一、丑陋代码

1、代码耦合度高

```
public class EmployeeServiceImpl implements IEmployeeService {
    private IEmployeeDAO employeeDAO;

    public EmployeeServiceImpl(){
        employeeDAO = new EmployeeDAOJdbcImpl();
    }

    public void save(Employee employee){
        // TODO
    }

    // ...
}
```

此时如果把 IEmployeeDAO 的实现类换成 EmployeeDAOMyBatisImpl，此时需要修改 EmployeeServiceImpl 的源代码，不符合开闭原则。
开闭原则：对于扩展是开放的，对于修改是关闭的。 开闭原则的好处：可维护性。

2、控制事务繁琐

考虑一个应用场景：需要对系统中的某些业务方法做事务管理，拿 save 方法举例。

```
public class EmployeeServiceImpl implements IEmployeeService {
    private IEmployeeDAO employeeDAO;

    public EmployeeServiceImpl(){
        employeeDAO = new EmployeeDAOJdbcImpl();
    }

    public void save(Employee employee){
        // 打开资源
        // 开启事务
        try {
            // 保存业务操作
            // 提交事务
        }catch (Exception e){
            // 回滚事务
        }finally{
            // 释放资源
        }
    }

    // ...
}
```

但问题，若很多方法都要加事务的话，就会存在大量的重复代码分散不同类的不同方法中，不利于维护。

3、第三方框架运用太麻烦

单独使用 MyBatis 框架的保存操作代码如下：

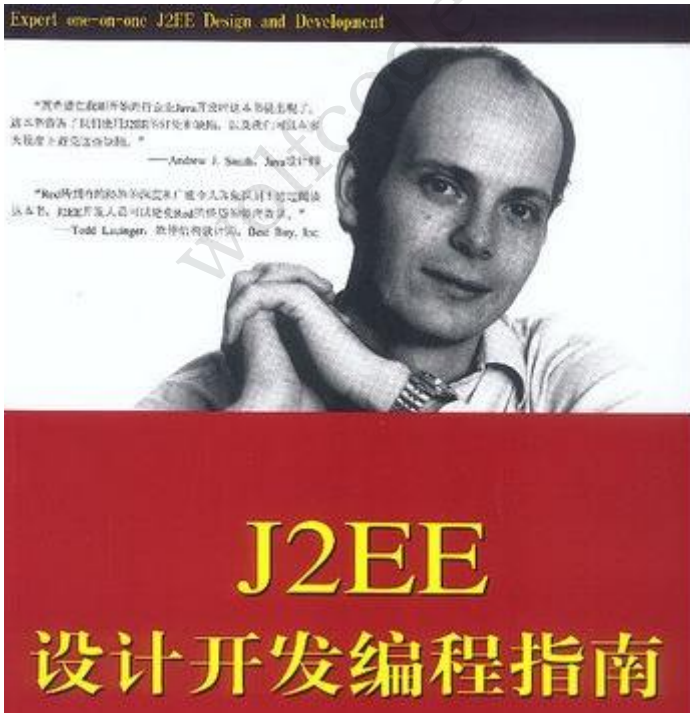
```
SqlSession session = MyBatisUtil.getSession();
EmployeeMapper employeeMapper = session.getMapper(EmployeeMapper.class);
Employee employee = new Employee();
employee.setSalary(new BigDecimal("900"));
employee.setId(1L);
employeeMapper.update(employee);
session.commit();
session.close();
```

对使用者而言最关心的是获取到 EmployeeMapper 对象使用，而不关心这个对象创建。

二、Spring 介绍

1、Spring 定义

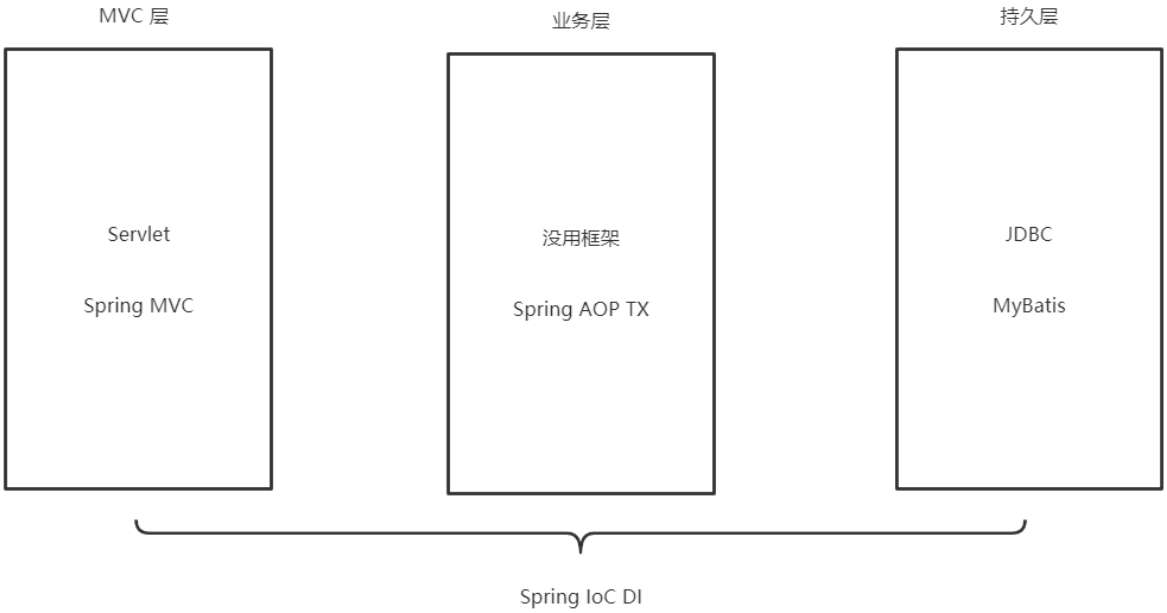
源于 Rod Johnson 在其著作 《Expert one on one J2EE design and development》 中阐述的部分理念和原型衍生而来。



Spring 是一个轻量级的 **IoC / DI和 AOP 容器**的开源框架，致力于构建轻量级的 JavaEE 应用，简化应用开发，本身涵盖了传统应用开发，还拓展到移动端，大数据等领域。
什么是容器（Container）：从程序设计角度看就是**装对象的的对象，因为存在放入、拿出等操作，所以容器还要管理对象的生命周期**，如 Tomcat 就是 Servlet 的容器。

2、Spring 在 JavaEE开发中作用

Spring 提供了 JavaEE 每一层的解决方案（full stack）。Web 开发中的最佳实践：根据职责的纵向划分：控制层、业务层、持久层：不同框架解决不同领域的问题。

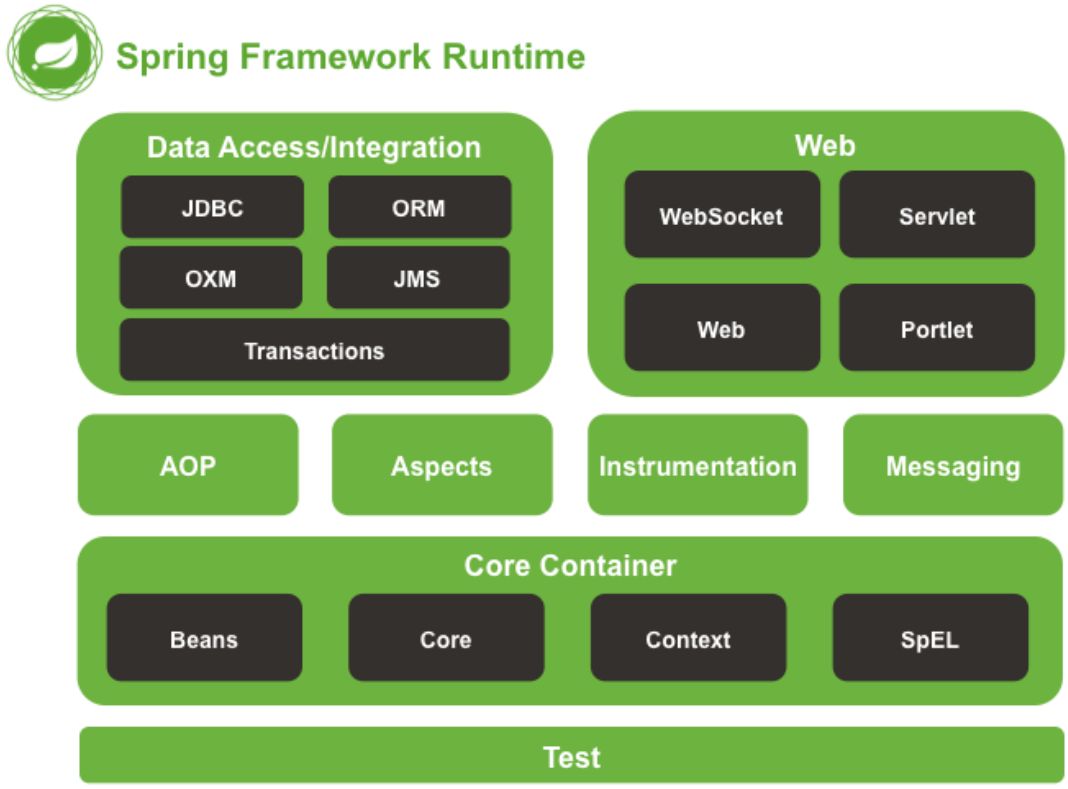


3、Spring 优势

Spring 除了不能帮我们写业务逻辑，其余的几乎什么都能帮助我们简化开发，作用如下

- Spring 能帮我们低侵入/低耦合地根据配置文件创建及组装对象之间的依赖关系。
- Spring 面向切面编程能帮助我们无耦合的实现日志记录，性能统计，安全控制等。
- Spring 能非常简单的且强大的声明式事务管理。
- Spring 提供了与第三方数据访问框架（如Hibernate、JPA）无缝集成，且自己也提供了一套 JDBC 模板来方便数据库访问。
- Spring 提供与第三方 Web（如 Struts1/2、JSF）框架无缝集成，且自己也提供了一套 Spring MVC 框架，来方便 Web 层搭建。
- Spring 能方便的与如 Java Mail、任务调度、缓存框架等技术整合，降低开发难度。

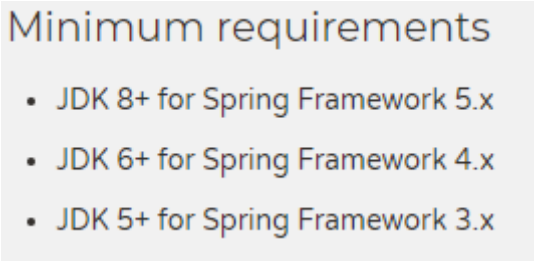
4、Spring FrameWork 体系



- Core Container（核心容器 IoC）包含有 Beans、Core、Context 和 SpEL 模块。
- Test 模块支持使用 JUnit 和 TestNG 对 Spring 组件进行测试。
- AOP 模块提供了一个符合 AOP 联盟标准的面向切面编程的实现。
- Data Access / Integration 层包含有 JDBC、ORM、OXM、JMS 和 Transaction 模块。
- Web 层包含了 Web、Web-Servlet、WebSocket、Web-Porlet 模块。

课程重点设涉及 Test、Core Container、AOP、TX、MVC。

5、Spring 依赖



这里我们使用 **5.0.8.RELEASE** 的版本。

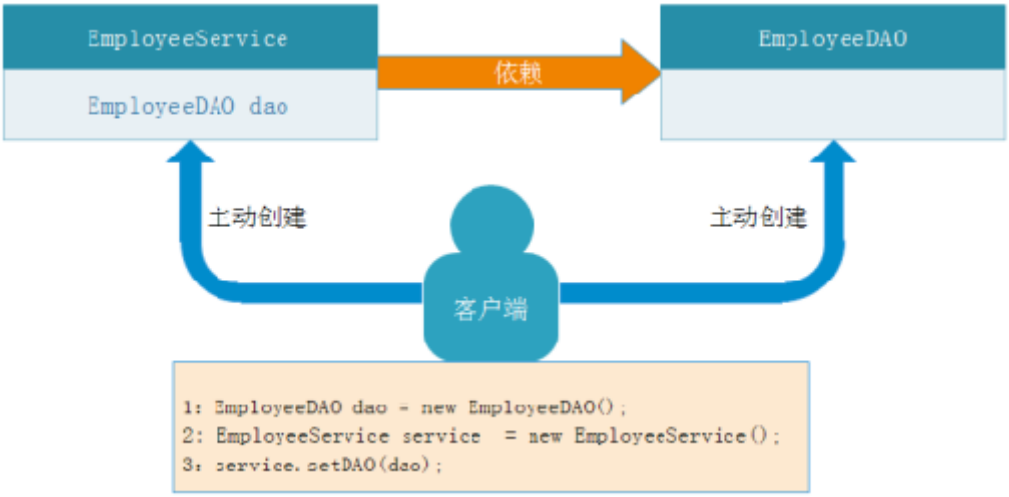
三、IoC 和 DI 思想

1、IoC

IoC：Inversion of Control（控制反转）：读作“反转控制”，更好理解，不是什么技术，而是一种设计思想，好比于 MVC。
就是将原本在程序中手动创建对象的控制权，交由 IoC 容器来管理。

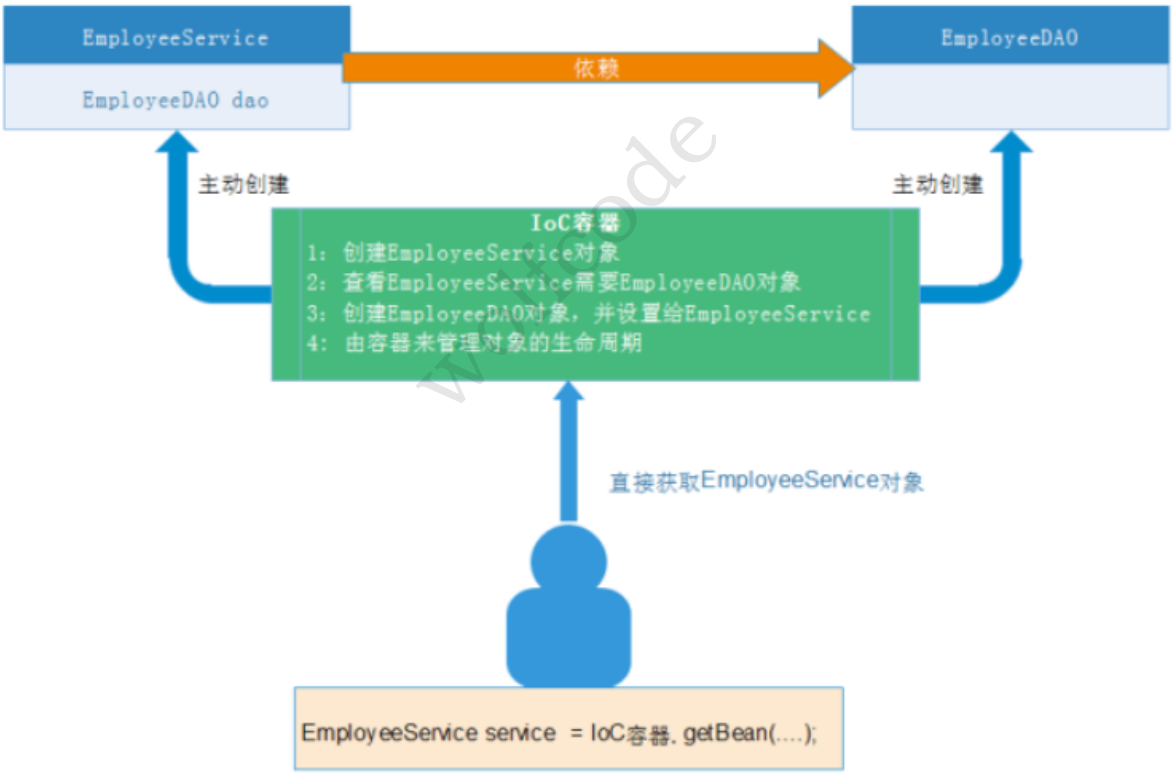
1.1、没用 IoC

若调用者需要使用某个对象，其自身就得负责该对象及该对象所依赖对象的创建和组装。

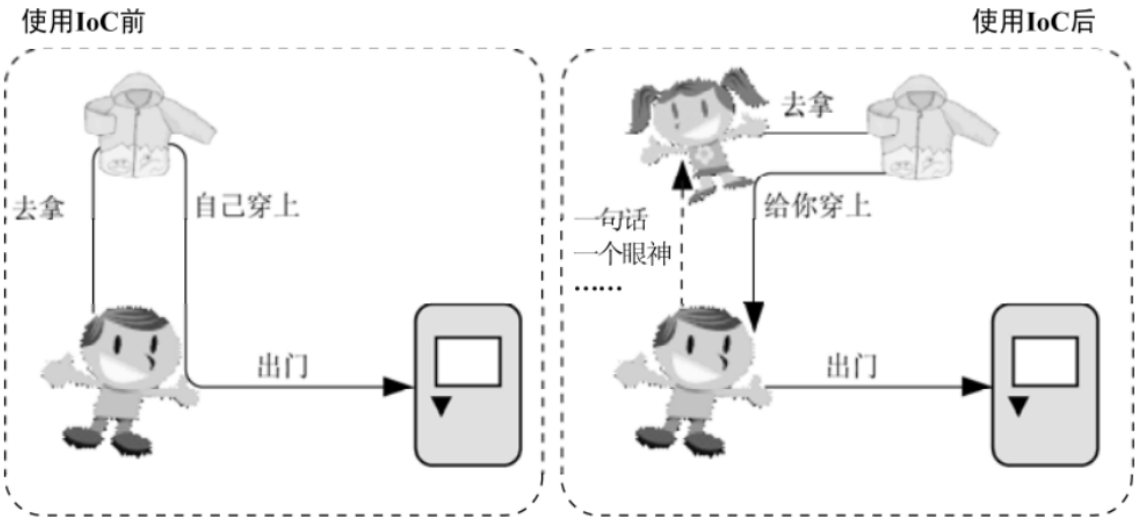


1.2、用 IoC

调用者只管负责从IoC 容器中获取需要使用的对象，不关心对象的创建过程，也不关心该对象依赖对象的创建以及依赖关系的组装，也就是把创建对象的控制权反转给了 IoC 容器。



1.3、对比



使用IoC前后的差别

2、DI

DI（Dependency Injection）：依赖注入。

IoC 从字面意思上很难体现出谁来维护对象之间的关系，Martin Fowler 提出一个新的概念 DI，更明确描述了"被注入对象（Service 对象）依赖 IoC 容器来配置依赖对象"。

所以两者配合在一起时：

- IoC：指将对象的创建权，反转给了IoC 容器；
- DI：指 IoC 容器创建对象的过程中，将对象依赖属性（常量，对象）通过配置设值给该对象。

四、Spring 的 HelloWorld 程序

1、需求

创建对象，设置对象属性，并调用对象的方法。

2、新建 Maven 项目

设置编译版本及添加依赖：

```
<properties>
    <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
    <maven.compiler.source>11</maven.compiler.source>
    <maven.compiler.target>11</maven.compiler.target>
</properties>

<dependencies>
    <dependency>
        <groupId>junit</groupId>
        <artifactId>junit</artifactId>
        <version>4.12</version>
        <scope>test</scope>
    </dependency>
</dependencies>
```

3、使用非 IoC 方式

3.1、编写类

```
package cn.wolfcode._01_hello;

public class Person {
    private String name;
    public String getName() {
        return name;
    }
    public void setName(String name) {
        this.name = name;
    }
    public void doWork() {
        System.out.println(this.name + "工作");
    }
}
```

3.2、编写单元测试类

```
public class PersonTest {
    @Test
    public void testNotIoc() {
        Person person = new Person();
        person.setName("小罗");
        person.doWork();
    }
}
```


4、使用 Spring IoC 方式

4.1、添加依赖

```
<dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-context</artifactId>
    <version>5.0.8.RELEASE</version>
</dependency>
```

4.2、通过配置告诉 Spring 来管理对象

在配置告诉 Spring 管理哪个类的对象，想想之前学 JavaWeb 的时候，怎么配置 Servlet 的，和其配置有些类似，都是使用 XML 配置。在 resources 目录新建 Spring 的配置文件 01.hello.xml，配置如下：

```
<!-- 告诉 Spring 帮我们创建什么对象，设置什么属性值 -->
<bean id="person" class="cn.wolfcode._01_hello.Person"> <!-- IoC -->
    <!--
        Person p = new Person();
        p.setName("红旗")
    -->
    <property name="name" value="大罗"/> <!-- DI -->
</bean>
```

4.3、启动容器获取对象

```
public class PersonTest {
    @Test
    public void testIoc() {
        // 解析配置启动容器
        @SuppressWarnings("resource")
        ApplicationContext ctx = new
ClassPathXmlApplicationContext("classpath:01.hello.xml");
        // 从容器中根据名称获取对象
        Person person = (Person) ctx.getBean("person");
        person.doWork();
    }

    // ...
}
```


五、Spring 的 HelloWorld 程序分析

1、ApplicationContext 及 bean

- BeanFactory：Spring 最底层的接口，只提供了的 IoC 功能，负责创建、组装、管理 bean，在应用中；
- ApplicationContext：继承了BeanFactory，除此之外还提供 AOP 集成、国际化处理、事件传播、统一资源加载等功能。

一般不使用 BeanFactory，而推荐使用 ApplicationContext（应用上下文）

被 Spring IoC 容器管理的对象称之为 bean。

2、Spring 配置方式

Spring IoC 容器启动时通过读取配置文件中的配置元数据，通过元数据对应用中的各个对象进行实例化及装配。元数据的配置有三种方式：

- XML-based configuration
- Annotation-based configuration
- Java-based configuration

3、Spring IoC 管理 bean 的原理（不讲）

- 通过 Resource 对象加载配置文件；
- 解析配置文件，解析 bean 元素，id 作为 bean 的名字，class 用于反射得到 bean 的实例，注意：此时，bean 类必须存在一个无参数构造器（和访问权限无关）；

- 调用 getBean 方法的时候，从容器中返回对象实例。

```
@Test
public void testMockSpring() throws Exception {
    // 假设已从配置文件中解析到如下数据
    String className = "cn.wolfcode._01_hello.Person";
    String objName = "person";
    String propertyName = "name";
    String propertyValue = "C罗";

    // 反射获取字节码对象
    Class<?> clzz = Class.forName(className);
    // 反射创建对象，对应类必须存在公共无参的构造函数
    Object obj = clzz.newInstance();

    BeanInfo beanInfo = Introspector.getBeanInfo(clzz, Object.class);
    for(PropertyDescriptor pd : beanInfo.getPropertyDescriptors()) {
        // 判断属性名与解析 XML 中属性是否一致
        if(pd.getName().equals(propertyName)) {
            // 若一致，调用对应 setter 方法设置
            pd.getWriteMethod().invoke(obj, propertyValue);
        }
    }

    // 把对象存容器 省略

    ((Person) obj).doWork();
}
```


六、Spring XML 基本配置

1、XML 语法提示

只要添加了对应的约束头，写配置的时候就可以提示。而在 IDEA 中是通过手写一些提示出来的。

2、bean 元素中 id 和 name 属性

在 Spring 配置中，id 和 name 属性都可以定义 bean 元素的名称，不同的是：

- id 属性，遵守 XML 语法 ID 约束。必须以字母开始，可以使用字母、数字、连字符、下划线、句话、冒号，不能以“/”开头。
- name 属性，就可以使用很多特殊字符，比如在 Spring 和 Struts1，就得使用 name 属性来的定义 bean 的名称。当然也可以使用 name 属性为 bean 元素起多个别名，多个别名之间使用逗号或空格隔开，在代码中依然通过容器对象.getBean(...) 方法获取。

注意：从 Spring3.1 开始，id 属性不再是 ID 类型了，而是 String 类型，也就是说 id 属性也可以使用“/”开头了，而 **bean 元素名称的唯一性由容器负责检查。**

```
<bean name="/login" class="cn.wolfcode.hello.web.controller.LoginController" />
```

```
<bean name="hello,hi" class="cn.woldcode.hello.HelloWorld"/>
```

建议：**bean 起名尽量规范，不要搞一些非主流的名字，尽量使用 id。**

七、getBean 方法及 Spring 常见异常

1、按照名字

缺点：要强转，不太安全。

```
Person bean1 = (Person)ctx.getBean("person");
```

2、按照类型

要求在 Spring 中只配置一个这种类型的实例。缺点：可能找多个报错。

```
Person bean2 = ctx.getBean(Person.class);
```

3、按照名字和类型

```
Person bean3 = ctx.getBean("person3", Person.class);
```

4、常见异常

异常1：NoSuchBeanDefinitionException: No bean named 'person2' available

按照 bean 名称去获取 bean 时，不存在名称为 person2 的 bean。

异常2：NoUniqueBeanDefinitionException: No qualifying bean of type 'cn.wolfcode._01_hello.Person' available: expected single matching bean but found 2: person,person2

按照 cn.wolfcode._01_hello.Person 类型去获取 bean 时，期望找到该类型唯一的一个 bean，可是此时找到了两个。

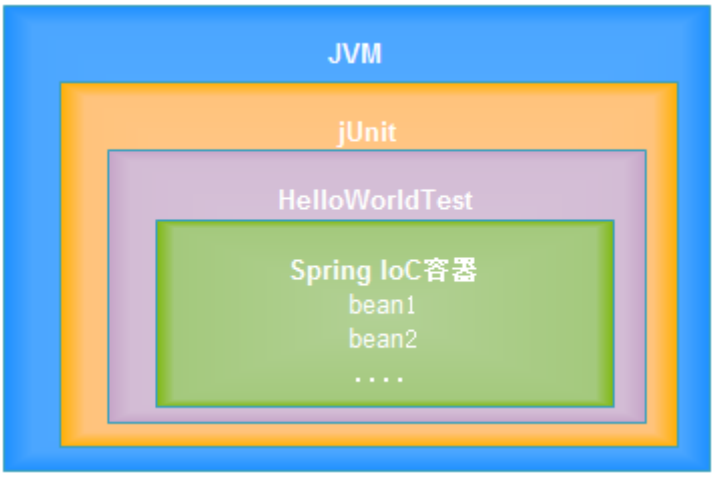
异常3：BeanDefinitionParsingException: Configuration problem: Bean name 'person' is already used in this element Offending resource: class path resource [01.hello.xml]

在 01.hello.xml 文件中，多个 bean 元素的名称是 person。

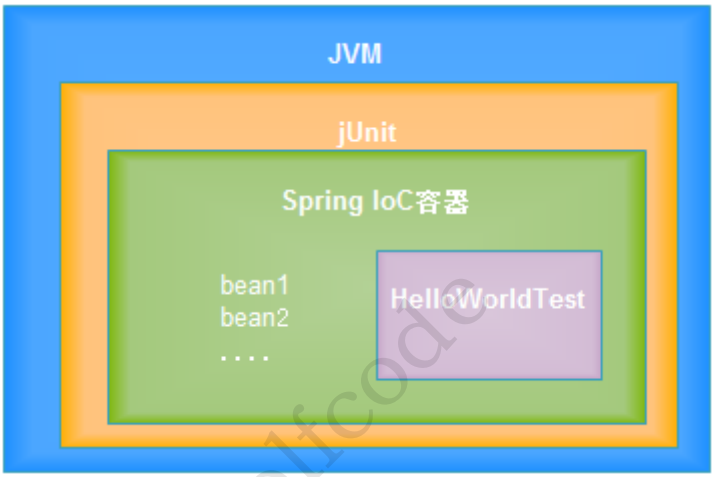
八、Spring 测试

1、传统测试存在的问题

每个测试都要重新启动 Spring 容器，启动容器的开销大，测试效率低下。不应该是测试代码管理 Spring 容器，应该是 Spring 容器在管理测试代码。



2、使用 Spring 测试



2.1、添加依赖

```
<dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-test</artifactId>
    <version>5.0.8.RELEASE</version>
    <scope>test</scope>
</dependency>
```

2.2、编写测试代码

```
@RunWith(SpringJUnit4ClassRunner.class) // 在测试方法之前启动容器
@ContextConfiguration("classpath:01.hello.xml") // 指定加载的配置文件
public class PersonTest {
    @Autowired // 在测试方法运行之前，获取容器对象 person = ctx.getBean(Person.class)
    private Person person;

    @Test
    public void testIoC() {
        person.doWork();
    }
}
```

若把 @ContextConfiguration("classpath:01.hello.xml") 写成 @ContextConfiguration，则去测试类的路径找测试类名-context.xml 配置文件，如：PersonTest-context.xml。

九、IoC 概述

将原本在程序中手动创建对象的控制权，交由 IoC 容器来管理，在 Spring 中被管理的对象称之为 bean。接下我们来看下 Spring 创建对象的时机、作用域等。

十、bean 创建时机

给类添加构造器：

```
package cn.wolfcode._01_hello;

public class Person {
    private String name;

    public Person() {
        System.out.println("对象创建了");
    }
    public String getName() {
        return name;
    }
    public void setName(String name) {
        this.name = name;
    }
    public void doWork() {
        System.out.println(this.name + "工作");
    }
}
```


运行单元测试，你会发现在启动 Spring 容器的时候就会创建所有的 bean。

十一、bean 实例方式

1、构造器实例化

一般使用无参构造器，最标准，也使用最多。

1.1、编写类

```
package cn.wolfcode._02_ioc;

public class Cat1 {
    public Cat1() {
        System.out.println("Cat1 对象被创建了");
    }
}
```

1.2、编写配置文件

在 resources 目录下新建 02.ioc.xml，配置如下：

```
<bean id="cat1" class="cn.wolfcode._02_ioc.Cat1"/>
```

1.3、编写测试类

```
@RunWith(SpringJUnit4ClassRunner.class)
@ContextConfiguration("classpath:02.ioc.xml")
public class Cat1Test {
    @Autowired
    private Cat1 cat1;

    @Test
    public void test() {
        System.out.println(cat1);
    }
}
```

2、实现 FactoryBean 接口实例化

2.1、编写类

```
package cn.wolfcode._02_ioc;
public class Cat2 {
    public Cat2() {
        System.out.println("Cat2 对象被创建了");
    }
}
```

```
package cn.wolfcode._02_ioc;

// 实列工厂类
public class Cat2FactoryBean implements FactoryBean<Cat2> {
    // 创建对象的方法，给 Spring 用，创建对象的时候使用
    @Override
    public Cat2 getObject() throws Exception {
        Cat2 cat2 = new Cat2();
        return cat2;
    }
    // 判断类型用的方法，给 Spring 用
    @Override
    public Class<?> getObjectType() {
        return Cat2.class;
    }
}
```


2.2、编写配置文件

在 02.ioc.xml，配置如下：

```
<bean id="cat2" class="cn.wolfcode._02_ioc.Cat2FactoryBean"/>
```

2.3、编写测试类

```java
@RunWith(SpringJUnit4ClassRunner.class)
@ContextConfiguration("classpath:02.ioc.xml")
public class Cat2Test {
    @Autowired
    private Cat2 cat2;
    @Test
    public void test() {
        System.out.println(cat2);
    }
}
```

2.4、配置解释及应用

因为 Spring 在对个 bean 元素解析创建对象的时候，还是先创建这个工厂对象；但发现这个类实现了 FactoryBean 这接口，就会再调用工厂对象的 getObject() 方法创建 bean 对象，把这个 bean 对象存在容器中。总结一句配置类发现类名是以 FactoryBean 结尾的话，注意到底是创建什么类型的对象存在容器中。

这种方式与其他框架整合的时候用的比较多，如集成 MyBatis 框架使用，就会配置 org.mybatis.spring.SqlSessionFactoryBean。也可以是工厂对象注入属性，为后面整合框架作铺垫。

十二、bean 作用域

1、作用域

在 Spring 容器中是指其创建的 bean 对象相对于其他 bean 对象的请求可见范围。

2、分类

- singleton：单例，在 Spring IoC 容器中仅存在一个 bean 实例 （缺省默认的 scope）。
- prototype：多例，每次从容器中调用 Bean 时，都返回一个新的实例，即每次调用 getBean() 时，相当于执行 new XxxBean()：不会在容器启动时创建对象。
- request：用于 web 开发，将 Bean 放入 request 范围，request.setAttribute("xxx")， 在同一个 request 获得同一个 Bean。
- session：用于 web 开发，将 Bean 放入 Session 范围，在同一个 Session 获得同一个 Bean。
- application：Scopes a single bean definition to the lifecycle of a ServletContext. Only valid in the context of a web-aware Spring ApplicationContext。
- websocket：Scopes a single bean definition to the lifecycle of a WebSocket. Only valid in the context of a web-aware Spring ApplicationContext。

3、配置

```xml
<bean id="" class="" scope="作用域"/>
```

默认不配置就是 singleton，注意当 scope 是 prototype 时容器启动不会创建该 bean。

4、使用总结

在开发中主要使用 scope="singleton"。对于 Struts1 的 Action 使用 request，Struts2 中的 Action 使用 prototype 类型，其他**使用 singleton，即不配置。**

十三、bean 初始化和销毁

1、问题

比如 DataSource，SqlSessionFactory 最终都需要关闭资源，以前用完（回收销毁前）需自己手动关闭资源，即都要调用 close 方法。

在使用 Spring 之后，这些对象等都会被 Spring IoC 容器管理的，但这些 bean 使用完之后也需要释放资源的，但现在这些可交由 Spring 来完成。

2、正常关闭容器

调用容器的销毁方法 close()，才是正常关闭容器。有以下几种操作方式：

- 使用 Spring Test 会自动关闭容器
- 手动关闭容器 ctx.close()（注意需类型强转）
- 使用 Java7 自动关闭资源
- 使用 Lombok 的注解 @CleanUp 贴要关闭的资源上
- 把 Spring 线程作为 JVM 的子线程：ctx.registerShutdownHook()

3、没使用 Spring

3.1、编写类

```
package cn.wolfcode._02_ioc;

// 模拟的数据库连接池
public class MyDataSource {
    public MyDataSource() {
        System.out.println("对象创建");
    }
    public void getConnection() {
        System.out.println("拿到连接");
    }
    public void init() {
        System.out.println("初始化池子");
    }
    public void close() {
        System.out.println("销毁池子");
    }
}
```


3.2、编写单元测试类

```
public class MyDataSourceTest {
    @Test
    public void testOld() {
        MyDataSource dataSource = new MyDataSource();
        dataSource.init();
        dataSource.getConnection();
        dataSource.close();
    }
}
```

4、使用 Spring

4.1、编写配置文件

在 02.ioc.xml，配置如下：

```
<bean id="myDataSource" class="cn.wolfcode._02_ioc.MyDataSource"
    init-method="init" destroy-method="close" scope="prototype"/>
```

4.2、修改单元测试类

```
@RunWith(SpringJUnit4ClassRunner.class)
@ContextConfiguration("classpath:02.ioc.xml")
public class MyDataSourceTest {
    @Autowired
    private MyDataSource dataSource;

    @Test
    public void testNew() {
        dataSource.getConnection();
    }
}
```


注意：

- 只有正常关闭容器，才会执行 bean 的配置的 destroy-method 方法。
- bean 的 scope 为 prototype 的，那么容器只负责创建和初始化，它并不会被 Spring 容器管理（不需要存起来），交给用户自己处理。即 bean 作用域为 prototype 的即使配置了 destroy-method 也不会被调用。

十四、DI 概述

1、定义

指 Spring 创建对象的过程中，将对象依赖属性通过配置设值给该对象。

2、注入方式

- setter 注入（或叫属性注入），**其类必须提供对应 setter 方法。**
- 构造器注入。

```java
public class User {
    private String name;

    public User() {}
    public User(String name) {
        this.name = name;
    }

    public void setName(String name) {
        this.name = name;
    }
}
```

```java
public class UserTest {
    @Test
    public void test() {
        User u1 = new User();
        u1.setName("郭嘉"); // 属性注入

        User u2 = new User("荀彧"); // 构造器注入
    }
}
```

3、注入值

常见注入值由常量、bean 等。

十五、注入常量值

给对象注入的值的类型为八大基本数据类型及其包装类，String，BigDecimal 等等。

1、XML 配置语法

```xml
<property name="对象属性名称"  value="需要注入的值"/>
```

2、代码示例

2.1、编写类

```java
package cn.wolfcode._03_di;

public class Employee {
    private String name;
    private int age;
    private BigDecimal salary;
    public String getName() {
        return name;
    }
    public void setName(String name) {
        this.name = name;
    }
    public int getAge() {
```

```
        return age;
    }
    public void setAge(int age) {
        this.age = age;
    }
    public BigDecimal getSalary() {
        return salary;
    }
    public void setSalary(BigDecimal salary) {
        this.salary = salary;
    }
    @Override
    public String toString() {
        return "Employee [name=" + name + ", age=" + age + ", salary=" + salary
+ "]";
    }
}
```

2.2、编写配置文件

在 resources 目录下新建 03.di.xml，配置如下：

```
<bean id="employee" class="cn.wolfcode._03_di.Employee">
    <!-- name 写属性名  value 是写属性值 -->
    <property name="name" value="罗老师"/>
    <property name="age" value="12"/>
    <property name="salary" value="1000000"></property>
</bean>
```

2.3、编写单元测试类

```
@RunWith(SpringJUnit4ClassRunner.class)
@ContextConfiguration("classpath:03.di.xml")
public class EmployeeTest {
    @Autowired
    private Employee employee;

    @Test
    public void test() {
        System.out.println(employee);
    }
}
```

注意：Spring 帮我们做类型转换，但前提是能转换，不能会报错的。

十六、注入 bean

给对象注入的值的是容器中的另外一个对象。

1、XML 配置语法

```
<property name="对象属性名称" ref="容器另外一个 bean 的 id 值"/>
```

2、代码示例

2.1、编写类

```
package cn.wolfcode._03_di;

public class EmployeeDao {
}
```

```
package cn.wolfcode._03_di;

public class EmployeeService {
    private EmployeeDao employeeDao;
    public void setEmployeeDao(EmployeeDao employeeDao) {
        this.employeeDao = employeeDao;
    }
    @Override
    public String toString() {
        return "EmployeeService [employeeDao=" + employeeDao + "]";
    }
}
```

2.2、编写配置文件

在 03.di.xml，配置如下：


```
<!-- 配置 EmployeeDao bean -->
<bean id="employeeDao" class="cn.wolfcode._03_di.EmployeeDao"/>

<!-- 配置 EmployeeService bean -->
<bean id="employeeService" class="cn.wolfcode._03_di.EmployeeService">
    <!--
        name 写的是属性名
        ref 注入 bean 的时候使用，写容器中另外一个 bean 的 id 的值
     -->
    <property name="employeeDao" ref="employeeDao"/>
</bean>
```

2.3、编写单元测试类

```
@RunWith(SpringJUnit4ClassRunner.class)
@ContextConfiguration("classpath:03.di.xml")
public class EmployeeServiceTest {
    @Autowired
    private EmployeeService employeeService;

    @Test
    public void test() {
        System.out.println(employeeService);
    }
}
```

十七、练习-使用 XML 配置模拟用户注册

1、需求

编写业务层和 DAO 层代码使用 JDBC 模拟用户注册，这些对象都被 Spring 管理（意思这类的对象都不要自己创建），并使用 Spring Test 进行测试。

2、新建 Maven 项目并设置编译版本及添加依赖

```
<properties>
    <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
    <maven.compiler.source>11</maven.compiler.source>
    <maven.compiler.target>11</maven.compiler.target>
</properties>

<dependencies>
    <dependency>
        <groupId>com.alibaba</groupId>
        <artifactId>druid</artifactId>
        <version>1.1.9</version>
    </dependency>
    <dependency>
        <groupId>mysql</groupId>
        <artifactId>mysql-connector-java</artifactId>
        <version>5.1.45</version>
        <scope>runtime</scope>
    </dependency>
    <dependency>
        <groupId>org.projectlombok</groupId>
        <artifactId>lombok</artifactId>
        <version>1.16.20</version>
        <scope>provided</scope>
    </dependency>
    <dependency>
        <groupId>org.springframework</groupId>
        <artifactId>spring-context</artifactId>
        <version>5.0.8.RELEASE</version>
    </dependency>
    <dependency>
        <groupId>org.springframework</groupId>
        <artifactId>spring-test</artifactId>
        <version>5.0.8.RELEASE</version>
        <scope>test</scope>
    </dependency>
    <dependency>
        <groupId>junit</groupId>
        <artifactId>junit</artifactId>
        <version>4.12</version>
        <scope>test</scope>
    </dependency>
</dependencies>
```


3、新建表

```
CREATE TABLE `student` (
  `id` bigint(20) NOT NULL AUTO_INCREMENT,
  `username` varchar(255) DEFAULT NULL,
  `password` varchar(255) DEFAULT NULL,
  PRIMARY KEY (`id`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8;
```

4、编写 db.properties

在 resources 目录新建 db.properties，配置如下：

```
jdbc.driverClassName=com.mysql.jdbc.Driver
jdbc.url=jdbc:mysql:///spring-demo
jdbc.username=root
jdbc.password=admin
```

5、编写 DAO 层代码

```
package cn.wolfcode.dao;

public interface IStudentDAO {
    void save(String username, String password) throws Exception;
}
```

```
package cn.wolfcode.dao.impl;

public class StudentDAOImpl implements IStudentDAO {
    private DataSource dataSource;
    public void setDataSource(DataSource dataSource) {
        this.dataSource = dataSource;
    }

    @Override
    public void save(String username, String password) throws Exception {
        @Cleanup
        Connection connection = dataSource.getConnection();
        @Cleanup
        PreparedStatement ps = connection.prepareStatement("INSERT INTO
student(username, password) VALUES(?, ?)");
        ps.setString(1, username);
        ps.setString(2, password);
        ps.executeUpdate();

    }
}
```

6、编写业务层代码

```java
package cn.wolfcode.service;

public interface IStudentService {
    void register(String username, String password) throws Exception;
}
```

```java
package cn.wolfcode.service.impl;

public class StudentServiceImpl implements IStudentService {
    private IStudentDAO studentDAO;
    public void setStudentDAO(IStudentDAO studentDAO) {
        this.studentDAO = studentDAO;
    }

    @Override
    public void register(String username, String password) throws Exception {
        studentDAO.save(username, password);
    }
}
```

7、编写 Spring 配置文件

在 resources 目录新建 applicationContext.xml，配置如下

```xml
<!-- 引入 db.properties -->
<context:property-placeholder location="classpath:db.properties"/>

<!-- 配置 DataSource bean -->
<bean id="dataSource" class="com.alibaba.druid.pool.DruidDataSource"
    init-method="init" destroy-method="close">
    <property name="driverClassName" value="${jdbc.driverClassName}"/>
    <property name="url" value="${jdbc.url}"/>
    <property name="username" value="${jdbc.username}"/>
    <property name="password" value="${jdbc.password}"/>
</bean>

<!-- 配置 StudentDAO bean -->
<bean id="studentDAO" class="cn.wolfcode.dao.impl.StudentDAOImpl">
    <!-- 前面属性名， 后面容器中另外一个 bean id 值 -->
    <property name="dataSource" ref="dataSource"/>
</bean>

<!-- 配置 StudentService bean -->
<bean id="studentService" class="cn.wolfcode.service.impl.StudentServiceImpl">
    <!-- 前面属性名， 后面容器中另外一个 bean id 值 -->
    <property name="studentDAO" ref="studentDAO"/>
</bean>
```

8、编写单元测试类

```java
@RunWith(SpringJUnit4ClassRunner.class)
@ContextConfiguration("classpath:applicationContext.xml")
public class StudentServiceTest {
    @Autowired
    private IStudentService studentService;

    @Test
    public void testRegister() throws Exception {
        studentService.register("罗老师", "666666");
    }
}
```