1 IOC

4个核心 jar

- `spring-beans-5.0.7.RELEASE.jar`
- `spring-core-5.0.7.RELEASE.jar`
- `spring-context-5.0.7.RELEASE.jar`
- `spring-expression-5.0.7.RELEASE.jar`
- `commons-logging-1.2.jar`(依赖包)

概念

- 控制反转，把**对象的创建调用**交给 Spring 容器管理
- 实例的创建不再由调用者管理，由 Spring 容器创建，Spring 容器负责控制程序间关系
- 使用 `IOC` 目的: 降低耦合度

IOC底层原理

- `xml` 解析、工厂模式、反射

IOC过程

- `context.xml` 中配置对象

```
<bean id = "myUser" class = "com.bean.User">
```

- 工厂类

```
ApplicationContext ctx = new ClassPathXmlApplicationContext("context.xml") ;
```

- 反射

```
User use = ctx.getBean("myUser”, User.class);
```

2 IOC 两种实现方式

IOC 容器底层就是对象工厂

2.1 BeanFactory

- 是 `IOC` 容器的底层实现，是 `Spring` 内部使用的接口，不供开发人员使用.
- **加载配置文件时不创建对象，获取对象时才创建**

```
BeanFactory bf=new XmlBeanFactory (new ClassPathResource("bean.xml"));
UserDaoImpl ud=(UserDaoImpl) bf.getBean ("userDao"); // 此时才调用UserDaoImpl构造器
```

2.2 ApplicationContext

- 是 `BeanFactory` 接口的子接口，功能更强大，供开发人员使用
- 加载配置文件时就把配置文件声明的对象进行创建
- 通过 `ClassPathXmlApplicationContext` 创建
    - 通过读取类路径下的配置文件创建Spring容器，配置文件必须在类路径下

```
ApplicationContext ctx = new ClassPathXmlApplicationContext ("bean.xml");
```

- 通过`FileSystemXmlApplicationContext` 创建
    - 通过读取类路径下的配置文件创建Spring容器，配置文件必须在类路径下

```
ApplicationContext ctx = new FileSystemXmlApplicationContext("文件的绝对路径");
```

3 Bean 的生命周期管理

- `Spring IOC` 容器负责创建和装配 `Bean` 对象，并对 `Bean` 对象的整个生命周期进行管理
- bean标签的 `init-method` 属性设置初始化方法
- bean标签的 `destroy-method` 属性设置销毁方法

单例对象的生命周期

- `singleton`
- 出生：容器创建时，对象出生。即容器一加载对象就创建
- 活着：只要容器存在，对象就一直活着
- 死亡：容器销毁，对象死亡

多例对象的生命周期

- `prototype`
- 出生：每次使用时，容器创建对象
- 活着：只要对象在使用过程中，对象就一直活着
- 死亡：spring搞不定，交给java垃圾回收机制处理

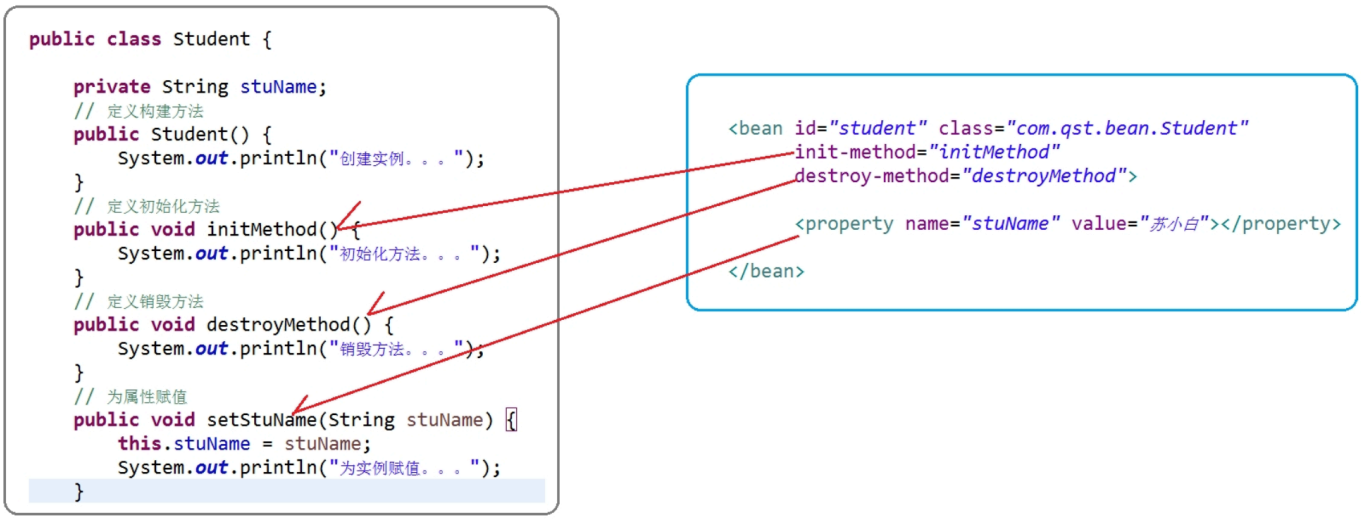
`Bean` 的后置处理器

- 在 `Bean` 的初始化前后，添加一些自定义的业务逻辑，定义一个或多个 `BeanPostProcess` 接口的实现

4 Bean 的作用域

- `singleton`：单例，默认，无论有多少 Bean 引用，始终指向一个
- `prototype`：原型，每调用一次 `getBean` 就创建一个新实例
- `request`：Http 请求中会有各自的实例
- `session`：在一个 session 中一个 Bean 对应一个实例
- `globalSession`：在全局 Session 中，一个 Bean 对应一个实例

```
<bean id = "" class = "" scope = "singleton"> </bean>
```



5 依赖注入

5.1 构造器注入

- 标签：`constructor-arg`，写在 bean 标签内部的子标签
- 属性
    - `name`：注入参数在构造函数中的名称
    - `value`：注入的数据值，只能是基本类型+String
    - `ref`：其他 `bean` 的 id

```
<bean id = "now" class = "java.util.Date"></bean>
<bean id = "userDao" class = "dao.UserDaoImpl">
    <constructor-arg name="name" value="张三"/>
    <constructor-arg name="age" value="18"/>
    <constructor-arg name="birth" ref="now"/>
</bean>
```

5.2 属性注入

- 标签：`property`，写在 bean 标签内部的子标签
- 属性：
    - `name`：指定 set方法的名称
    - `value`：注入的数据值，只能是基本类型+String
    - `ref`：其他bean的id

```
<bean id = "userDao" class = "dao.UserDaoImpl2">
    <property name = "name" value = "jim"/>
    <property name = "age" value = "20"/>
    <property name = "birth" ref = "now"/>
</bean>
<bean id = "now" class = "java.util.Date"></bean>
```

6 Bean 的配置

5.1 基于 XML

- 基于 `XML`：集中式的元数据，与源代码无绑定
- 基于注解：分散式的元数据，与源代码紧绑定
- `Spring` 对 `XML` 配置文件的名称和位置没有特定要求
- 根元素必须是 `<beans>`，需引入 `XML` 命名空间并指定对应的 `schemaLocation`。
- 使用 `Spring` 框架的某个功能时，就需引入其对应的命名空间
- `Bean` 的 `id` 若没有指定，自动将权限定性类名作为 `Bean` 的名字

```
<context:component-scan base-package = "要扫描的包"/>
```

5.2 基于注解

1、用于创建对象

`@Component`、`@Controller`、`@Service`、`@Repository`，写在对应的类上

- `@Component`
    - 作用：相当于在 spring 的 xml 中写一个 `bean` 标签
    - 属性：`value`，指定 `bean` 的 `id`，省略时，默认为首字母小写的类名
    - `@Component` 衍生的 3 个注解：
        - `@Controller`（表现层）
        - `@Service`（业务层）
        - `@Repository`（持久层）

    - 它们的作用和属性与`@Component`完全相同，是 spring 为了提供更明确的语义

2、用于注入数据

- `@Autowired`、`@Qualifier`、`@Resource`、`@value`，写到对应属性上
- `@Autowired`、`@Qualifier`、`@Resource`只能用于注入bean
- `@Value`用户注入基本数据类型 + String
- `@Autowired`: 主要应用在只有唯一一个类型匹配
    - 作用：自动按类型注入，只要容器中有唯一的匹配类型，就可注入成功使用此注解注入时，可以省略`set`方法
    - 属性：`required`, 是否必须注入成功，默认`true`
    - 注意：当有多个属性匹配时，先按照类型找到符合条件的多个对象，再用变量的名称作为bean的id，在这多个对象中继续查找,找到则注入成功，找不到则报错
- `@Qualifier`
    - 作用：在`@Autowired`后按照 bean 的 id 注入
    - 属性：value，指定 bean
    - 注意：不能单独使用，在`@Autowired`后面用
    - 举例：`@Autowired` `@Qualifier("abc")`，注入id为abc的bean
      缺少`@Autowired`会报空指针异常
- `@Resource`：常用
    - 作用：直接按照 bean 的 id 注入
    - 属性：name，指定实例对象的 id
    - 举例：`@Resource (name=" abc" )`

3、用于改变作用范围

`@Scope`

- 作用：用于修改`bean`的作用范围，`singleton`(默认)、`prototype`
- 属性：value

4、生命周期相关：

- @PostConstruct
    - 作用：指定初始化方法，与配置文件中的init-method属性效果一样
- @PreDestroy
    - 作用：指定销毁方法，与配置文件中的destroy-method属性效果一样