

尚马教育 JAVA 课程

Spring 容器

文档编号：C03

创建日期：2017-07-07

最后修改日期：2019-10-18

版本号：V3.5

电子版文件名：尚马教育-第三阶段-3.spring 容器.docx

文档修改记录：

更新日期	更新作者	更新说明	版本号
2017-07-19	张元林	初始版本	V1.0
2018-08-08	王绍成	版本更新	V2.0
2019-10-17	冯勇涛	课件格式以及课程深度加深	V3.0
2021-01-20	冯勇涛	调整课程顺序，优化课程	V3.5

目录

尚马教育 JAVA 课程	1
Spring 容器	1
1. spring 介绍	4
1.1. 认识 spring	4
1.2. Spring 的好处	4
1.3. Spring 核心服务	4
1.3.1. IOC	5
1.3.2. DI	5
1.3.3. AOP	6
2. Spring 容器对象结构图	9
3. Spring 快速入门 1	9
3.1. 版本选择	10
3.2. 下载 jar 包	10
3.3. 编写代码并添加 xml 配置文件	10
3.4. 初始化 spring 容器对象	11
3.5. 注意点	12
4. Spring 快速入门 2	12
4.1. 添加 mysql 驱动与 Druid 连接池包	12
4.2. 修改 xml 配置文件配置 DataSource	13
4.3. 初始化 spring 容器对象	13
4.4. 注意点	13
5. bean 标签属性详解	14
6. IOC 控制反转	15
6.1. Bean 标签配置	15
6.2. 工厂实例化 bean	15
6.2.1. 静态工厂实例化 bean	15
6.2.2. 实例工厂实例化 bean	16
6.3. 包扫描	17
7. DI 依赖注入	18

7.1. Set 注入.....	18
7.2. 构造器注入.....	19
7.3. 集合注入.....	20
7.4. 内部 bean 注入.....	21
7.5. 自动装配.....	21
8. 类中常用注解.....	22
9. Xml 中常用标签	22
10. Spring 测试环境搭建	23
10.1. 环境准备.....	23
10.2. 编写测试类.....	23

知识点:

知识点 1: 认识 IOC 控制反转思想, 并熟练使用 spring 的 IOC 配置

知识点 2: 认识 DI 依赖注入思想, 熟练使用。

1. spring 介绍

1.1. 认识 spring

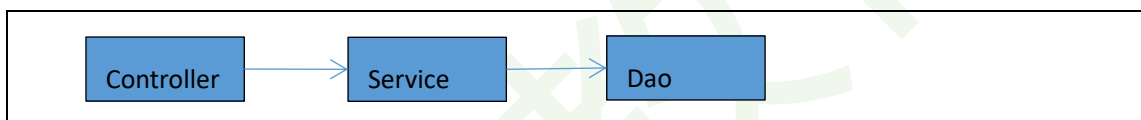
Spring 是一个开源的控制反转(Inversion of Control ,IoC)和面向切面(AOP)的容器框架。帮助开发人员分离组件之间的依赖关系，它的主要目的是简化企业开发。

Spring 利用了一些现有的技术，如 ORM 框架、日志框架、Quartz，线程池，任务 等。

1.2. Spring 的好处

在项目中引入 spring 立即可以带来下面的好处：

- 降低组件之间的耦合度,实现软件各层之间的解耦。



- 可以使用容器提供的众多服务，如：事务管理服务、消息服务等等。当我们使用容器管理事务时，开发人员就不再需要手工控制事务。
- 容器提供单例模式支持，开发人员不再需要自己编写实现代码。
- 容器提供了 AOP 技术，利用它很容易实现如权限拦截、运行期监控等功能。
- 容器提供的众多辅助类，使用这些类能够加快应用的开发，如：JdbcTemplate、RedisTemplate、RestTemplate 等。
- Spring 对于主流的应用框架提供了集成支持，如：集成 Hibernate 、Struts 、JPA 等，这样更便于应用的开发。

1.3. Spring 核心服务

核心服务提供 Spring 框架的基本功能 Bean Container。Spring 以 bean 的方式组织和管理 Java 应用中的各个组件及其关系, Spring 使用 [BeanFactory](#) 来产生和管理 Bean，它是工厂模式的实现。BeanFactory 使用控制反转(IoC)模式将应用的配置和依赖性规范与实际的应用程序代码分开。

Spring AOP 模块直接将面向切面的编程思想集成到了 Spring 框架中。AOP 模块为基于 Spring 的应用程序中的对象提供了事务管理服务。通过使用 Spring AOP，很简单就可以将声明性

事务管理集成到应用程序中。

1.3.1. IOC

控制反转：

```
public class PersonServiceBean {

    private PersonDao personDao = new PersonDaoImpl();

    public void save(Person person){

        personDao.save(person);

    }

}
```

`PersonDaoImpl` 是在应用内部创建及维护的。所谓控制反转就是应用本身不负责依赖对象的创建及维护，依赖对象的创建及维护是由外部容器负责的。这样控制权就由应用转移到了外部容器，控制权的转移就是所谓反转。

1.3.2. DI

依赖注入理解：当我们把依赖对象交给外部容器负责创建，那么 `PersonServiceBean` 类可以改成如下：

```
public class PersonServiceBean {

    private PersonDao personDao ;

    //通过构造器参数，让容器把创建好的依赖对象注入进 PersonServiceBean，当然也可以使用 setter 方法进行注入。

    public PersonServiceBean(PersonDao personDao){

        this.personDao=personDao;

    }

    public void save(Person person){

        personDao.save(person);

    }

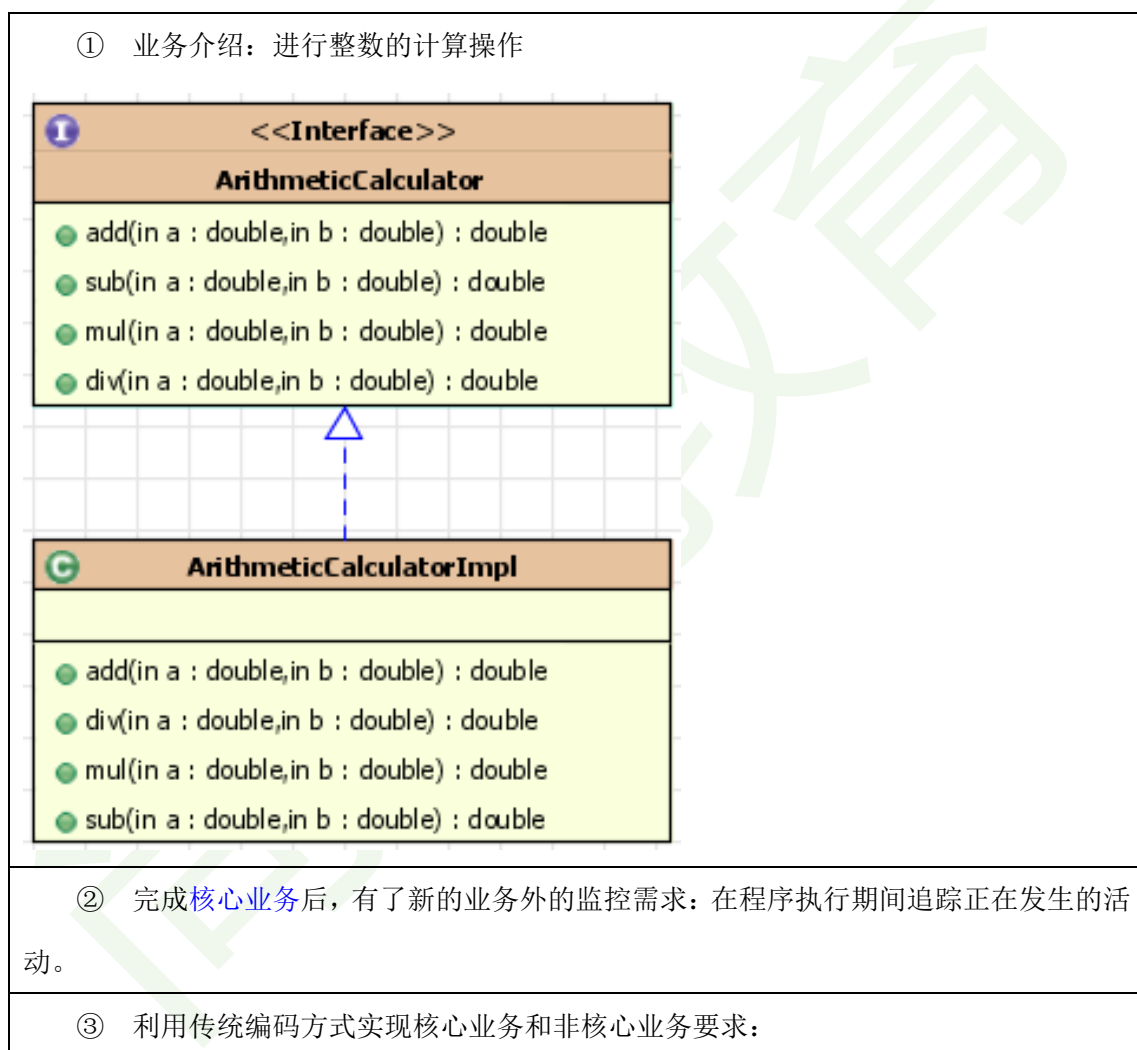
}
```

所谓依赖注入就是指：在运行期间，由外部容器动态地将依赖对象注入到组件中。

1.3.3. AOP

AOP 面向切面编程：在运行期间通过动态代理实现程序功能统一维护的一种技术，AOP 是 OOP 的延续，是软件开发中的一个热点，也是 Spring 框架中的一个重要内容，利用 AOP 可以对业务逻辑的各个部分进行隔离，从而使得业务逻辑各部分之间的耦合度降低，提高程序的可重用性，同时提高了开发的效率。

以一个实际业务开发中非常常见的应用示例，日志：



```
public class ArithmeticCalculatorImpl implements ArithmeticCalculator {

    @Override
    public void add(int i, int j) {

        System.out.println("日志:The method add begins with [" +
            i + ", " + j + "]" );

        int result = i + j;
        System.out.println("result: " + result);

        System.out.println("日志:The method add ends with " + result);
    }

    @Override
    public void sub(int i, int j) {

        System.out.println("日志:The method sub begins with [" +
            i + ", " + j + "]" );

        int result = i - j;
        System.out.println("result: " + result);

        System.out.println("日志:The method sub ends with " + result);
    }
}
```

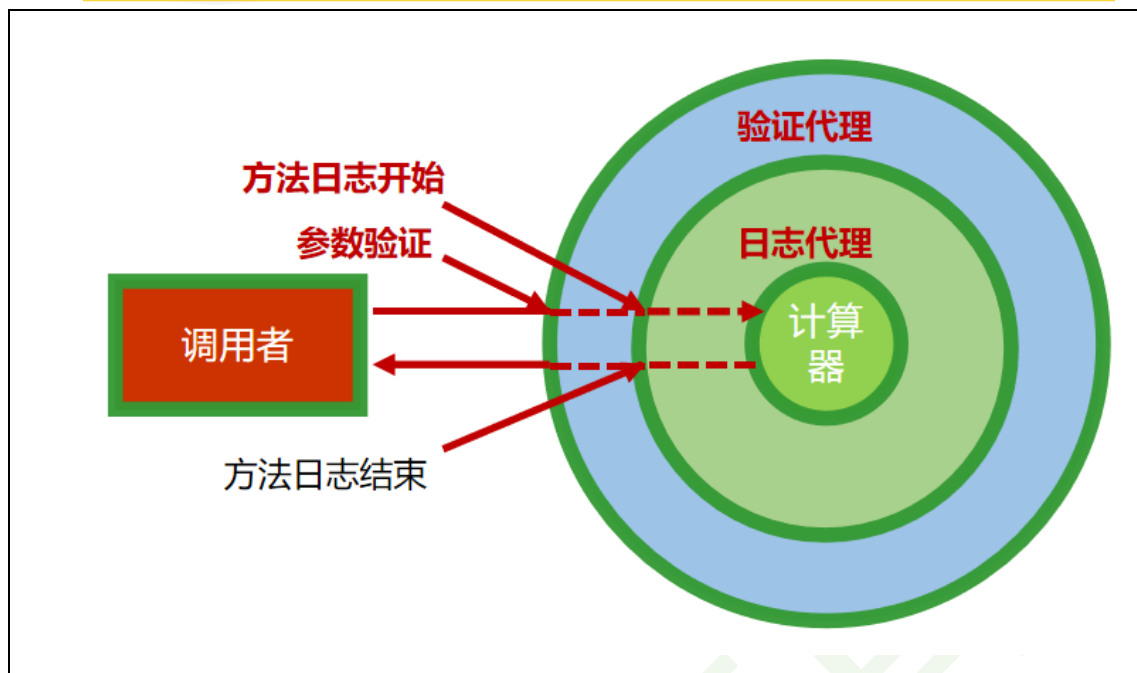
缺点:

代码混乱: 越来越多的非业务需求(日志,验证等)代码的添加,使原有的业务方法急剧膨胀. 每个方法在处理核心逻辑的同时还必须兼顾其他辅助性的关注点。

代码分散: 以日志需求为例, 只是为了满足这个单一需求, 就需要在多个方法里多次重复相同的日志代码. 如果日志需求发生变化, 必须修改所有模块。

④ 使用 AOP 思路实现:

使用 AOP 利用一个代理将原业务对象包装起来, 然后用该代理对象取代原始对象, 将日志处理的代码织入到原始对象的方法执行代码中:



2. Spring 容器对象结构图





3. Spring 快速入门 1

1. 选择合适的 spring 版本
2. 下载对应版本的 jar 包,并导入核心包到工程
3. 创建 spring.xml 配置文件
4. 实例化 spring 容器对象,测试 bean 的获取

3.1. 版本选择

Spring Framework

5.3.3

OVERVIEW

LEARN

Documentation

Each **Spring project** has its own; it explains in great details how you can use **project features** and what you can achieve with them.

5.3.3	CURRENT GA 正式版	Reference Doc.	API Doc.
5.3.4-SNAPSHOT	SNAPSHOT 测试版	Reference Doc.	API Doc.
5.2.13.BUILD-SNAPSHOT	GA	Reference Doc.	API Doc.
5.2.12.RELEASE	GA	Reference Doc.	API Doc.

带 GA 表示正式发布版本,说明是 release 版本。习惯选用非最新正式版。






可选择 5.2 版本。

3.2. 下载 jar 包

Spring 官网: <https://spring.io/>

下载地址: <http://repo.spring.io/release/org/springframework/spring/>

当前入门案例需要 4 个 spring 的核心 jar 包, 以及依赖的 common-logging.jar。

 commons-logging-1.1.1.jar
  spring-beans-5.2.8.RELEASE.jar
  spring-context-5.2.8.RELEASE.jar
  spring-core-5.2.8.RELEASE.jar
  spring-expression-5.2.8.RELEASE.jar

3.3. 编写代码并添加 xml 配置文件

```
public class SysuserServiceImpl implements ISysuserService {

    public SysuserServiceImpl() {

        System.out.print("SysuserServiceImpl constructor");

    }

    @Override
```

```
public Sysuser login(String uname, String upwd) {

    System.out.println("login: " + uname + "--" + upwd);

    return null;

}

}
```

```
<?xml version="1.0" encoding="UTF-8"?>

<beans xmlns="http://www.springframework.org/schema/beans"

    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

    xsi:schemaLocation="http://www.springframework.org/schema/beans

http://www.springframework.org/schema/beans/spring-beans.xsd">

    <bean id="sysuserService" class="com.javasm.sys.service.SysuserServiceImpl">

    </bean>

</beans>
```

注意点:

注意点 1: 该文件必须有 beans 根标签, 并且 beans 标签中必须引入 beans.xsd 模板定义文件;

注意点 2: bean 标签的 class 不能配置接口类, 必须是一个可实例化的类。

注意点 3: bean 标签的 id 必须唯一, 不可重复。

注意点 4: xml 配置文件放在 src 类路径下, 名称随意。此处文件名 spring.xml

3.4. 初始化 spring 容器对象

```
@Test

public void test2_springUserService() {

    ApplicationContext ac = new ClassPathXmlApplicationContext("spring.xml");

    ISysuserService userService1 = (ISysuserService) ac.getBean("userService");

    ISysuserService userService = ac.getBean(ISysuserService.class);

    System.out.println(userService);

    System.out.println(userService1);

    System.out.println(userService==userService1);

}
```

```
userService.login("fyt", "fyt");
}
```

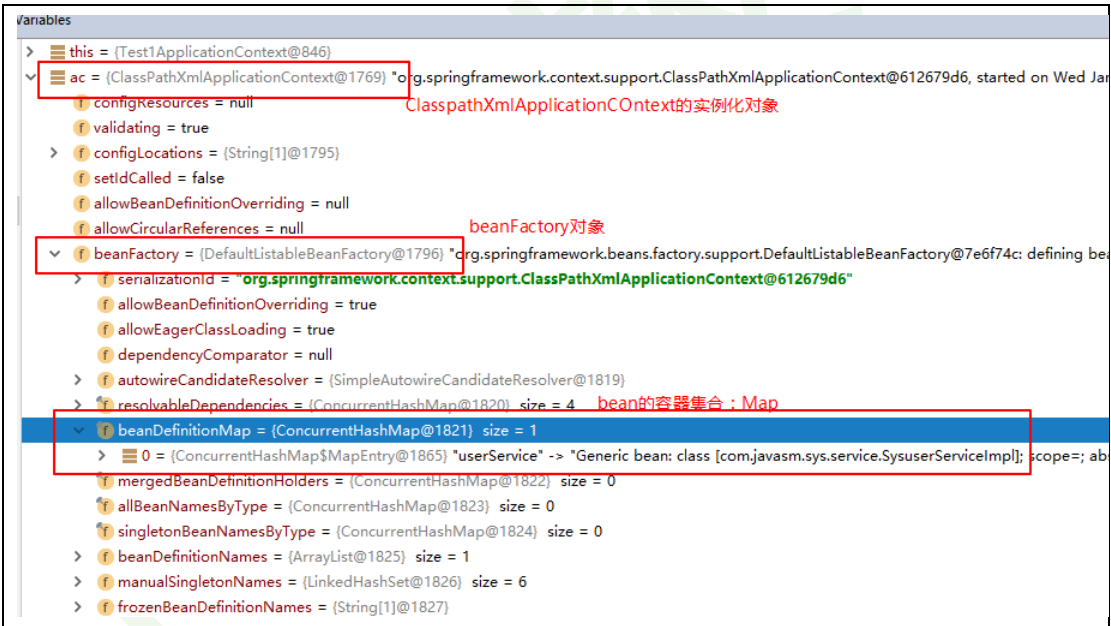
3.5. 注意点

注意点 1: 当前加载类路径下 xml 配置, 使用 ClassPathXmlApplicationContext 创建 spring 容器。

注意点 2: ApplicationContext 接口中的三个 getBean 重载方法, 两个按照 bean 的 id 获取对象, 1 个按照类型获取对象。

注意点 3: 三个 getBean 方法得到的对象, 是同一个对象, 即 spring 管理的 bean 对象默认单例状态。

注意点 3: 当执行 new ClassPathXmlApplicationContext 对象时, SysuserServiceImpl 对象的构造器会立即执行。在该行代码处打断点, 查看对象内数据:



Variables window showing the state of the `ClassPathXmlApplicationContext` instance:

- `this`: (Test1ApplicationContext@846)
- `ac`: (ClassPathXmlApplicationContext@1769) "org.springframework.context.support.ClassPathXmlApplicationContext@612679d6, started on Wed Jan 10, 2018 at 10:10:10 AM"
 - `configResources`: null
 - `validating`: true
 - `configLocations`: (String[1]@1795)
 - `setIdCalled`: false
 - `allowBeanDefinitionOverriding`: null
 - `allowCircularReferences`: null
 - `beanFactory`: (DefaultListableBeanFactory@1796) "org.springframework.beans.factory.support.DefaultListableBeanFactory@7e6f74c: defining beans"
 - `allowBeanDefinitionOverriding`: true
 - `allowEagerClassLoading`: true
 - `dependencyComparator`: null
 - `autowireCandidateResolver`: (SimpleAutowireCandidateResolver@1819)
 - `resolvableDependencies`: (ConcurrentHashMap@1820) size = 4
 - `beanDefinitionMap`: (ConcurrentHashMap@1821) size = 1
 - `0`: (ConcurrentHashMap\$MapEntry@1865) "userService" -> "Generic bean: class [com.javasm.sys.service.SysuserServiceImpl]; scope=; ab"
 - `mergedBeanDefinitionHolders`: (ConcurrentHashMap@1822) size = 0
 - `allBeanNamesByType`: (ConcurrentHashMap@1823) size = 0
 - `singletonBeanNamesByType`: (ConcurrentHashMap@1824) size = 0
 - `beanDefinitionNames`: (ArrayList@1825) size = 1
 - `manualSingletonNames`: (LinkedHashSet@1826) size = 6
 - `frozenBeanDefinitionNames`: (String[1]@1827)

4. Spring 快速入门 2

4.1. 添加 mysql 驱动与 Druid 连接池包

```
> druid-1.1.10.jar
> mysql-connector-java-5.1.40-bin.jar
```

4.2. 修改 xml 配置文件配置 DataSource

```
<bean id="dataSource" class="com.alibaba.druid.pool.DruidDataSource" init-method="init" destroy-
method="close">

    <property name="url" value="jdbc:mysql://127.0.0.1:3306/704b"></property>

    <property name="driverClassName" value="com.mysql.jdbc.Driver"></property>

    <property name="username" value="root"></property>

    <property name="password" value="root"></property>

    <property name="initialSize" value="5"></property>

    <property name="minIdle" value="5"></property>

</bean>
```

4.3. 初始化 spring 容器对象

```
@Test

public void test3_springDataSource() {

    ApplicationContext ac = new ClassPathXmlApplicationContext("spring.xml");

    DataSource ds = ac.getBean(DataSource.class);

    System.out.println(ds);

    ((ClassPathXmlApplicationContext) ac).close();

}
```

4.4. 注意点

注意点 1: bean 标签的 init-method 属性用来指定 bean 实例化后立即执行的初始化方法; destroy-method 属性指定 spring 容器销毁后执行的释放资源方法。

注意点 2: 只有显示执行了 ApplicationContext 的 close 方法才能够触发 destroy-method。

注意点 3: property 标签的 name 属性值是 DruidDataSource 类中的成员变量名, value 是赋值, 等价于调用了 set 方法。配置 bean 标签与如下代码效果类似:

```
@Test
public void test3_dataSource() throws SQLException {
    DruidDataSource ds = new DruidDataSource();
    ds.setUrl("jdbc:mysql://127.0.0.1:3306/704b");
    ds.setDriverClassName("com.mysql.jdbc.Driver");
    ds.setUsername("root");
    ds.setPassword("root");
    ds.setInitialSize(5);
    ds.setMinIdle(5);
    try {
        ds.init();
    } catch (SQLException e) {
        e.printStackTrace();
    }
    System.out.println(ds.getConnection());
    ds.close();
}
```

5. bean 标签属性详解

Id 属性:注册 bean 对象在 spring 容器中的标示名称。

Scope 属性:bean 的作用域:

```
<bean id="userService" class="com.javasm.sys.service.SysuserServiceImpl"
scope="prototype"></bean>
```

.singleton

在每个 Spring IoC 容器中一个 bean 定义只有一个对象实例。

.prototype

每次从容器获取 bean 都是新的对象。<bean id= “ ” class= “ ” scope= “prototype” />

Init-method 与 Destory-method 属性:bean 的初始化和销毁方法

```
<bean id="dataSource" class="com.alibaba.druid.pool.DruidDataSource" init-method="init"
destroy-method="close">
```

Lazy-init 属性: bean 对象延迟初始化

```
<bean id="userService" class="com.javasm.sys.service.SysuserServiceImpl"
lazy-init="true"></bean>
```

6. IOC 控制反转

IOC: 控制反转, 应用本身不负责依赖对象的创建及维护, 依赖对象的创建及维护是由外部容器负责的。这样控制权就由应用转移到了外部容器, 控制权的转移就是所谓反转。

6.1. Bean 标签配置

入门案例中即在 xml 中配置 bean 标签注册 bean 对象到 spring 容器。一般 jar 包中类需要注册进 spring 容器使用 bean 标签配置。

```
<bean id="userService"
class="com.javasm.sys.service.SysuserServiceImpl"></bean>
```

6.2. 工厂实例化 bean

构建对象的过程比较复杂的情况下使用工厂实例化 bean, 比如 sqlSessionSessionFactory 对象的构建, 如果需要把 mybatis 的 sqlSessionSessionFactory 对象交给 spring 容器, 可使用工厂实例化 bean。

6.2.1. 静态工厂实例化 bean

```
public class MySqlSessionFactoryBean {
    //静态工厂
    public static SqlSessionFactory createSqlSessionFactory() {
        try {
            InputStream resourceAsStream =
Resources.getResourceAsStream("mybatis-config.xml");
```

```

        SqlSessionFactory f = new
        SqlSessionFactoryBuilder().build(resourceAsStream);

        return f;
    } catch (IOException e) {
        e.printStackTrace();
    }

    return null;
}
}

```

```

<bean id="factoryBean" class="com.javasm.factory.MySqlSessionFactoryBean"
factory-method="createSqlSessionFactory"></bean>

```

注意点：工厂方法是 `static` 修饰，表示静态工厂方法。

6.2.2. 实例工厂实例化 bean

```

public class MySqlSessionFactoryBean {

    //实例方法

    public SqlSessionFactory createSqlSessionFactory() {

        try {

            InputStream resourceAsStream = Resources.getResourceAsStream("mybatis-
            config.xml");

            SqlSessionFactory f = new SqlSessionFactoryBuilder().build(resourceAsStream);

            return f;

        } catch (IOException e) {

            e.printStackTrace();

        }

        return null;

    }

}

```



```

<!--先实例化工厂对象-->

<bean id="factoryBean" class="com.javasm.factory.MySqlSessionFactoryBean"></bean>

<!--再调用实例工厂对象的方法，把方法的返回值注册到容器中-->

<bean id="sqlSessionFactory" factory-bean="factoryBean" factory-
method="createSqlSessionFactory"></bean>

```

注意点 1: 工厂方法没有 `static` 修饰，必须先实例化出工厂对象，再调用工厂对象的实例方法。

注意点 2: `factory-bean` 属性值是工厂 bean 对象的 id 值，表示引用对象。

6.3. 包扫描

一般由开发人员自定义的类使用包扫描方式注册进 `spring` 容器。

```

<!-- 识别类上的@Controller, @Service, @Repository, @Component 等等注解。-->

<context:component-scan base-package="com.javasm"></context:component-scan>

```

`@Controller` 用于标注控制层组件；

`@Service` 用于标注业务层组件；

`@Repository` 用于标注数据访问组件，即 DAO 组件；

`@Component` 泛指组件，当组件不好归类的时候，我们可以使用这个注解进行标注；

```

@Component("userController")
public class SysuserController {}

```

```

@Service
public class SysuserServiceImpl implements ISysuserService {}

```

```

@Component
public class AsyncManager {}

```

注意点 1: `context:component-scan` 不仅仅识别这四个注解，后续学习其它。

注意点 2: 通过注解方式注册的 bean，默认 id 是类名首字母小写；可以自定义 id

注意点 3: 这四个注解当前是没有区别，效果都是注册 bean 到 `spring` 容器，后续 `springMVC` 框架有区别，我们需要有好的代码习惯，分别注解各层的 bean 对象。

注意点 4: 不能注解到接口类上。

7. DI 依赖注入

依赖注入：指组件之间依赖关系由容器在运行期决定，形象的说，即由容器动态的将某个依赖关系注入到组件之中。具体使用方式如下：

Set 注入：调用 bean 对象中的 set 方法注入依赖值。

构造器注入：实例化 bean 对象时，调用有参构造注入依赖值。

集合注入：bean 对象中的 Array，List，Map 等集合属性注入值。

内部 bean 注入：本质仍然是 set 注入或构造器注入。

自动装配：基于包扫描 bean 配置。

7.1. Set 注入

```
public class SysuserController {  
  
    private ISysuserService sysuserService;  
  
    private String str;  
  
    public void setSysuserService(ISysuserService sysuserService) {  
  
        this.sysuserService = sysuserService;  
  
    }  
  
    public void setStr(String str) {  
  
        this.str = str;  
  
    }  
  
}
```

```
<bean id="userController" class="com.javasm.sys.controller.SysuserController">  
    <property name="sysuserService" ref="userService"></property>  
    <property name="str" value="admin"></property>  
</bean>
```

注意点 1: property 标签的 ref 与 value 两个属性的选择，如果需要注入的值是容器中的 bean 对象，则使用 ref 指定引用 bean 的 id。如果注入的值是简单类型（String，Date，Integer 等），则使用 value 指定值。

注意点 2: 另一种 set 注入方法：引入 p 命名空间，在 bean 节点中以“p:属性名=属性值”的方式为属性注入值。

```
public class SysuserServiceImpl implements ISysuserService {

    private ISysuserMapper um;

    private String uname;

}

<bean id="sysuserService"
class="com.javasm.sys.service.SysuserServiceImpl" p:uname="fyt" p:um-
ref="userMapper">

</bean>

<bean id="userMapper" class="com.javasm.sys.mapper.SysuserMapperImpl">

</bean>
```

7.2. 构造器注入

```
public class SysUserController {

    private ISysuserService sysuserService;

    private String str;

    private Integer uage;

    public SysUserController(ISysuserService sysuserService,String str)

    {

        System.out.println(" SysUserController init——");

        this.sysuserService = sysuserService;

        this.str = str;

    }

}

<bean id="UserController" class="com.javasm.sys.controller.SysUserController">

    <constructor-arg name="sysuserService" ref="userService"></constructor-arg>

    <constructor-arg name="str" value="张三"></constructor-arg>

</bean>
```

注意点 1: constructor-arg 标签中的 name 属性表示形参名; index 属性表示形参索引号。

7.3. 集合注入

```
public class SysroleController {

    private String strAttr;

    private Integer intAttr;

    private List<String> strList;

    private List<Sysuser> usersList;

    private Integer[] intsArray;

    private Map<String, Integer> strMaps;

    //TODO setter,getter 生成

}
```

```
<bean id="sysroleController" class="com.javasm.sys.controller.SysroleController">

    <property name="strAttr" value="stringValue"></property>

    <property name="intAttr" value="100"></property>

    <property name="strList">

        <list>

            <value>老贾</value>

            <value></value>

            <value>ccc</value>

        </list>

    </property>

    <property name="usersList">

        <list>

            <bean class="com.javasm.sys.entity.Sysuser">

                <constructor-arg index="0" value="admin"></constructor-arg>

                <constructor-arg index="1" value="admin"></constructor-arg>

                <constructor-arg index="2" value="11111"></constructor-arg>

            </bean>

            <bean class="com.javasm.sys.entity.Sysuser">
```

```

        <constructor-arg index="0" value="admin2"></constructor-arg>

        <constructor-arg index="1" value="admin2"></constructor-arg>

        <constructor-arg index="2" value="2222"></constructor-arg>

    </bean>

</list>

</property>

<property name="intsArray">
    <array>
        <value>111</value>
        <value>222</value>
        <value>333</value>
    </array>
</property>

<property name="strMaps">
    <map>
        <entry key="a" value="111"></entry>
        <entry key="b" value="222"></entry>
    </map>
</property>
</bean>

```

7.4. 内部 bean 注入

以上 7.3 案例中 `usersList` 属性的注入即内部 bean 注入，当前 bean 对象不能被其他对象引用。

7.5. 自动装配

通过注解方式来进行 DI 注入；一般与包扫描结合使用。

<pre><context:component-scan package="com.javasm"></context:component-scan></pre>	base-
<pre>@Component public class SysuserController { @Resource private ISysuserService sysuserService; }</pre>	
<p>如果没有指定 name 属性，并且按照默认的名称找不到依赖对象时，@Resource 注解会回退到按类型装配。但一旦指定了 name 属性，就只能按名称进行装配。</p>	

注意点 1：在 java 代码中使用@Autowired 或@Resource 注解方式进行装配，这两个注解的区别是：

@Autowired 默认按类型装配，类型匹配不上，再按照形参名称装配。

@Resource 默认按形参名称装配，当找不到与名称匹配的 bean 才会按类型装配。

注意点 2：Resource 注解可以指定名称@Resource(name="userService"),指定后则只能按照名称进行装配，一般没有必要。

8. 类中常用注解

@Scope 用于指定 scope 作用域的（用在类或方法上），等价于 bean 标签的 scope 属性

@PostConstruct 用于指定初始化方法（用在方法上），等价于 bean 标签的 init-method 属性

@PreDestroy 用于指定销毁方法（用在方法上），等价于 bean 标签的 destroy-method 属性

9. Xml 中常用标签

1. context:property-placeholder 标签，导入类路径下的 properties 文件到 spring 容器。

定义 dao.xml 文件：

```
<context:property-placeholder location="classpath:jdbc.properties" ignore-
unresolvable="true"></context:property-placeholder>

<bean id="dataSource" class="com.alibaba.druid.pool.DruidDataSource" init-method="init" destroy-
method="close" >

    <property name="driverClassName" value="${jdbc.driver}"></property>
```

```
<property name="url" value="${jdbc.url}"></property>

<property name="username" value="${jdbc.username}"></property>

<property name="password" value="${jdbc.password}"></property>

</bean>
```

注意点：context:property-placeholder 标签的 ignore-unresolvable 属性是当引入了多个 properties 文件时必须指定的属性。

2.import 标签，导入其它 spring 风格的 xml 文件

在 Spring.xml 总配置中通过 import 标签包含 dao.xml:

```
<import resource="classpath:dao.xml"></import>
```

10. Spring 测试环境搭建

10.1. 环境准备

加入 Junit4.0 jar 包

加入 spring-test-xxx.jar

使用@RunWith(SpringJUnit4ClassRunner.class)注解描述当前测试用例依赖启动器

使用@ContextConfiguration(locations = { "classpath:applicationContext.xml" }) 配置 spring 配置文件

10.2. 编写测试类

```
@RunWith(SpringJUnit4ClassRunner.class)
@ContextConfiguration(locations = { "classpath:applicationContext.xml" })
public class TestAnnotation {
    @Resource
    private PersonDao personDao;
```

@Test

```
public void testAnnotation(){  
    personDao.add();  
}  
}
```