

Spring 的依赖

用来设置两个 bean 的创建顺序。

IoC 容器默认情况下是通过 spring.xml 中 bean 的配置顺序来决定创建顺序的，配置在前面的 bean 会先创建。

在不更改 spring.xml 配置顺序的前提下，通过设置 bean 之间的依赖关系来调整 bean 的创建顺序。

```
<bean id="account" class="com.southwind.entity.Account" depends-on="user">
</bean>
<bean id="user" class="com.southwind.entity.User"></bean>
```

上述代码的结果是先创建 User，再创建 Account。

Spring 读取外部资源

实际开发中，数据库的配置一般会单独保存到后缀为 properties 的文件中，方便维护和修改，如果使用 Spring 来加载数据源，就需要在 spring.xml 中读取 properties 中的数据，这就是读取外部资源。

jdbc.properties

```
user = root
password = root
url = jdbc:mysql://localhost:3306/library
driverName = com.mysql.cj.jdbc.Driver
```

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
">

    <!-- 导入外部资源 -->
    <context:property-placeholder location="classpath:jdbc.properties">
</context:property-placeholder>

    <!-- SpEL -->
    <bean id="dataSource" class="com.southwind.entity.DataSource">
```

```
<property name="user" value="${user}"></property>
<property name="password" value="${password}"></property>
<property name="driverName" value="${driverName}"></property>
<property name="url" value="${url}"></property>
</bean>

</beans>
```

Spring p 命名空间

p 命名空间可以用来简化 bean 的配置。

```
<bean id="student" class="com.southwind.entity.Student" p:id="1" p:name="张三"
p:age="22" p:classes-ref="classes">
</bean>

<bean id="classes" class="com.southwind.entity.Classes" p:id="1" p:name="一班">
</bean>
```

Spring 工厂方法

IoC 通过工厂模式创建 bean 有两种方式：

- 静态工厂方法
- 实例工厂方法

区别在于静态工厂类不需要实例化，实例工厂类需要实例化

静态工厂方法

1、创建 Car 类

```
@Data
@AllArgsConstructor
public class Car {
    private Integer num;
    private String brand;
}
```

2、创建静态工厂类、静态工厂方法

```

public class StaticCarFactory{
    private static Map<Integer, Car> carMap;
    static {
        carMap = new HashMap<>();
        carMap.put(1,new Car(1,"奥迪"));
        carMap.put(2,new Car(2,"奥拓"));
    }
    public static Car getCar(Integer num){
        return carMap.get(num);
    }
}

```

3、spring.xml

```

<bean id="car1" class="com.southwind.factory.StaticCarFactory" factory-
method="getCar">
    <constructor-arg value="1"></constructor-arg>
</bean>

```

factory-method 指向静态方法

constructor-arg 的 value 属性是调用静态方法传入的参数

实例工厂方法

1、创建实例工厂类、工厂方法

```

public class InstanceCarFactory {
    private Map<Integer, Car> carMap;
    public InstanceCarFactory(){
        carMap = new HashMap<>();
        carMap.put(1,new Car(1,"奥迪"));
        carMap.put(2,new Car(2,"奥拓"));
    }

    public Car getCar(Integer num){
        return carMap.get(num);
    }
}

```

2、spring.xml

```
<!-- 实例工厂 -->
<bean id="instanceCarFactory"
class="com.southwind.factory.InstanceCarFactory"></bean>
<!-- 通过实例工厂获取Car -->
<bean id="car2" factory-bean="instanceCarFactory" factory-method="getCar" >
    <constructor-arg value="2"></constructor-arg>
</bean>
```

区别：

静态工厂方法创建 Car 对象，不需要实例化工厂对象，因为静态工厂的静态方法，不需要创建对象即可调用，spring.xml 中只需要配置一个 bean，即最终的结果 Car 即可。

实例工厂方法创建 Car 对象，需要实例化工厂对象，因为 getCar 方法是非静态的，就必须通过实例化对象才能调用，所以就必须要创建工厂对象，spring.xml 中需要配置两个 bean，一个是工厂 bean，一个是 Car bean。

spring.xml 中 class + factory-method 的形式是直接调用类中的工厂方法

spring.xml 中 factory-bean + factory-method 的形式则是调用工厂 bean 中的工厂方法，就必须先创建工厂 bean。

Spring IoC 自动装载 autowire

自动装载是 Spring 提供了一种更加简便的方式来完成 DI，不需要手动配置 property，IoC 容器会自动选择 bean 完成注入。

自动装载有两种方式：

- byName，通过属性名完成自动装载。
- byType，通过属性对应的数据类型完成自动装载。

byName 的操作如下所示。

1、创建 Person 实体类。

```
@Data
public class Person {
    private Integer Id;
    private String name;
    private Car car;
}
```

2、在 spring.xml 中配置 Car 和 Person 对应的 bean，并且通过自动装载完成依赖注入。

```

<bean id="person" class="com.southwind.entity.Person" autowire="byName">
    <property name="id" value="1"></property>
    <property name="name" value="张三"></property>
</bean>

<bean id="car" class="com.southwind.entity.Car">
    <constructor-arg name="num" value="1"></constructor-arg>
    <constructor-arg name="brand" value="奥迪"></constructor-arg>
</bean>

```

byType 的操作如下所示。

```

<bean id="person" class="com.southwind.entity.Person" autowire="byType">
    <property name="id" value="1"></property>
    <property name="name" value="张三"></property>
</bean>

<bean id="car2" class="com.southwind.entity.Car">
    <constructor-arg name="num" value="1"></constructor-arg>
    <constructor-arg name="brand" value="奥迪"></constructor-arg>
</bean>

```

使用 byType 进行自动装载时，必须保证 IoC 中只有一个符合条件的 bean，否则会抛出异常。

```

.NoUniqueBeanDefinitionException: No qualifying bean
fig.DependencyDescriptor.resolveNotUnique(DependencyD
port.DefaultListableBeanFactory.doResolveDependency(D
port.DefaultListableBeanFactory.resolveDependency(Def
port.AbstractAutowireCapableBeanFactory.autowireByTyp

```

Spring IoC 基于注解的开发

Spring IoC 的作用是帮助开发者创建项目中所需要的 bean，同时完成 bean 之间的依赖注入关系，DI。

实现该功能有两种方式：

- 基于 XML 配置。
- 基于注解。

基于注解有两步操作，缺一不可：

- 1、配置自动扫包。
- 2、添加注解。

```
<context:component-scan base-package="com.southwind.entity">
</context:component-scan>
```

```
@Data
@Component
public class Repository {
    private DataSource dataSource;
}
```

DI

```
@Data
@Component
public class DataSource {
    private String user;
    private String password;
    private String url;
    private String driverName;
}
```

```
@Data
@Component(value = "myrepo")
public class Repository {
    @Autowired
    private DataSource dataSource;
}
```

@Autowired 默认是通过 byType 进行注入的，如果要改为 byName，需要配置 @Qualifier 注解来完成

```
@Autowired
@Qualifier(value = "ds")
private DataSource dataSource;
```

表示将 IoC 中 id 为 ds 的 bean 注入到 repository 中。

实体类中普通的成员变量（String、包装类等）可以通过 @Value 注解进行赋值。

```

@Data
@Component(value = "ds")
public class DataSource {
    @Value("root")
    private String user;
    @Value("root")
    private String password;
    @Value("jdbc:mysql://localhost:3308/library")
    private String url;
    @Value("com.mysql.cj.Driver")
    private String driverName;
}

```

等同于 spring.xml

```

<bean id="ds" class="com.southwind.entity.DataSource">
    <property name="user" value="root"></property>
    <property name="password" value="root"></property>
    <property name="url" value="jdbc:mysql://localhost:3306/library"></property>
    <property name="driverName" value="com.mysql.cj.jdbc.Driver"></property>
</bean>

```