day04 Spring基础

1. 什么是spring

spring是一个第三方的容器框架，用来管理对象

- 是反射工厂模式的应用
- 提供了很多的工具类：RestTemplate jdbc Template，RedisTemplate....
- spring框架提供了管理器对象：事务管理器对象
- spring提供了线程池的支持，支持异步任务，支持定时任务

servlet容器：httpservletrequest session servletContext

ServletContext:全局上下文对象，存储自定义对象，存储到该对象中的数据能够被所有的请求所共享

spring带来的最大好处：

- 解耦：下层变动-----不影响当前层，上层只依赖下层的抽象接口，不依赖其具体的实现类。

2. 入门使用

- 添加spring的4个核心包，一个commons-logging日志包，共5个jar包
- 创建spring风格的xml配置文件

```
  <bean id="sysuserController"
class="com.javasm.controller.SysuserController"></bean>
```

- 加载xml文件，实例化spring容器对象

```
  //1. 加载类路径下的xml文件，初始化BeanFactory工厂对象，创建一个bean对象
  AplicationContext ac = new
ClassPathXmlApplicationContext(spring.xml))
```

- 测试使用

```
//1. 加载类路径下的xml文件，初始化BeanFactory工厂对象，创建一个bean对象
AplicationContext ac = new ClassPathXmlApplicationContext(spring.xml))
//2. 根据id从容器中获取bean对象
(SysuserController)ac.getBean("sysuserController")
//3. 根据类型从容器中获取bean
SysuserController uc2 = ac.getBean(SysuserCOntroller.class)
```

总结

- spring管理的对象，默认是单例的
- 从spring中获取对象，有两种方式，byName,byType
- 有了spring后，除了实体类，以后尽量不再new对象，是把对象注册到spring容器，然后需要用的时候，从容器中get。

Spring容器的类图

最顶级的接口BeanFactory----->DefaultListableBeanFactory

BeanFactory----->容器接口ApplicationContext的实现类

FileSystemXmlApplicationContext 加载系统文件

ClasspathXmlApplicationContext 从类路径中寻找

AnnotationConfigApplicationContext

XmlWebApplicationContext

AnnocationConfigWebApplicationContext

3. spring中的几个概念

- IOC:控制反转，对象的管理由自身维护交给第三方的容器进行管理，称为控制反转

```
<!--spring通过反射SysuserServiceImpl对象-->
<bean id="sysuserService" class="com.javasm.service.impl.SysuserServiceImpl">
</bean>
```

- DI: 依赖注入：bean对象依赖谁，对象之间的关系也由spring容器进行管理

```
<bean id="sysuserController"
class="com.javasm.controller.SysuserController">
  <!--SysuserController依赖的us成员变量,也在容器,通过
property标签进行set注入-->
  <property name="us" ref="sysuserService"></property>
  <property name="str" value="aaa123"></property>
</bean>
```

5. bean标签属性

- id: bean的名称
- class：bean的类名
- scope: 对象的单例多例状态，singleton|prototype 默认是singleton单例。。单例模式会先进行实例化，多例模式不进行实例化，只有在调用的时候才会创建对象，且两次调用的对象非同一个对象
- lazy-init:true （懒汉单例模式）仍然是单例状态，延迟初始化， 默认是饿汉式单例模式

6. 指定bean对象的初始化方法

方法1：对jar包中的类，使用bean标签的init-method方法

```
一般用在jar包中的注册spring容器时,指定初始化与销毁的方法
init-method="init" ：  指定bean对象的初始化方法（使用数据库连接池进行举例，在无法更改导入的jar文件下，如何实现创建对象时的初始化工作）
destroy-method="close": 指定bean对象的销毁方法-----只有显示执行了ApplicaiotnContext的close方法才能够触发destroy-method。
```

方法2：对自定义的类，使用InitializingBean接口

```
如果自定义的类注册spring容器,需要指定初始化与销毁方法,可以通过InitializingBean接口实现.
进行方法(afterPropertiesSet)的重写------->spring会检测 配置的类是否实现了InitializingBean接口，若有实现则自动执行afterPropertiesSet方法进行初始化，（无法做销毁工作）
public class SysuserServiceImpl implements ISysuserSerivce,InitializingBean {
    //当该bean被实例化完成后,立即执行afterPropertiesSet方法进行初始化工作.
    @Override
    public void afterPropertiesSet() throws Exception {
        System.out.println("初始化方法.......");
    }
}
```

如果同时定义了两种初始化的方法则默认先走实现接口时重写的afterPropertiesSet()方法

7. ioc的实现方式（重要）

IOC: 把对象注册到spring容器中

方法1：bean标签，指定类名的方式进行bean注册，常用于jar包中的类注册到容器中使用

```
 <bean class="com.javasm.service.impl.SysuserServiceImpl">
</bean>
```

方法2：工厂注册bean：复杂对象注册到spring容器中,可以使用工厂注册bean

应用举例：将SqlSessionFactory对象注册到spring容器中

- 静态工厂方法：在标签属性中添加factory-method="静态方法名" ： 将类中的一个静态方法的返回值注册到容器中

  注意：并没有把SessionFactoryBean类注册到容器中，因此如果输出，则会有NoSuchBeanDefinitionException: No qualifying bean of type 异常

```
<!--静态工厂注册bean-->
<!--把SessionFactoryBean.createFactory方法的返回值注册到容器-->
<bean id="sqlSessionFactory" class="com.javasm.factory.SessionFactoryBean" factory-method="createFactory"></bean>
```

- 实例工厂方法：将类中的非静态方法的返回值注册到容器中

  注意，该方法同时将SessionFactoryBean2注册到了spring容器中，因此可以使用该对象

  ```
  <!--工厂对象-->
  <bean id="factoryBean2" class="com.javasm.factory.SessionFactoryBean2" >
  </bean>
  <!--factoryBean2.createFactory()-->
  <!--先注册factoryBean2这个类，然后通过factory-bean获取类的对象，然后经过actory-method将实例方法的返回值(对象)注册到spring中-->
  <bean id="sqlSessionFactory" factory-bean="factoryBean2" factory-method="createFactory">
  </bean>
  ```

方法3：**包扫描（自定义类的容器注册）**

```
对包下的所有类进行递归扫描，通过反射检测类上是否有指定注解，有的话将该类注册到容器中
特定的注解：
@Controller:注解控制层的bean
    @Scope("prototype")：指定多例状态，默认是单例（用在类或方法上）
    @PostConstruct：指定对象的初始化方法（用在方法上）
    @PreDestroy：指定对象的销毁方法（用在方法上）
@Service：注解服务层的bean
@Repository：注解dao层的bean
@Component：注解其他的bean
<context:component-scan base-package="com.javasm">
</context:component-scan>
```

包扫描的注意事项

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。

5. di的实现方式（重要）

---

dl：**依赖注入，维护对象之间的依赖关系**

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。

方法1：set注入，要求成员变量必须有set和get方法

标签中ref|value:

ref：引用其他bean的id值

value是用来给简单类型赋值。

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。

```
<bean id="ds"
class="com.alibaba.druid.pool.DruidDataSource" init-
method="init" destroy-method="close">
  <!--set注入,调用对象的set方法进行值得注入;ref|value:ref用来
引用其他bean的id,value用来给简单类型赋值-->
  <property name="url"
value="jdbc:mysql://localhost:3306/crm"></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>
</bean>

另一种方式：通过 引入p命名空间
<bean id="ds"
class="com.alibaba.druid.pool.DruidDataSource"
p:name="url">
<bean id="ds"
class="com.alibaba.druid.pool.DruidDataSource"
p:user_ref="user">
```

方法2：构造器注入()

```
通过反射：获取其有参构造
index:形参索引
name: 形参名
value:形参的值(简单类型)
ref:对象类型的参数(其他注入对象的bean的id值)
<!--new SysuserDaoImpl(ds,12)-->
<bean class="com.javasm.dao.impl.SysuserDaoImpl">
  <!--index:形参索引;name:形参名-->
  <constructor-arg index="0" ref="ds"></constructor-
arg>
  <constructor-arg name="sint" value="12">
</constructor-arg>
</bean>
```

方法3：集合注入

```
// 需要初始化的参数
```

```java
public class SysuserDaoImpl implements ISysuserDao {
    private DataSource ds;
    private Integer sint;
    private List<String> strList;
    private Integer[] ints;
    private Map<String,Double> douMap;
    private Sysuser suser;
   public SysuserDaoImpl(DataSource ds,Integer sint) {
        this.ds = ds;
        this.sint=sint;
    }
   public Sysuser getSuser() {
        return suser;
    }

    public void setSuser(Sysuser suser) {
        this.suser = suser;
    }
}
public class Sysuser {
    private String uname;
    private String upwd;
}
```

```xml
// di 通过集合注入的方法
<!--new SysuserDaoImpl(ds,12)-->
<bean class="com.javasm.dao.impl.SysuserDaoImpl">
  <!--index:形参索引;name:形参名-->
  <constructor-arg index="0" ref="ds"></constructor-arg>
  <constructor-arg name="sint" value="12"></constructor-arg>
  <property name="strList">
    <list>
      <value>aaa</value>
      <value>bbb</value>
    </list>
  </property>
  <property name="ints">
    <array>
      <value>11</value>
```

```
        <value>22</value>
      </array>
    </property>
    <property name="douMap">
      <map>
        <entry key="a" value="1.2"></entry>
        <entry key="b" value="1.3"></entry>
      </map>
    </property>
  </bean>
```

方法4：内部bean注入

```
<bean class="com.javasm.dao.impl.SysuserDaoImpl">
  <property name="suser">
  此标签放在proerty内部，该sysuser这个类不会被引入到spring容器中。
       <bean class="com.javasm.entity.Sysuser"></bean>
  </property>
</bean>
```

方法5：**自动装配（重要）**

在开启了包扫描后，可以通过注解(@AutoWired|@Resource)给成员变量赋值（dI自动赋值）

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。

- 如果接口有多个实现类则会报异常：因为spring不知道该找哪个实现类进行装配（因为在定义成员变量时一般都是定义的接口类型 比如：Private ISysuserService user--------> 如果ISysuserService有两个实现类，则会报异常，因为不知道需要赋值的是哪个实现类）----------------> 产生的异常：beans.factory.NoUniqueBeanDefinitionException

- 如果该成员变量（类对象）没有在spring容器中，则会报错。

@AutoWired: 先byType 再byName

@Resource:先byName再byType(一般用这个)

9.xml文件的补充

```
通过import标签可以导入另外的xml文件中的id名为""的类--->xml之间的嵌套
<import resource="dao.xml"></import>
通过context:property-placeholder标签可以引入其他配置文件中，将其他配置文件中的数据引入到spring容器中
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。
<context:property-placeholder location="jdbc.properties"></context:property-placeholder>
引入的方式，通过${key名}
<bean id="ds" class="com.alibaba.druid.pool.DruidDataSource" init-method="init" destroy-method="close">
  <property name="url" value="${jdbc.url}"></property>
  <property name="driverClassName" value="${jdbc.driver}"></property>
  <property name="username" value="${jdbc.username}"></property>
  <property name="password" value="${jdbc.password}"></property>
  <property name="initialSize" value="${jdbc.initialSize}"></property>
</bean>
```

10 反射工厂模式的应用

简单静态工厂：这种工厂没有可扩展性

反射工厂：基于反射+配置的方式提高代码的可扩展性，

- 定义配置 文件（名字type：全类名）
- ，通过property对象，将配置文件的信息导入到property对象中

- p.getProperty(type)获取全类名clzName
- 通过clz = Class.forName(clzName)获取type对应的类对象
- clz.newInstance()，获取该类型所对应的对象