Spring Framework

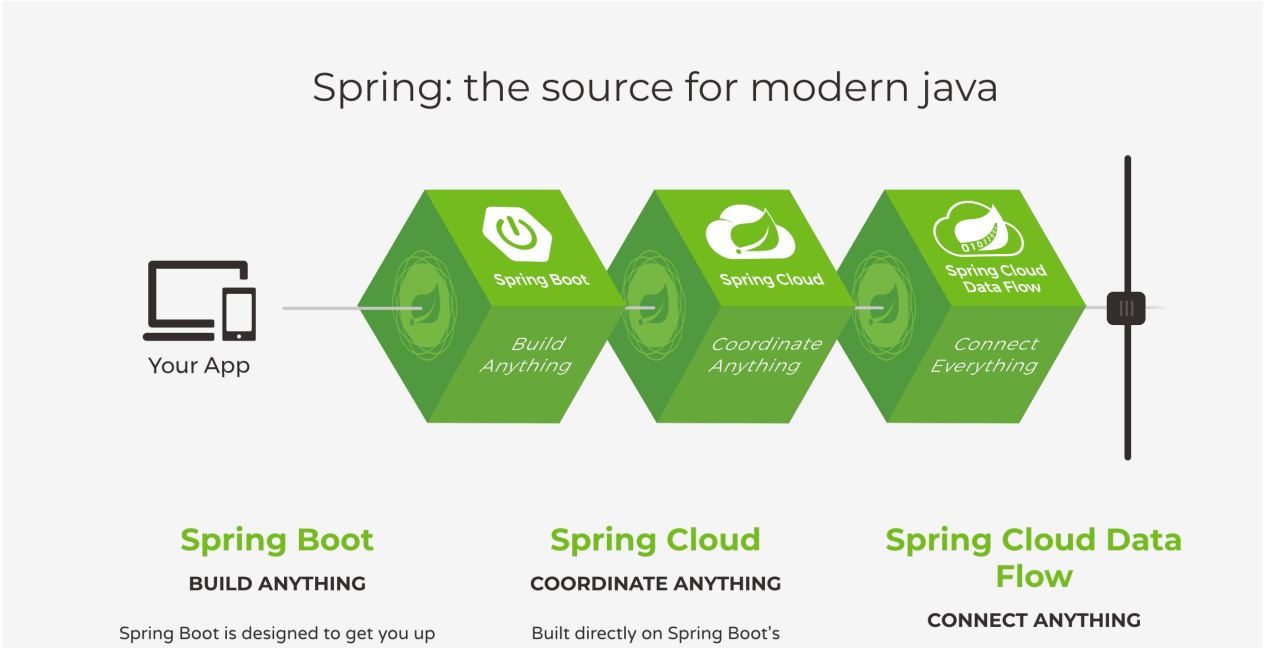
前言

Spring 是当前 Java 开发的行业标准，第一框架。

Spring 概念诞生于 2002 年，于 2003 年正式发布第一个版本 Spring Framework 0.9。

经过十几年的优化迭代，Spring Framework 已经从最初的取代 EJB 的框架逐步发展为一套完整的生态，最新的版本是 5.X。

Spring 架构体系图



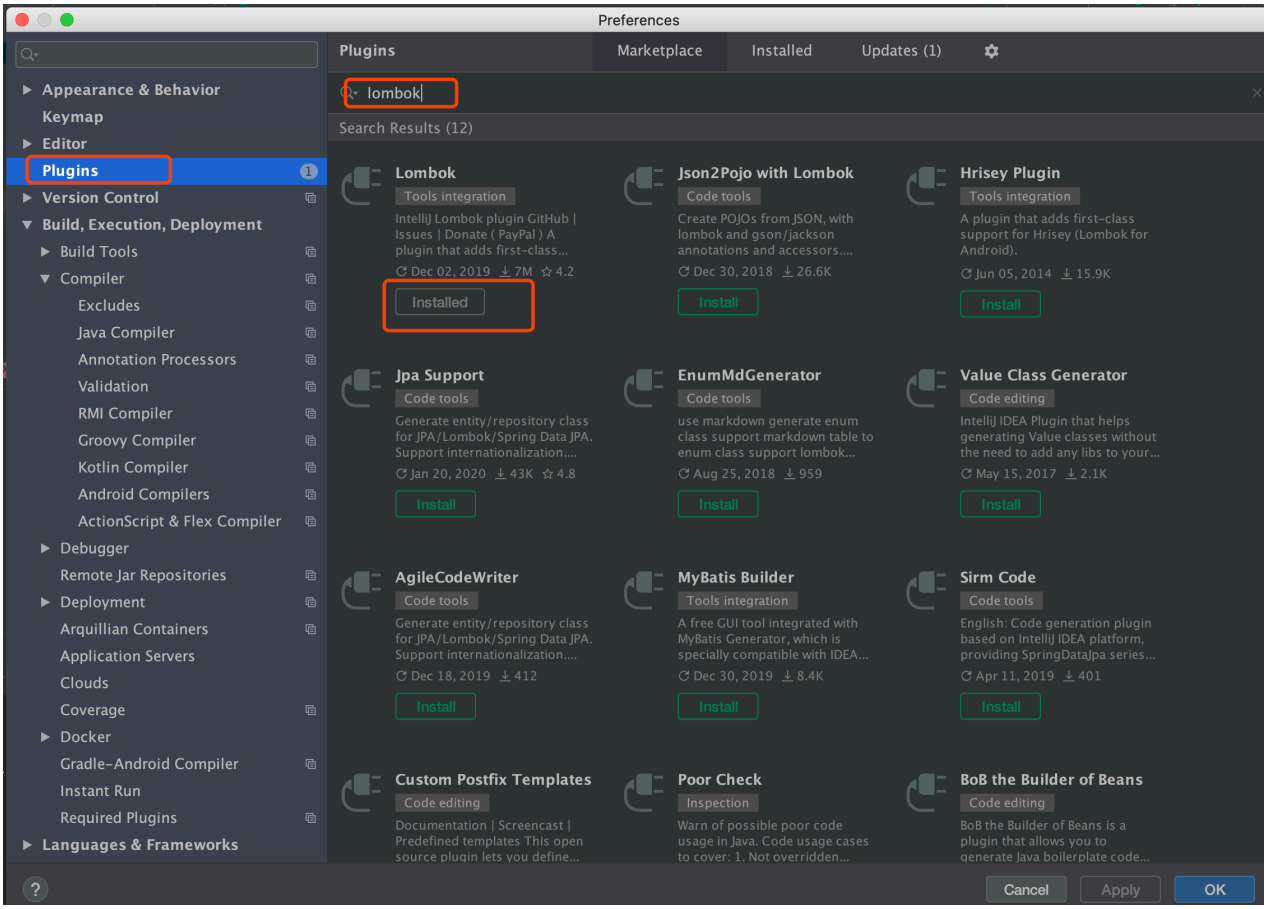
Spring 两大核心机制

- IoC：工厂模式
- AOP：代理模式

IoC

IoC 是 Spring 框架的灵魂，控制反转。 容器：对象无需自己去创建

lombok 可以帮助开发者自动生成实体类相关方法，在 IDEA 中使用，必须预先安装插件。



开发步骤

1、创建 Maven 工程，pom.xml 导入依赖。

```
<dependencies>
  <dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-context</artifactId>
    <version>5.2.3.RELEASE</version>
  </dependency>
</dependencies>
```

2、在 resources 路径下创建 spring.xml。

```
<?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"
       xmlns:p="http://www.springframework.org/schema/p"
       xsi:schemaLocation="http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans-3.2.xsd
   http://www.springframework.org/schema/context
http://www.springframework.org/schema/context/spring-context-4.3.xsd
">

    <bean id="student" class="com.southwind.entity.Student"></bean>

</beans>
```

3、IoC 容器通过读取 spring.xml 配置文件，加载 bean 标签来创建对象。

4、调用 API 获取 IoC 容器中已经创建的对象。

```
ApplicationContext applicationContext = new
ClassPathXmlApplicationContext("spring.xml");
Student student = (Student)applicationContext.getBean("student");
System.out.println(student);
```

IoC 容器创建 bean 的两种方式

- 无参构造函数

```
<bean id="student" class="com.southwind.entity.Student"></bean>
```

给成员变量赋值

```
<bean id="student" class="com.southwind.entity.Student">
  <property name="id" value="1"></property>
  <property name="name" value="张三"></property>
  <property name="age" value="22"></property>
</bean>
```

使用student类中的无参构造创建对象

property标签可以给属性赋值

student类里面有AllArgsConstructor有参构造的注解

- 有参构造函数

```
<bean id="student3" class="com.southwind.entity.Student">
    <constructor-arg index="1" value="王五"></constructor-arg>
    <constructor-arg index="0" value="3"></constructor-arg>
    <constructor-arg index="2" value="18"></constructor-arg>
</bean>
```

构造器参数

如果和有参构造顺序不一样可以用index来指定顺序

```
<bean id="student3" class="com.southwind.entity.Student">
    <constructor-arg name="name" value="王五"></constructor-arg>
    <constructor-arg name="id" value="3"></constructor-arg>
    <constructor-arg name="age" value="18"></constructor-arg>
</bean>
```

从 IoC 容器中取 bean

- 通过 id 取值

```
Student student = (Student)applicationContext.getBean("student");
```

- 通过类型取值。

```
Student student = (Student)applicationContext.getBean(Student.class);
```

当 IoC 容器中同时存在两个以上 Student Bean 的时候就会抛出异常，因为此时没有唯一的 bean。

```
ption in thread "main" org.springframework.beans.factory.NoUniqueBeanDefinitionException: No q
at org.springframework.beans.factory.support.DefaultListableBeanFactory.resolveNamedBean(Defau
at org.springframework.beans.factory.support.DefaultListableBeanFactory.resolveBean(DefaultLis
at org.springframework.beans.factory.support.DefaultListableBeanFactory.getBean(DefaultListabl
at org.springframework.beans.factory.support.DefaultListableBeanFactory.getBean(DefaultListabl
at org.springframework.context.support.AbstractApplicationContext.getBean(AbstractApplicationC
at com.southwind.test.Test.main(Test.java:15)
```

bean 的属性中如果包含特殊字符，如下处理即可

```
<bean id="classes" class="com.southwind.entity.Classes">
    <property name="id" value="1"></property>
    <property name="name">
        <value><![CDATA[<一班>]]></value>
    </property>
</bean>
```

IoC DI

DI 指 bean 之间的依赖注入，设置对象之间的级联关系。

Classes

IOC（控制反转）

全称为：Inverse of Control .将对在自身对象中的一个内置对象的控制反转，反转后不再由自己本身的对象进行控制这个内置对象的创建，而是由第三方系统去控制这个内置对象的创建。简单来说就是把本来在类内部控制的对象，反转到类外部进行创建后注入，不在由类本身镜像控制，这就是IOC的本质

DI（依赖注入）

全称为Dependency Injection，意思是自身对象中的内置对象是通过注入的方式进行创建

IOC和DI的关系

ioc就是容器，di就是注入这一行为，那么di确实就是ioc的具体功能的实现。而ioc则是di发挥的平台和空间。所以说。ioc和di即是相辅相成的拍档。他们都是为了实现解耦而服务的

DI是如何实现的

依赖注入可以通过setter方法注入（设值注入）、构造器注入和接口注入三种方式来实现，Spring支持setter注入和构造器注入，通常使用构造器注入来注入必须的依赖关系，对于可选的依赖关系，则setter注入式更好的选择，setter注入需要类提供无参构造器或者无参的静态工厂方法来创建对象

```
@Data
public class Classes {
    private Integer id;
    private String name;
}
```

Student

```
@Data
@NoArgsConstructor
public class Student {
    private Integer id;
    private String name;
    private Integer age;
    private Classes classes;

    public Student(Integer id, String name, Integer age) {
        System.out.println("通过有参构造创建对象");
        this.id = id;
        this.name = name;
        this.age = age;
    }

    public Student(Integer id, String name) {
        this.id = id;
        this.name = name;
    }
}
```

spring-di.xml

```
<?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"
       xmlns:p="http://www.springframework.org/schema/p"
       xsi:schemaLocation="http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans-3.2.xsd
   http://www.springframework.org/schema/context
http://www.springframework.org/schema/context/spring-context-4.3.xsd
">

    <!-- Classes -->
    <bean id="classes" class="com.southwind.entity.Classes">
        <property name="id" value="1"></property>
        <property name="name" value="一班"></property>
    </bean>
```

```
    <!-- Student -->
    <bean id="student" class="com.southwind.entity.Student">
        <property name="id" value="100"></property>
        <property name="name" value="张三"></property>
        <property name="age" value="22"></property>
        <property name="classes" ref="classes"></property>
    </bean>

</beans>
```

bean 之间的级联需要使用 ref 属性完成映射，而不能直接使用 value ，否则会抛出类型转换异常。

```
y.support.AbstractAutowireCapableBeanFactory.populateBean(AbstractAutowireCapa
y.support.AbstractAutowireCapableBeanFactory.doCreateBean(AbstractAutowireCapa

ion: Cannot convert value of type 'java.lang.String' to required type 'com.sou
nverterDelegate.convertIfNecessary(TypeConverterDelegate.java:262)
ctNestablePropertyAccessor.convertIfNecessary(AbstractNestablePropertyAccessor
```

Classes

```
@Data
public class Classes {
    private Integer id;
    private String name;
    private List<Student> studentList;
}
```

spring.xml

```
<!-- Classes -->
<bean id="classes" class="com.southwind.entity.Classes">
    <property name="id" value="1"></property>
    <property name="name" value="一班"></property>
    <property name="studentList">
        <list>
            <ref bean="student"></ref>
            <ref bean="student2"></ref>
        </list>
    </property>
</bean>

<!-- Student -->
<bean id="student" class="com.southwind.entity.Student">
    <property name="id" value="100"></property>
    <property name="name" value="张三"></property>
    <property name="age" value="22"></property>
```

关联多个

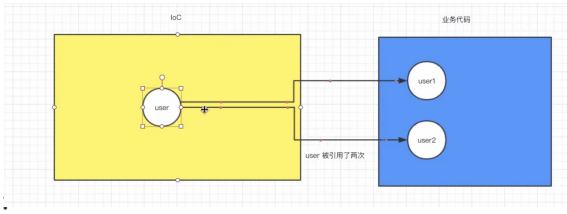
把值赋给age

把bean赋值给classes，ref只能关联一个

```
    <property name="classes" ref="classes"></property>
</bean>

<bean id="student2" class="com.southwind.entity.Student">
    <property name="id" value="200"></property>
    <property name="name" value="李四"></property>
    <property name="age" value="18"></property>
    <property name="classes" ref="classes"></property>
</bean>
```

singleton

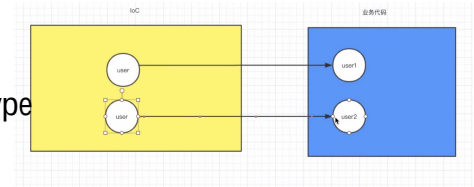


Spring 中的 bean

bean 是根据 scope 来生成，表示 bean 的作用域，scope 有 4 种类型：

- singleton，单例，表示通过 Spring 容器获取的对象是唯一的。默认值。
- prototype，原型，表示通过 Spring 容器获取的对象是不同的。
- request，请求，表示在一次 HTTP 请求内有效。
- session，会话，表示在一个用户会话内有效。

prototype



requset，session 适用于 Web 项目。

singleton 模式下，只要加载 IoC 容器，无论是否从 IoC 中取出 bean，配置文件中的 bean 都会被创建。不管取多少次对象只有一个

prototype 模式下，如果不从 IoC 中取 bean，则不创建对象，取一次 bean，就会创建一个对象。

Spring 的继承

Spring 继承不同于 Java 中的继承，区别：Java 中的继承是针对于类的，Spring 的继承是针对于对象（bean）。

Spring 的继承中，子 bean 可以继承父 bean 中的所有成员变量的值。

user2继承了user1的id

```
<bean id="user1" class="com.southwind.entity.User">
    <property name="id" value="1"></property>
    <property name="name" value="张三"></property>
</bean>

<bean id="user2" class="com.southwind.entity.User" parent="user1">
    <property name="name" value="李四"></property>
</bean>
```

通过设置 bean 标签的 parent 属性建立继承关系，同时子 bean 可以覆盖父 bean 的属性值。

Spring 的继承是针对对象的，所以子 bean 和 父 bean 并不需要属于同一个数据类型，只要其成员变量列表一致即可。