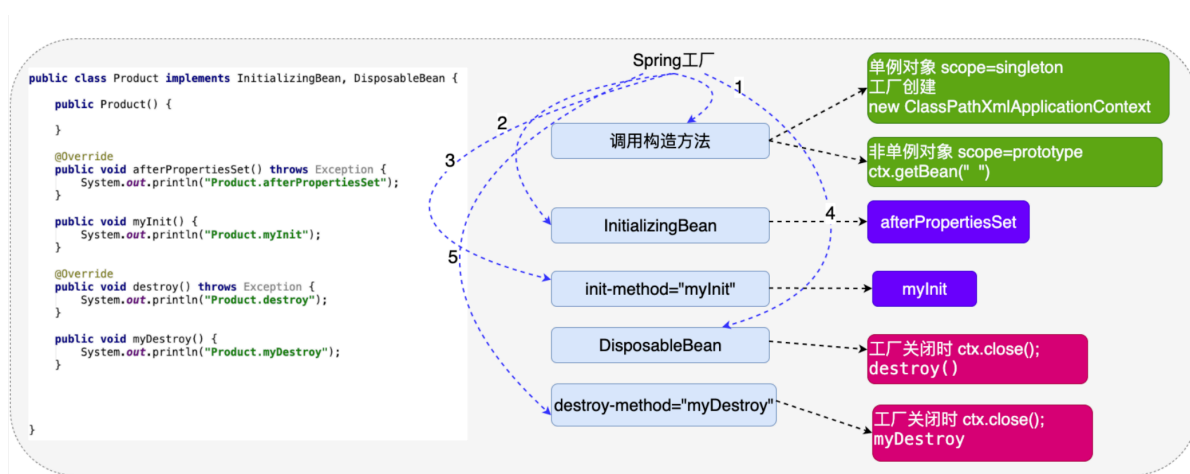


百知教育 — Spring系列课程 — 工厂高级特性

第十章、对象的生命周期



1. 什么是对象的生命周期

- 1 指的是一个对象创建、存活、消亡的一个完整过程

2. 为什么要学习对象的生命周期

- 1 由Spring负责对象的创建、存活、销毁，了解生命周期，有利于我们使用好Spring为我们创建的对象

3. 生命周期的3个阶段

- 创建阶段

- 1 Spring工厂何时创建对象

- scope="singleton"

- 1 Spring工厂创建的同时，对象的创建
- 2
- 3 注意：设置scope=singleton 这种情况下 也需要在获取对象的同时，创建对象
- 4 `<bean lazy-init="true" />`

- scope="prototype"

- 1 Spring工厂会在获取对象的同时，创建对象
- 2 `ctx.getBean(" ")`

- 初始化阶段

- 1 Spring工厂在创建完对象后，调用对象的初始化方法，完成对应的初始化操作
- 2
- 3 1. 初始化方法提供：程序员根据需求，提供初始化方法，最终完成初始化操作
- 4 2. 初始化方法调用：Spring工厂进行调用

- InitializingBean接口

```
1 //程序员根据需求，实现的方法，完成初始化操作
2 public void
  afterPropertiesSet(){
3
4 }
```

- 对象中提供一个普通的方法

```
1 public void myInit(){
2
3 }
4
5 <bean id="product"
  class="xxx.Product" init-
  method="myInit" />
```

- 细节分析

1. 如果一个对象即实现InitializingBean 同时又提供的 普通的初始化方法 顺序

```
1 1. InitializingBean
2 2. 普通初始化方法
```

2. 注入一定发生在初始化操作的前面

3. 什么叫做初始化操作

```
1  资源的初始化：数据库 IO 网络
   . . . . .
```

- 销毁阶段

```
1  Spring销毁对象前，会调用对象的销毁方法，完成销毁操作
2
3  1. Spring什么时候销毁所创建的对象？
4      ctx.close();
5  2. 销毁方法：程序员根据自己的需求，定义
   销毁方法，完成销毁操作
6      调用：Spring工厂完成调用
```

- DisposableBean

```
1  public void destroy()throws
   Exception{
2
3  }
```

- 定义一个普通的销毁方法

```
1  public void myDestroy()throws  
    Exception{  
2  
3  }  
4  <bean id="" class="" init-  
    method="" destroy-  
    method="myDestroy" />
```

- 细节分析

1. 销毁方法的操作只适用于 scope="singleton"
2. 什么叫做销毁操作

```
1  主要指的就是 资源的释放操作  
    io.close()  
    connection.close();
```

第十一章、配置文件参数化

- 1 把Spring配置文件中需要经常修改的字符串信息，转移到一个更小的配置文件中
- 2
- 3 1. Spring的配置文件中存在需要经常修改的字符串？
- 4 存在 以数据库连接相关的参数 代表
- 5 2. 经常变化字符串，在Spring的配置文件中，直接修改
- 6 不利于项目维护(修改)
- 7 3. 转移到一个小的配置文件
- 8 (.properties)
- 9 利于维护(修改)
- 10 配置文件参数化：利于Spring配置文件的维护(修改)

1. 配置文件参数的开发步骤

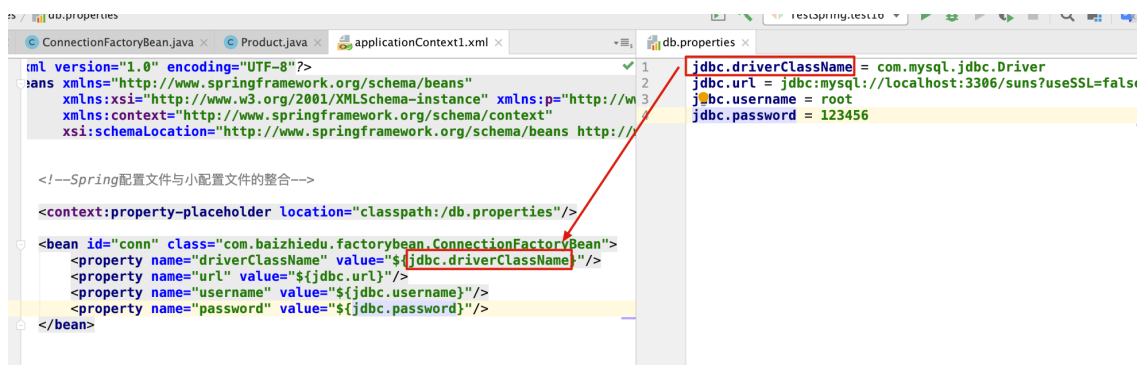
- 提供一个小的配置文件(.properties)

```
1 名字：随便
2 放置位置：随便
3
4 jdbc.driverClassName =
  com.mysql.jdbc.Driver
5 jdbc.url =
  jdbc:mysql://localhost:3306/suns?
  useSSL=false
6 jdbc.username = root
7 jdbc.password = 123456
```

- Spring的配置文件与小配置文件进行整合

```
1 applicationContext.xml
2 <context:property-placeholder
  location="classpath:/db.properties" />
```

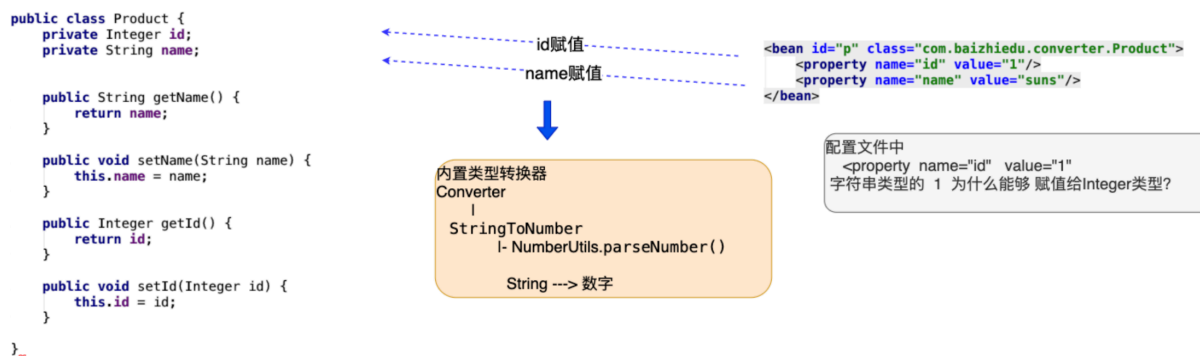
- 在Spring配置文件中通过\${key}获取小配置文件中对应的值



第十二章、自定义类型转换器

1. 类型转换器

- 1 作用：Spring通过类型转换器把配置文件中字符串类型的数据，转换成了对象中成员变量对应类型的数据，进而完成了注入



2. 自定义类型转换器

- 1 原因：当Spring内部没有提供特定类型转换器时，而程序员在应用的过程中还需要使用，那么就需要程序员自己定义类型转换器

- 类 implements Converter接口

```
1 public class MyDateConverter
  implements Converter<String,
    Date> {
2     /*
```



```
3         convert方法作用: String --
->    Date
4
5         SimpleDateFormat sdf = new
SimpleDateFormat();
6
7         sdf.parse(String) ----> Date
8         param:source 代表的是配置文
件中 日期字符串 <value>2020-10-
11</value>
9
10        return : 当把转换好的Date作
为convert方法的返回值后, Spring自动
的为birthday属性进行注入 (赋值)
11
12        */
13        @Override
14        public Date convert(String
source) {
15
16            Date date = null;
17            try {
18                SimpleDateFormat sdf = new
SimpleDateFormat("yyyy-MM-dd");
19                date = sdf.parse(source);
20            } catch (ParseException e) {
```

```
20         e.printStackTrace();
21     }
22     return date;
23 }
24 }
```

- 在Spring的配置文件中配置
 - MyDateConverter对象创建出来

```
1 <bean id="myDateConverter"
    class="xxxx.MyDateConverter" />
```

- 类型转换器的注册

```
1  目的：告知Spring框架，我们所创建的
    MyDateConverter是一个类型转换器
2  <!--用于注册类型转换器-->
3  <bean id="conversionService"
    class="org.springframework.con
    ext.support.ConversionServiceFa
    ctoryBean">
4      <property name="converters">
5          <set>
6              <ref
                bean="myDateConverter" />
7          </set>
8      </property>
9  </bean>
```

3. 细节

- MyDateConverter中的日期的格式，通过依赖注入的方式，由配置文件完成赋值。

```
1
2  public class MyDateConverter
    implements Converter<String,
    Date> {
3      private String pattern;
4
5      public String getPattern() {
```

```
6         return pattern;
7     }
8
9     public void
setPattern(String pattern) {
10         this.pattern = pattern;
11     }
12
13     /*
14         convert方法作用: String --
->   Date
15
16         SimpleDateFormat sdf = new
SimpleDateFormat();
17
18         sdf.parse(String) ----> Date
19         param:source 代表的是配置文
件中 日期字符串 <value>2020-10-
11</value>
20
21         return : 当把转换好的Date作
为convert方法的返回值后, Spring自动
的为birthday属性进行注入 (赋值)
22
23         */
@Override
```

```

24     public Date convert(String
      source) {
25
26         Date date = null;
27         try {
28             SimpleDateFormat sdf
= new SimpleDateFormat(pattern);
29             date =
sdf.parse(source);
30         } catch (ParseException
e) {
31             e.printStackTrace();
32         }
33         return date;
34     }
35 }

```

```

1  <!--Spring创建MyDateConverter类型
   对象-->
2  <bean id="myDateConverter"
   class="com.baizhiedu.converter.My
   DateConverter">
3      <property name="pattern"
   value="yyyy-MM-dd" />
4  </bean>

```

- ConversionServiceFactoryBean 定义 id属性 值必须 conversionService
- Spring框架内置日期类型的转换器

1 日期格式：2020/05/01 （不支持 ： 2020-05-01）

第十三章、后置处理Bean

1 BeanPostProcessor作用：对Spring工厂所创建的对象，进行再加工。

2

3 AOP底层实现：

4

5 注意： BeanPostProcessor接口

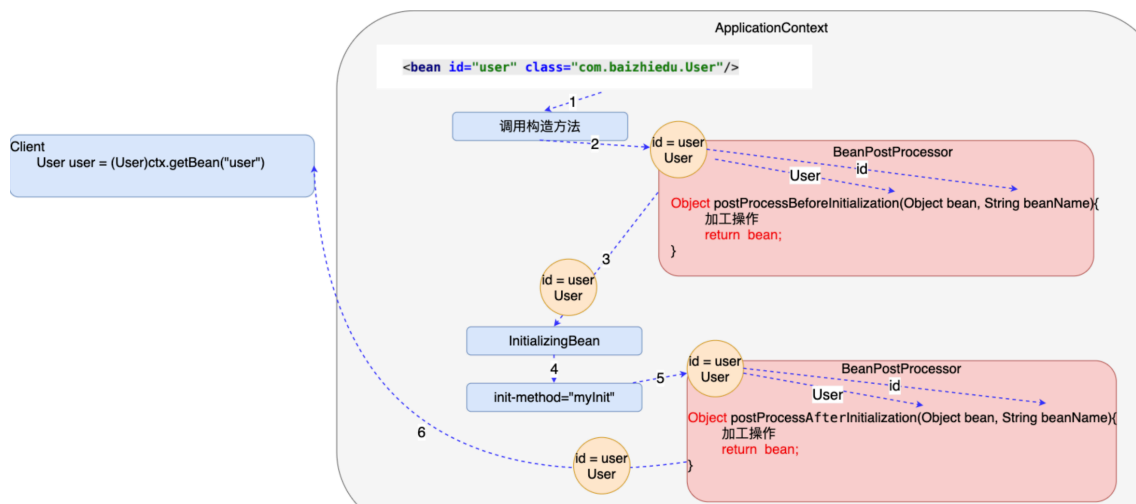
6 `xxxx() {`

7

8

9 `}`

- 后置处理Bean的运行原理分析



- 1 程序员实现BeanPostProcessor规定接口中的方法：
- 2
- 3 `Object`
`postProcessBeforeInitialization(Object bean, String beanName)`
- 4 作用：Spring创建完对象，并进行注入后，可以运行Before方法进行加工
- 5 获得Spring创建好的对象：通过方法的参数
- 6 最终通过返回值交给Spring框架
- 7
- 8 `Object`
`postProcessAfterInitialization(Object bean, String beanName)`
- 9 作用：Spring执行完对象的初始化操作后，可以运行After方法进行加工
- 10 获得Spring创建好的对象：通过方法的参数
- 11 最终通过返回值交给Spring框架
- 12
- 13 实战中：

```
14 很少处理Spring的初始化操作：没有必要区分Before After。只需要实现其中的一个After方法即可
15 注意：
16
    postProcessBeforeInitialization
17    return bean对象
```

- BeanPostProcessor的开发步骤

1. 类实现 BeanPostProcessor接口

```
1  public class
    MyBeanPostProcessor
    implements BeanPostProcessor
    {
2
3        @Override
4        public Object
        postProcessBeforeInitialization(
            Object bean, String
            beanName) throws
            BeansException {
5            return bean;
6        }
7    }
```



```

8         @Override
9         public Object
    postProcessAfterInitializati
on(Object bean, String
    beanName) throws
    BeansException {
10
11             Category category =
    (Category) bean;
12
    category.setName("xiaowb");
13
14
15             return category;
16         }
17     }

```

2. Spring的配置文件中配置

```

1     <bean
    id="myBeanPostProcessor"
    class="xxx.MyBeanPostProcesso
r" />

```

3. BeanPostProcessor细节

- 1 BeanPostProcessor会对Spring工厂中所有创建的对象进行加工。

