Spring01

今日学习目标：

spring 的 IOC 的容器

spring 的 IOC 的入门案例（spring 中容器是怎么创建对象的）

实例化 Bean 的三种方式

实例化 Bean 的三种方式

Bean 的生命周期的配置

spring 的依赖注入（DI）

spring 的 IOC 的容器

1：理解程序的耦合和解耦

（1）什么是程序的耦合

在 A 类中，new B 就是耦合

- 案例一：使用反射技术创建 JDBC 连接对象

```
public class JdbcDemo1 {
    public static void main(String[] args) throws  Exception{
        //1.注册驱动
//        DriverManager.registerDriver(new com.mysql.jdbc.Driver());
        Class.forName("com.mysql.jdbc.Driver");

        //2.获取连接（Connection）
        Connection conn =
DriverManager.getConnection("jdbc:mysql://localhost:3306/itcastspring","root","root");
        //3.获取操作数据库的预处理对象（PreparedStatement）
        PreparedStatement pstm = conn.prepareStatement("select * from account");
        //4.执行SQL，得到结果集（ResultSet）
        ResultSet rs = pstm.executeQuery();
        //5.遍历结果集
        while(rs.next()){
            System.out.println(rs.getString("name"));
        }
        //6.释放资源
        rs.close();
        pstm.close();
        conn.close();
    }
}
```

- 案例二：使用工厂模式+反射技术创建对象（使用工厂类读取 xml 的配置，底层使用反射完成解耦）

（2）解决程序耦合的思路一：反射

（3）解决程序耦合的思路二：工厂模式解耦

在 resource 下，创建 applicationContext.xml

```
<?xml version="1.0" encoding="utf-8" ?>
<beans>
    <bean id="accountService" class="com.itheima.service.impl.AccountServiceImpl">
        <property name="accountDao" ref="accountDao"/>
    </bean>

    <bean id="accountDao" class="com.itheima.dao.impl.AccountDaoImpl"/>
</beans>
```

1.创建对象

1.使用 xml 文件去描述对象属性的值，以及对象之间的关系(定义一个 id 作为对象标识)

2.写一个工具类 BeanFactory.java，读取 xml 文件(applicationContext.xml)的内容，解析并创建对象

3.将对象存入到内存中（map<id,对象实例>），（表示对象创建,IOC）

4.读取对象描述的属性，如果有对象关联就将对应的对象传入该对象（表示对象注入,DI）

2.对象的获取/使用

1.在工具类里面创建静态 getbean 方法，通过 id 获取对象实例

2.在外部调用工具类 getbean 方法，获取对象实例

```
        //1.读取xml文件
        SAXReader reader = new SAXReader();
        InputStream in =
BeanFactory.class.getClassLoader().getResourceAsStream("applicationContext.xml");
        Document doc = reader.read(in);
        //2.将读到的javabean创建并存储到objMap里面,存储bean节点下的其他属性
        Element rootElement = doc.getRootElement();
        //获取所有bean节点
        List<Element> beans = rootElement.elements("bean");
        for (Element bean : beans) {
            String id = bean.attributeValue("id");
            String clazz = bean.attributeValue("class");
            Object obj = Class.forName(clazz).newInstance();
            //存储对象放置到objMap中
            objMap.put(id,obj);
            //存储bean节点下的其他属性
            List<Element> elements = bean.elements();
            if(elements!=null && elements.size()>0){
                // 存储bean节点下的其他属性放置到eleMap中
                eleMap.put(id, elements);
            }
        }
```

```
        //3.遍历eleMap，将javabean下的属性一一赋值
        for (String id : eleMap.keySet()) {
            Object obj = objMap.get(id);
            List<Element> elements = eleMap.get(id);
            for (Element element : elements) {
                if("property".equals(element.getName())){
                    String name = element.attributeValue("name");
                    String ref = element.attributeValue("ref");
                    if(ref!=null && !"".equals(ref)){
                        Object objProperty = objMap.get(ref);
                        //根据属性名拿到对应的属性
                        Field declaredField = obj.getClass().getDeclaredField(name);
                        //设置该属性为可访问的
                        declaredField.setAccessible(true);
                        //设置该属性的值为对应的对象
                        declaredField.set(obj,objProperty);
                    }
                }
            }
        }
    }
```

2：控制反转 IOC

（1）ioc 概念，控制反转，将对象的创建交给 spring 去管理，我们需要的时候去 spring 容器中取就可以了

（2）好处:实现了解耦 AccountDao dao = new AccountDao(); AccountService service = new AccountService (); 现在就是要什么直接去容器中取。

spring 的 IOC 的入门案例（spring 中容器是怎么创建对象的）

BeanFactory 才是 Spring 容器中的顶层接口。

ApplicationContext 是它的子接口。

BeanFactory 和 ApplicationContext 的区别：

创建的方式都表示单例对象。

创建对象的时间点不一样。

ApplicationContext：只要一读取配置文件，默认情况下就会创建对象。（立即加载）

BeanFactory：什么时候使用什么时候创建对象。（延迟加载）

ApplicationContext 接口的实现类

（1）ClassPathXmlApplicationContext： （重点）

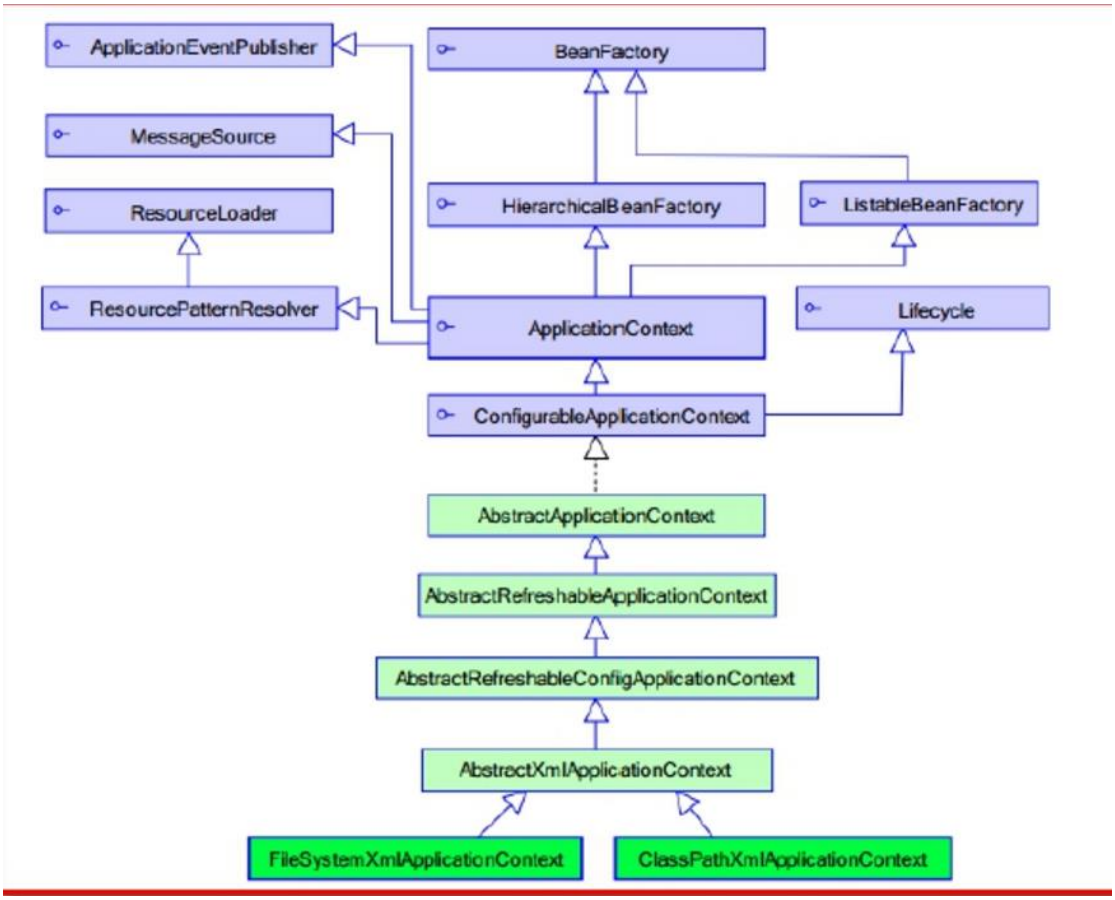
它是从类的根路径下加载配置文件 推荐使用这种

（2）FileSystemXmlApplicationContext：

它是从磁盘路径上加载配置文件，配置文件可以在磁盘的任意位置。

（3）AnnotationConfigApplicationContext: （第 2 天讲）

当我们使用注解配置容器对象时，需要使用此类来创建 spring 容器。它用来读取注解。



实例化 Bean 的三种方式

l 第一种：采用无参数的构造方法方式实例化（用的最多）

（1）applicationContext.xml

```
<!--创建Bean的三种方式 -->
<!-- 第一种方式：使用默认构造函数创建。
        在spring的配置文件中使用bean标签，配以id和class属性之后，且没有其他属性和标签时。
        采用的就是默认构造函数创建bean对象，此时如果类中没有默认构造函数，则对象无法创建。
-->
<bean id="accountService" class="com.itheima.service.impl.AccountServiceImpl"></bean>
```

（2）需要无参的构造方法

```
public class AccountServiceImpl implements AccountService {
    //可以不写，默认就是无参构造
    public AccountServiceImpl(){
        System.out.println("对象创建了");
    }
}
```

第二种：采用静态工厂实例化的方式

（1）applicationContext.xml

```
<!-- 第二种方式：使用工厂中的静态方法创建对象（使用某个类中的静态方法创建对象，并存入spring容器）
-->
<bean id="accountService" class="com.itheima.factory.StaticFactory" factory-
method="getAccountService"></bean>
```

（2）AccountServiceImpl.java

```
public class AccountServiceImpl implements AccountService {

}
```

（3）StaticFactory.java，静态工厂类

```
/**
 * 模拟一个工厂类（该类可能是存在于jar包中的，我们无法通过修改源码的方式来提供默认构造函数）
 */
public class StaticFactory {

    public static AccountService getAccountService(){
        return new AccountServiceImpl();
    }
}
```

第三种：采用实例工厂（非静态的）实例化的方式

（1）applicationContext.xml

```
<!-- 第三种方式：使用普通工厂中方法创建对象（使用某个类中的方法创建对象，并存入spring容器） -->
<bean id="instanceFactory" class="com.itheima.factory.InstanceFactory"></bean>
<bean id="accountService" factory-bean="instanceFactory" factory-
method="getAccountService"></bean>
```

（2）AccountServiceImpl.java

```
public class AccountServiceImpl implements AccountService {}
```

（3）InstanceFactory.java

```
/**
 * 模拟一个工厂类（该类可能是存在于jar包中的，我们无法通过修改源码的方式来提供默认构造函数）
 */
public class InstanceFactory {
    public AccountService getAccountService(){
        return new AccountServiceImpl();
    }
}
```

成

applicationContext.xml中的配置

```
<!-- bean的作用范围调整
    bean标签的scope属性：
        作用：用于指定bean的作用范围
        取值： 常用的就是单例的和多例的
        singleton：单例（默认值）一般创建对象单例（例如Service对象、Dao对象，数据源的对象...）
        prototype：多例。（1）：如果spring创建数据库连接对象Connection（每个线程使用的数据库连
                   接对象是不同的，保证线程安全）
                        （2）：Struts2（本身==多实例==，多线程），如果spring创建struts2的对
                   象，一定是多例（了解）
        request：作用于web应用的请求范围
        session：作用于web应用的会话范围
        global-session：作用于集群环境的会话范围（全局会话范围），当不是集群环境时，它就是
session
-->
<bean id="accountService" class="com.itheima.service.impl.AccountServiceImpl"
scope="prototype"></bean>
```

Bean 的生命周期的配置

单例对象

出生：当容器创建时对象出生

活着：只要容器还在，对象一直活着

死亡：容器销毁，对象消亡

==总结：单例对象的生命周期和容器相同==

多例对象

出生：当我们使用对象时spring框架为我们创建

活着：对象只要是在使用过程中就一直活着。

死亡：当对象长时间不用，且没有别的对象引用时，由Java的垃圾回收器回收

> ==总结：多例对象的生命周期和对象是否被使用有关。与容器是否被销毁无关。==

spring 的依赖注入（DI）

简单的说，就是框架把持久层对象传入业务层，而不用我们自己去获取。 这就是依赖注入。

依赖注入：

能注入的数据：有三类

（1）基本类型和String类型（值的注入）

（2）其他bean对象类型（在配置文件中或者注解配置过的bean）（对象的注入）

（3）复杂类型/集合类型（集合的注入）

能注入的方式：有三种

（1）第一种：使用构造函数提供

（2）第二种：使用set方法提供（使用p名称空间注入）（用的最多）

（3）第三种：使用注解提供（明天的内容）

第一种：构造函数注入 （了解）

1：AccountServiceImpl.java 提供传递参数的构造方法

2：applicationContext.xml（注入属性值）

3：测试类 Client.java

第一步：AccountServiceImpl.java 提供传递参数的构造方法。

```
public class AccountServiceImpl implements AccountService {
    //如果是经常变化的数据，并不适用于注入的方式
    private String name;
    private Integer age;
    private Date birthday;
    public AccountServiceImpl(){
    }
    public AccountServiceImpl(String name,Integer age,Date birthday){
        this.name = name;
        this.age = age;
        this.birthday = birthday;
    }
    public void  saveAccount(){
        System.out.println("service中的saveAccount方法执行
了。。。"+name+","+age+","+birthday);
    }
}
```

第二步：applicationContext.xml

```
<!--构造函数注入：
    使用的标签:constructor-arg
    标签出现的位置：bean标签的内部
    标签中的属性
        type：用于指定要注入的数据的数据类型，该数据类型也是构造函数中某个或某些参数的类型
        index：用于指定要注入的数据给构造函数中指定索引位置的参数赋值。索引的位置是从0开始
        name：用于指定给构造函数中指定名称的参数赋值
常用的
        =============以上三个用于指定给构造函数中哪个参数赋值==============================
        value：用于提供基本类型和String类型的数据
        ref：用于指定其他的bean类型数据。它指的就是在spring的Ioc核心容器中出现过的bean对象
    优势：
        在获取bean对象时，注入数据是必须的操作，否则对象无法创建成功。
    弊端：
        改变了bean对象的实例化方式，使我们在创建对象时，如果用不到这些数据，也必须提供。
-->
<bean id="accountService" class="com.itheima.service.impl.AccountServiceImpl">
    <constructor-arg name="name" value="泰斯特"></constructor-arg>
    <constructor-arg name="age" value="18"></constructor-arg>
    <constructor-arg name="birthday" ref="now"></constructor-arg>
</bean>
<!-- 配置一个日期对象 -->
<bean id="now" class="java.util.Date"></bean>
```

第三步：测试类 Client.java

```
public class Client {
    public static void main(String[] args) {
        //1.获取核心容器对象
        ApplicationContext ac = new
ClassPathXmlApplicationContext("applicationContext.xml");
        //2.根据id获取Bean对象
        AccountService as  = (AccountService)ac.getBean("accountService");
        as.saveAccount();
    }
}
```

第二种：**set 方法注入 （推荐使用）**

1：AccountServiceImpl2.java 提供属性的 set 方法。

2：applicationContext.xml（注入属性值）

3：测试类 Client.java

第一步：AccountServiceImpl2.java 提供属性的 set 方法。

```
public class AccountServiceImpl2 implements AccountService {
    //如果是经常变化的数据，并不适用于注入的方式
    private String name;
    private Integer age;
    private Date birthday;
    public void setName(String name) {
        this.name = name;
    }
    public void setAge(Integer age) {
        this.age = age;
    }
    public void setBirthday(Date birthday) {
        this.birthday = birthday;
    }
    public void  saveAccount(){
        System.out.println("service中的saveAccount方法执行
了。。。"+name+","+age+","+birthday);
    }
}
```

第二步：applicationContext.xml

```
<!-- set方法注入 更常用的方式
    涉及的标签：property
    出现的位置：bean标签的内部
    标签的属性
        name：用于指定注入时所调用的set方法名称
        value：用于提供基本类型和String类型的数据
        ref：用于指定其他的bean类型数据。它指的就是在spring的Ioc核心容器中出现过的bean对象
    优势：
        1：使用set方法创建对象时没有明确的限制，可以直接使用默认构造函数；
        2：使用set方法注入值或者对象，需要哪个属性只需要注入哪个属性
-->
<bean id="accountService2" class="com.itheima.service.impl.AccountServiceImpl2">
    <property name="name" value="小明" ></property>
    <property name="age" value="21"></property>
    <property name="birthday" ref="now"></property>
</bean>
<!-- 配置一个日期对象 -->
<bean id="now" class="java.util.Date"></bean>
```

使用 p 名称空间注入数据（本质还是调用 set 方法）（了解）

- （1）引入p名称空间

  ==xmlns:p="http://www.springframework.org/schema/p"==

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

- （2）使用p名称空间完成属性注入

  语法：普通属性p:属性名=""　　对象类型　p:属性名-ref=""

```
<!--
    使用p空间完成注入
    语法：普通属性     p:属性名=""        对象类型    p:属性名-ref=""
-->
    <bean id="accountService2" class="com.itheima.service.impl.AccountServiceImpl2"
          p:name="小刚" p:age="25" p:birthday-ref="now">
    </bean>
    <!-- 配置一个日期对象 -->
    <bean id="now" class="java.util.Date"></bean>
```

注入集合属性（复杂类型）

```
public class AccountServiceImpl3 implements AccountService {
    Object[] arrays;
    List<Object> list;
    Set<Object> set;
    Map<String,Object> map;
    Properties properties;
    public void setArrays(Object[] arrays) {
        this.arrays = arrays;
    }
    public void setList(List<Object> list) {
        this.list = list;
    }
    public void setSet(Set<Object> set) {
        this.set = set;
    }
    public void setMap(Map<String, Object> map) {
        this.map = map;
    }
    public void setProperties(Properties properties) {
        this.properties = properties;
    }
    public void saveAccount() {
        System.out.println("执行AccountServiceImpl中的saveAccount方法!
arrays:"+Arrays.toString(arrays)+"       list:"+list+"      set:"+set+"
map:"+map+"         properties"+properties);
    }
```

第二步：applicationContext.xml

```
<!-- 复杂类型的注入/集合类型的注入
     用于给 List 结构集合注入的标签：
          list array set
     用于个 Map 结构集合注入的标签:
          map    props
     结构相同，标签可以互换
-->
     <bean id="accountService3" class="com.itheima.service.impl.AccountServiceImpl3">
     <!--数组-->
     <!--在 spring 的集合注入中，array，list，set 是可以通用的-->
     <property name="arrays">
          <set>
               <value>张三</value>
               <value>22</value>
               <ref bean="date"></ref>
          </set>
     </property>
     <!--list 集合-->
     <property name="list">
          <set>
               <value>李四</value>
               <value>20</value>
               <ref bean="date"></ref>
          </set>
     </property>
     <!--set 集合-->
     <property name="set">
          <set>
               <value>王五</value>
               <value>25</value>
               <ref bean="date"></ref>
          </set>
     </property>
     <!--map 集合-->
     <property name="map">
          <map>
               <entry key="key001">
                    <value>赵六</value>
               </entry>
               <entry key="key002" value="23"></entry>
               <entry key="key003">
                    <ref bean="date"></ref>
               </entry>
          </map>
     </property>
     <!--properties 集合，和 map 集合很相似，也是键值对，键和值只能是 String-->
     <!--集合属性的应用场景：初始化系统中使用常量-->
     <property name="properties">
          <props>
               <prop key="driver">com.mysql.jdbc.Driver</prop>
               <prop key="url">jdbc:mysql:///itcastspring</prop>
               <prop key="username">root</prop>
               <prop key="password">root</prop>
          </props>
     </property>
</bean>
```

在 Service 中，注入 Dao

第一步：

在AccountServiceImpl上提供set方法。

```
public class AccountServiceImpl implements AccountService {

    AccountDao accountDao;
    // 注入：set方法，构造方法
    public void setAccountDao(AccountDao accountDao) {
        this.accountDao = accountDao;
    }
}
```

第二步：applicationContext.xml

```
<bean id="accountService" class="com.itheima.service.impl.AccountServiceImpl">
    <property name="accountDao" ref="accountDao"></property>
</bean>
<bean id="accountDao" class="com.itheima.dao.impl.AccountDaoImpl"></bean>
```