置的功能
88 @Configuration

```

```
89 @ComponentScan("com.lg")
90 public class SpringConfiguration {
91     @Bean(name = "user")
92     public User user(){
93         return new User(2,"xiaohei","456","男");
94     }
95     @Bean(name = "array")
96     public Object[] array(){
97         // 跟配置文件有点不一样
98         return new Object[]{"array1","array2"};
99     }
100     @Bean(name = "list")
101     public List<Object> list(){
102         // 跟配置文件有点不一样
103         List<Object> list=new ArrayList<>();
104         list.add("list1");
105         list.add("list2");
106         return list;
107     }
108     @Bean(name = "set")
109     public Set<Object> set(){
110         // 跟配置文件有点不一样
111         Set<Object> set=new HashSet<>();
112         set.add("set1");
113         set.add("set2");
114         return set;
115     }
116     @Bean(name = "map")
117     public Map<Object,Object> map(){
118         Map<Object,Object> map=new HashMap<>();
119         map.put("key1","value1");
120         map.put("key2","value12");
121         return map;
122     }
123     @Bean(name = "properties")
124     public Properties properties(){
125         Properties properties=new Properties();
126         properties.put("p1","value1");
127         properties.put("p2","value12");
128         return properties;
```

```
129     }
130     @Bean(name = "e")
131     public String e(){
132         return "";
133     }
134     @Bean(name = "ne")
135     public String ne(){
136         // 不能注入null, 在@Bean中
137         return "null";
138     }
139     @Bean
140     public Person person(){
141         return new Person();
142     }
143     @Bean
144     public SpringEL el1(){
145         return new SpringEL();
146     }
147 }
148
149 @Data
150 @Component("test1")
151 public class TestBean {
152     /**注入Integer类型参数*/
153     @Value("1")
154     private Integer id;
155     /**注入String类型参数*/
156     @Value("<![CDATA[P&G]]>")
157     private String name;
158     /**注入实体Bean*/
159     @Autowired
160     @Qualifier("user")
161     private User user;
162     /**注入数组*/
163     @Autowired
164     @Qualifier("array")
165     private Object[] array;
166     /**注入List集合*/
167     @Autowired
168     @Qualifier("list")
```

```
169     private List<Object> list;
170     /**注入Set集合*/
171     @Autowired
172     @Qualifier("set")
173     private Set<Object> set;
174     /**注入Map键值对*/
175     @Autowired
176     @Qualifier("map")
177     private Map<Object, Object> map;
178     /**注入properties类型*/
179     @Autowired
180     @Qualifier("properties")
181     private Properties properties;
182     /**注入空字符串*/
183     @Autowired
184     @Qualifier("e")
185     private String emptyValue;
186     /**注入null值*/
187     @Autowired
188     @Qualifier("ne")
189     private String nullValue = "";
190     /**检测注入的属性是否全部正确*/
191     public Boolean checkAttr() {
192         if(id == null) {
193             return false;
194         } else {
195             System.out.println("id:" + id);
196         }
197         System.out.println("-----");
198         if(name == null) {
199             return false;
200         } else {
201             System.out.println("name:" + name);
202         }
203         System.out.println("-----");
204         if(user == null) {
205             return false;
206         } else {
207             System.out.println("Bean:" + user.getId() + "|" +
208                 user.getUsername()+ "|" + user.getPsw());
```

```
209     }
210     System.out.println("-----");
211     if(array == null) {
212         return false;
213     } else {
214         System.out.println("array:");
215         for (Object object : array) {
216             System.out.println(object.toString());
217         }
218     }
219     System.out.println("-----");
220     if(list == null) {
221         return false;
222     } else {
223         System.out.println("list:");
224         for (Object object : list) {
225             System.out.println(object.toString());
226         }
227     }
228     System.out.println("-----");
229     if(set == null) {
230         return false;
231     } else {
232         System.out.println("set:");
233         for (Object object : set) {
234             System.out.println(object.toString());
235         }
236     }
237     System.out.println("-----");
238     if(map == null) {
239         return false;
240     } else {
241         Set<Map.Entry<Object, Object>> set = map.entrySet();
242         System.out.println("map:");
243         for (Map.Entry<Object, Object> entry : set) {
244             System.out.println(entry.getKey() + "|" + entry.getValue());
245         }
246     }
247     System.out.println("-----");
248     if(properties == null) {
```



```

249         return false;
250     } else {
251         Set<Map.Entry<Object, Object>> set = properties.entrySet();
252         System.out.println("properties:");
253         for (Map.Entry<Object, Object> entry : set) {
254             System.out.println(entry.getKey() + "|" + entry.getValue());
255         }
256     }
257     System.out.println("-----");
258     if(!"".equals(emptyValue)) {
259         return false;
260     }
261     System.out.println("-----");
262     if(!(null == nullValue)) {
263         return false;
264     }
265     System.out.println("-----");
266     System.out.println("全部正确!!!");
267     return true;
268 }
269 }
270
271 @Data
272 public class SpringEL {
273     @Value("#{3*5}")
274     private Integer num;
275     @Value("#{user}")
276     private User user;
277     @Value("#{person.say('HelloWorld')}")
278     private String sayWhat;
279     @Value("#{person.getName()}")
280     private String name;
281     @Value("#{T(Math).random()*100}")
282     private int rand;
283     @Value("#{user.id==2}")
284     private boolean flag;
285 }
286
287 * 单元测试
288 public class AppTest2

```

```
289 {
290
291     @Test
292     public void test1(){
293         //了解：早期的做法
294         BeanFactory beanFactory=new AnnotationConfigApplicationContext(SpringCo
295         UserDao userDao = (UserDao) beanFactory.getBean("userDao");
296         userDao.addUser(new User());
297     }
298     @Test
299     public void test2(){
300         ApplicationContext context=new AnnotationConfigApplicationContext(Sprir
301         UserDao userDao = (UserDao) context.getBean("userDao");
302         userDao.addUser(new User());
303     }
304
305     @Test
306     public void test3(){
307         ApplicationContext context=new AnnotationConfigApplicationContext(Sprir
308         User user = (User) context.getBean("user");
309         System.out.println(user);
310     }
311
312     @Test
313     public void test6() {
314         ApplicationContext context = new AnnotationConfigApplicationContext(Spr
315         TestBean testBean= (TestBean) context.getBean("test1");
316         testBean.checkAttr();
317     }
318
319     @Test
320     public void test7() {
321         ApplicationContext context = new AnnotationConfigApplicationContext(Spr
322         SpringEL el= (SpringEL) context.getBean("el1");
323         System.out.println(el);
324     }
325
326     @Test
327     public void test8(){
328         ApplicationContext context = new AnnotationConfigApplicationContext(Spr
```

```

329         EmployeeDao employeeDao = (EmployeeDao) context.getBean("employeeDao1")
330         employeeDao.addEmployee(new Employee());
331     }
332
333     @Test
334     public void test9(){
335         ApplicationContext context = new AnnotationConfigApplicationContext(Spring
336         EmployeeService employeeService = (EmployeeService) context.getBean("en
337         employeeService.saveEmployee(new Employee());
338     }
339
340     @Test
341     public void test10(){
342         ApplicationContext context = new AnnotationConfigApplicationContext(Spring
343         EmployeeService employeeService = (EmployeeService) context.getBean("en
344         EmployeeService employeeService1 = (EmployeeService) context.getBean("e
345         System.out.println(employeeService==employeeService1);
346     }
347 }
348
349 * 案例五： (@Import,@PropertySource)
350 * 配置文件db.properties
351 lg.driver=com.mysql.jdbc.Driver
352 lg.url=jdbc:mysql://localhost:3306/lg01?characterEncoding=utf-8
353 lg.username=root
354 lg.password=root
355
356 * 代码
357 @Configuration
358 @PropertySource("db.properties")
359 @Data
360 public class DbConfig {
361     @Value("${lg.driver}")
362     private String driver;
363     @Value("${lg.url}")
364     private String url;
365     @Value("${lg.username}")
366     private String username;
367     @Value("${lg.password}")
368     private String password;

```

```
369 }
370 @Configuration
371 @ComponentScan("com.lg")
372 @Import(DbConfig.class)
373 public class SpringConfiguration {
374     ....
375     @Bean
376     public DbConfig dbConfig(){
377         return new DbConfig();
378     }
379 }
380 * 单元测试
381 @Test
382 public void test11(){
383     ApplicationContext context = new AnnotationConfigApplicationContext(SpringCor
384     DbConfig dbConfig= (DbConfig) context.getBean("dbConfig");
385     System.out.println(dbConfig);
386 }
```