Spring概念

1、Spring框架是一个轻量级开源的JavaEE应用程序框架

`轻量级：依赖jar包数量比较少、体积比较小，使用过程中不依赖其他的组件，可以单独使用。`

2、Spring可以解决企业应用开发的复杂性，即框架概念。

3、Spring中两个核心部分：*IOC*和*AOP*

IOC：Inversion of Control `即控制反转。`

`作用：我们不用手动new来创建对象，而是把创建对象的过程交给Spring进行管理。`

AOP：`Aspect Oriented Programming`即面向切面编程。

`作用：不修改源代码进行功能增强`

4、Spring框架特点

- 方便解耦，简化开发
- AOP编程支持
- 方便程序的测试
- 方便集成其他优秀框架 如Mybatis，HIbernate等
- 方便进行事务操作
- 降低API开发难度

Spring入门案例

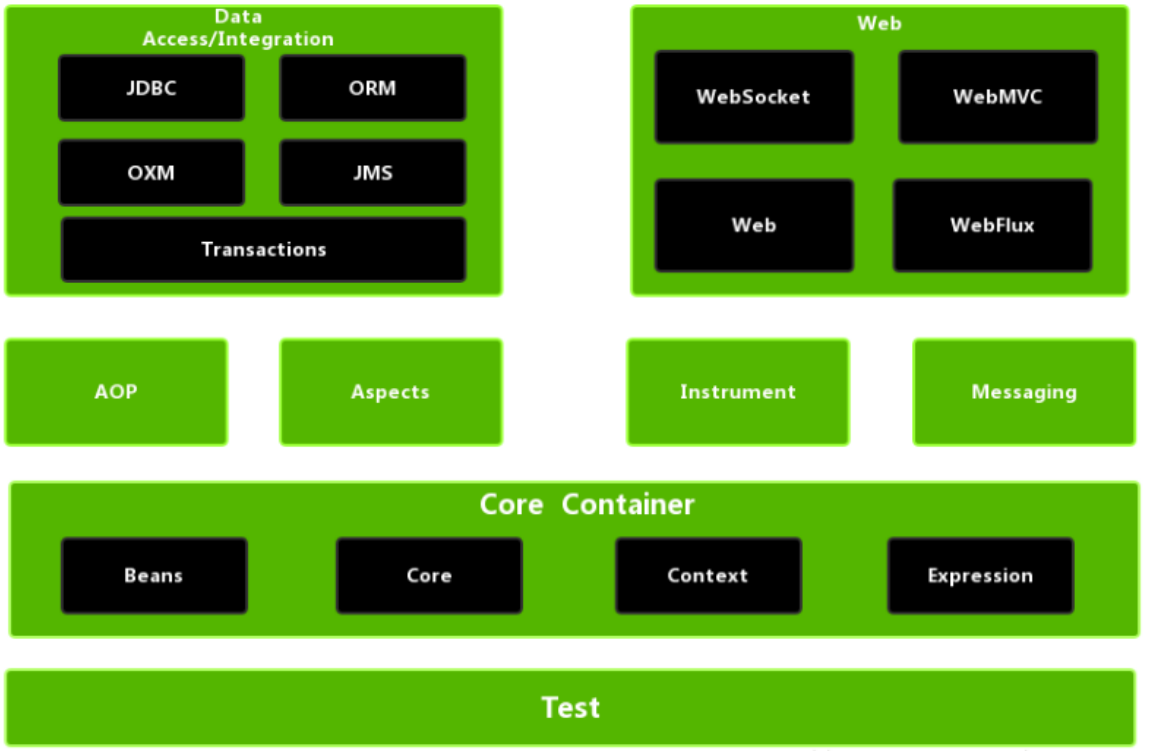
1、spring版本 5.2.6GA

2、下载地址：repo.spring.io

3、建个普通Java工程

4、导入spring框架基础包（必须的4个）

`Beans,Core,Context,Expression`



5、使用spring方式创建对象，创建Spring配置文件，在配置文件配置要创建的对象。

- Spring配置文件是xml格式

```
<!--配置User类的创建-->
<bean id="user" class="com.ly.spring5.User"></bean>
```

6、测试代码

```
 //1、加载spring的配置文件
ApplicationContext context = new ClassPathXmlApplicationContext("bean1.xml");
//2、获取配置要创建的对象
User user = context.getBean("user", User.class);
user.add();
```

IOC容器

容器本质上就是工厂

> IOC控制反转，把对象的创建和对象之间的调用过程，交给Spring进行管理。
>
> 使用IOC目的：降低耦合度
>
> 入门案例就是IOC实现

1、IOC底层原理

演变：普通调用（高耦合） =>工厂模式（中耦合）==>IOC（低耦合）

> - xml解析
> - 工厂设计模式：解耦合
> - 反射

2、IOC接口（BeanFactory）

`IOC思想基于IOC容器，而IOC容器底层就是对象工厂`

Spring提供IOC容器实现的两个方式：（两个接口）

(1)接口：BeanFactory

`Spring内置基本方式，一般是Spring内部使用，不建议Spring外部使用。`

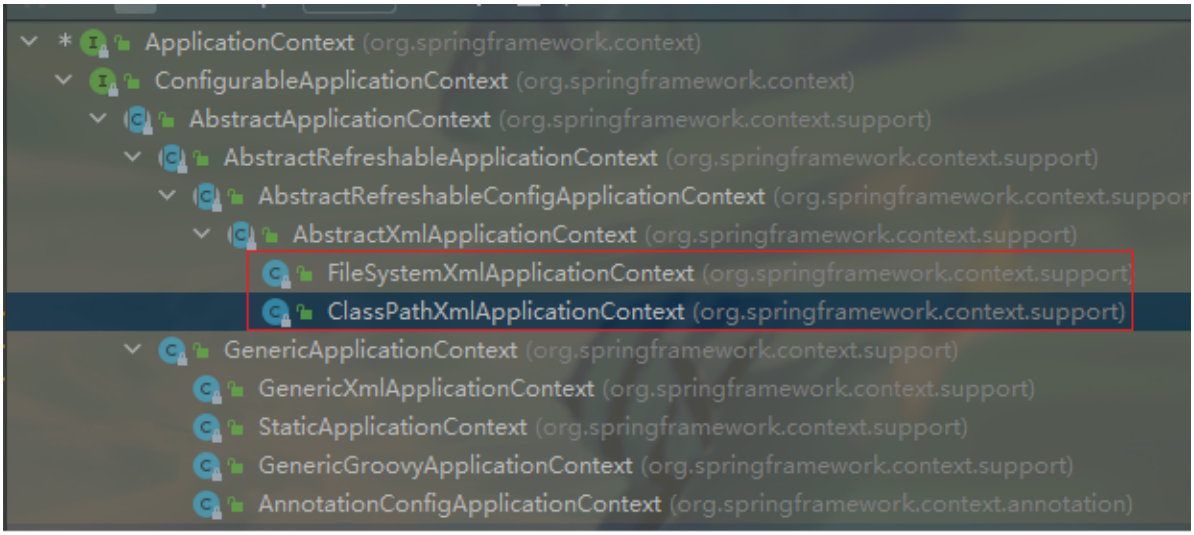
(2)接口：ApplicationContext

`是BeanFactory接口的子接口，功能更多。建议开发人员使用的（即Spring外部）`

两个接口的区别：

> `1、BeanFactory在加载配置文件时不会同时创建配置文件类对象，仅在调用其方法时才会创建。`
>
> `2、ApplicationContext在加载配置文件的同时也会创建配置文件类对象。`
>
> 因此：推荐第二种，把创建对象等耗时耗资源都放在服务启动时。

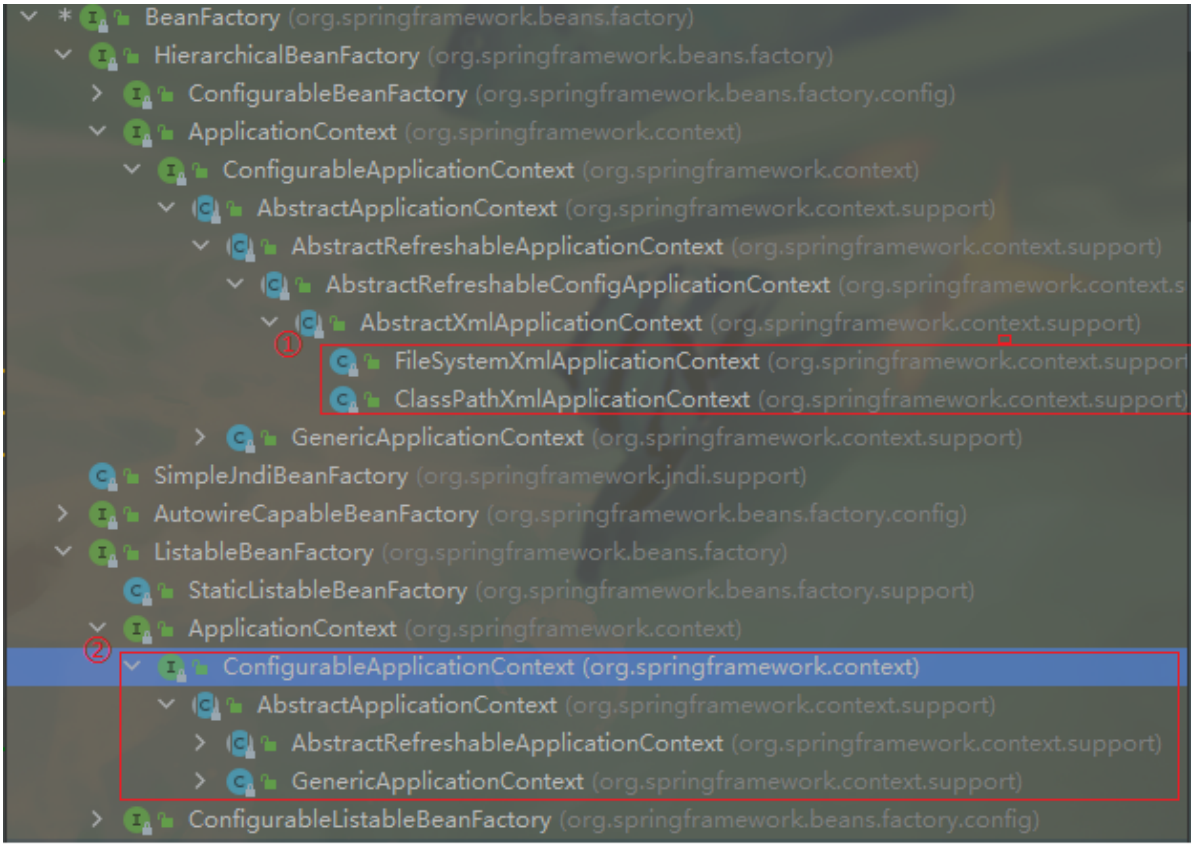
***ApplicationContext*实现类介绍:**



FileSystemXmlApplicationContext：参数是配置文件的绝对路径

ClassPathXmlApplicationContext：参数是配置文件的基于src目录下的路径

***BeanFactory*实现类介绍：**



①实现类和ApplicationContext的实现类一样

②ConfigurableApplicationContext接口及其实现类包含一些扩展功能。

3、IOC操作Bean管理（基于xml/基于注解）

Bean管理：就是下面两个操作

（1）Spring创建对象 ，代替我们手动new

（2）Spring属性注入，代替类中的set方法

3.1、IOC操作Bean管理（基于xml）

***Bean*管理基于*xml*方式创建对象：**

2. 在Spring配置文件中，使用bean标签，标签里添加对应属性即可。

```
<!-- bean标签下属性：
    id：表示类别名，即key 唯一标识
    class：全类名
    name:作用和id一样都是key，但是可以加入特殊符号，id不行
    autowire:自动注入，表示注入属性是选择
             byName ：按属性名称注入
             byType:按类型进行注入
             bystructor:按照构造方法进行注入
             default：默认注入方式
-->
<bean id="user" class="com.ly.spring5.User"></bean>
```

2. Spring创建对象，默认使用无参数构造器

***Bean*管理基于*xml*方式注入属性：**

`DI：Dependency Injection 是IOC容器中一种具体实现，表示依赖注入即注入属性。需要在创建对象的基础上实现。`

Spring注入两种属性方式：即DI

1. set方法注入属性

```
<!-- 在Spring配置文件中使用properties标签 完成属性注入-->
<bean id="book" class="com.ly.spring5.Book">
    <!-- name属性表示Book类中的BookName属性，value表示给其赋值-->
    <property name="bookName" value="猪猪侠"></property>
    <property name="author" value="GG BOY"></property>
</bean>
```

```
//Spring框架获取注入属性
public void test1() {
    //1、加载配置文件
    ApplicationContext context = new
ClassPathXmlApplicationContext("bean1.xml");

    //2、创建对象
    Book book = context.getBean("book", Book.class);
    System.out.println(book);
}
```

2. 有参构造器注入属性

```
<!--    有参构造器注入属性-->
<bean id="order" class="com.ly.spring5.Order">
    <!--        name 属性名称，value自己赋值-->
    <constructor-arg name="orderName" value="易筋经"></constructor-arg>
    <constructor-arg name="address" value="少林寺"></constructor-arg>
    <!-- 通过序号 也可以赋值-->
    <constructor-arg index="0" value="无相神功"></constructor-arg>
</bean>
```

```
public void test2() {
    //1、加载配置文件
    ApplicationContext context = new
ClassPathXmlApplicationContext("bean1.xml");

    //2、建立类对象 即调用构造器
    Order order = context.getBean("order", Order.class);
    System.out.println(order);
}
```

3、p标签注入(了解)，实际上就是set方法的简化

```
<!--1、使用p名称空间注入，可以简化基于xml的配置方法 ,需要在beans标签内加入p属性-->
<!-- 加入 xmlns:p="http://www.springframework.org/schema/p"  -->
<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"
       xsi:schemaLocation="http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans.xsd">

    <!--    p标签 set方法注入属性-->
    <bean id="book" class="com.ly.spring5.Book" p:author="天龙八部" p:bookName="金
庸"></bean>
</beans>
```

***xml*注入其他类型属性：**

1、字面量：定义属性时赋值，或者通过的set方法复制的都均叫做字面量。

(1)字面量为null，怎么操作？

```
<!--        set方法注入属性-->
<bean id="book" class="com.ly.spring5.Book">
    <!-- address属性设置控制，使用null标签-->
    <property name="address">
        <null/>
    </property>
</bean>
```

(2)字面量包含特殊符号，怎么做？

```
<!--        set方法注入属性，属性值可以拆开写-->
<!--        value为 <<南京>>
            两种方法：
                1、转义字符 < &lt   > &gt
                2、CDATA域 <![CDATA[文字域]]>
-->
        <property name="email">
            <value><![CDATA[<<南京>>]]></value>
        </property>
    </bean>
```

2、注入属性 - 外部bean

如：service层调用dao层

```
<!-- 使用Spring配置文件 service层调用dao层方法-->
<!--    注入属性 外部bean-->
<!--
        1、创建UserService和UserDAO对象
        2、给userService对象的userDao属性注入 外部bean对象 通过唯一id来完成
    -->
<bean id="userService" class="com.ly.service.UserService">
    <!-- rel指向bean标签的id-->
    <property name="userDao" ref="userDao">
    </property>
</bean>
<bean id="userDao" class="com.ly.dao.UserDaoImpl"></bean>
```

3、注入内部bean和级联赋值

举例：部门和员工的一对多关系

内部bean

```
<!--    Spring配置文件，注入内部bean-->
    <!-- 和emp中的 bean id="dept"不冲突，因为层级不同-->
    <bean id="dept" class="com.ly.bean.Dept">
        <property name="deptName" value=""></property>
    </bean>


    <bean id="emp" class="com.ly.bean.Employee">
        <property name="empName" value="张三"></property>
        <property name="empGender" value="男"></property>
        <property name="dep">
            <!--  bean写在属性里面就是内部bean ， 如果写在外面用ref连接就是外部bean     --
>
            <bean id="dept" class="com.ly.bean.Dept">
                <property name="deptName" value="财务部"></property>
            </bean>
        </property>
    </bean>
```

级联赋值： 同时向有关联的 属性类 数值（如上面的Employee类中的deptName属性）

```
<bean id="dept" class="com.ly.bean.Dept">
    <property name="deptName" value="哈哈哈"></property>
</bean>

<!--    级联赋值 外部bean-->
<bean id="emp" class="com.ly.bean.Employee">
    <property name="empName" value="张三"></property>
    <property name="empGender" value="男"></property>
    <property name="dep" ref="dept"></property>
</bean>
```

级联赋值，外部bean属性值 第二种写法：

```
<bean id="dept" class="com.ly.bean.Dept">
    <property name="deptName" value="哈哈"></property>
</bean>
<!--    级联赋值 外部bean属性值 第二种写法-->
<bean id="emp1" class="com.ly.bean.Employee">
    <property name="empName" value="李四"></property>
    <property name="empGender" value="男"></property>
    <property name="dep" ref="dept"></property>
    <!--其实就是 类.属性 来赋值
                dep表示  Employee类下的getDep方法，去掉了get
                depName 其实就是Dept类下的setDeptName方法-->
    //此处输出 保安部，因为覆盖了 哈哈 【原因他们是使用同一个Dept对象】
    <property name="dep.deptName" value="保安部"></property>
</bean>

<!--    级联赋值 外部bean-->
```

```
<bean id="emp1" class="com.ly.bean.Employee">
    <property name="empName" value="李四"></property>
    <property name="empGender" value="男"></property>
    //此处也会输出 保安部，因为覆盖了 哈哈 【原因他们是使用同一个Dept对象在整个xml文件中】
    <property name="dep" ref="dept"></property>
</bean>
```

4、xml注入集合属性

注入数组，List集合，Map集合，Set集合

```
    <bean id="stu" class="com.ly.spring5.Stu">
        <!--   注入数组属性     -->
        <property name="courses">
            <array>
                <value>英语</value>
                <value>语文</value>
                <value>数学</value>
                <value>化学</value>
            </array>
        </property>

        <!--   注入List集合属性     -->
        <property name="list" >
            <list>
                <value>list1</value>
                <value>list2</value>
                <value>list3</value>
            </list>
        </property>

        <!--   注入Map集合属性     -->
        <property name="maps" >
            <map>
                <entry key="k1" value="map1"></entry>
                <entry key="k2" value="map2"></entry>
                <entry key="k3" value="map3"></entry>
            </map>
        </property>

        <!--   注入Set集合属性     -->
        <property name="set" >
            <set>
                <value>set1</value>
                <value>set2</value>
            </set>
        </property>
    </bean>
```

细节1：集合里面设置对象类型

```
<bean id="stu" class="com.ly.spring5.collectionType.Stu">
    <!--   注入List集合 属性为类对象 -->
    <property name="courseList" >
        <list>
            //指向外部bean
            <ref bean="c1"></ref>
```

```
            <ref bean="c2"></ref>
        </list>
    </property>

    <!--   注入Map集合属性为类对象      -->
    <property name="mapCourse" >
        <map>
            //和List有些许不同  value-ref="c1"
            <entry key="k1" value-ref="c1"> </entry>
            <entry key="k1" value-ref="c2"> </entry>
        </map>
    </property>
</bean>

//外部bean
<bean id="c1" class="com.ly.spring5.collectionType.Course">
    <property name="cname" value="语文"></property>
</bean>
<bean id="c2" class="com.ly.spring5.collectionType.Course">
    <property name="cname" value="数学"></property>
</bean>
```

细节2：集合注入参数为公共参数，提取出来 其实就是外部bean方法，通过ref连接

- 在Spring配置文件中，引入名称空间（util）

```
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       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">

    <!--  新增两个部分：
        1、xmlns：util...
        2、xsi:schemaLocation...util.. //bean全换成util-->
</beans>
```

- 使用util标签完成List集合注入提取

```
<!--  外部bean     [list集合  注意：需要是同一个类型的才行]-->
    <bean id="course" class="com.ly.spring5.collectionType.Course">
        <property name="cname" value="物理"></property>
    </bean>
    <!--  list集合注入属性，提取出公共部分  -->
    <util:list id="list2">
        <value>list集合</value>
        <value>哈哈</value>
    </util:list>

    <util:list id="bookList">
        <ref bean="course"></ref>
    </util:list>

    <!--   list集合注入属性 通过ref   [list集合  注意：需要是同一个类型的才行]-->
    <bean id="book" class="com.ly.spring5.collectionType.Book">
```

```
        <property name="list" ref="bookList"></property>
    </bean>
```

3.2、IOC操作Bean管理（基于注解）