Day01的主要学习内容

一、相关概念

- 1.1 控制反转：IOC(Inversion of Control)
  - 在使用Spring之前，在使用对象时，是由我们取主动new外部类创造对象的，在使用Spring后，此过程中对象创建控制权由程序转移到外部，此思想称为控制反转。
  - 在service层难免会使用DAO层的对象，在以前是自己new的，而现在交由外部来控制，而这个外部就是Spring的IOC容器。
- 1.2 Spring和IOC以及IOC容器之间的关系是什么呢?
  - Spring技术对IOC思想进行了实现
  - Spring提供了一个容器，称为==IOC容器==，用来充当IOC思想中的"外部"
  - IOC思想中的别人[外部]指的就是Spring的IOC容器
- 1.3 IOC容器的作用
  - IOC容器负责一切对象的创建以及初始化操作主要是DAO层和Service层的类对象
- 1.4 IOC容器中存放的是什么?
  - 是所有类的对象，这一个个的对象在IOC容器中被统一称为==bean==
  - IOC容器中放的就是一个个的Bean对象

> 在项目中不可避免的会存在Service层与DAO层的交互，一般地，在Service层中都会调用对应的DAO层的对象，或者是创建其余DAO层的对象，这时就需要用到==DI(Dependency Injection)==

业务层实现

```
public class BookServiceImpl implements BookService {

    private BookDao bookDao;

    public void save() {
        bookDao.save();
    }
}
```

依赖dao对象运行

数据层实现

```
public class BookDaoImpl implements BookDao {
    public void save() {
        System.out.println("book dao save ...");
    }
}
```

- 1.5 DI(Dependency Injection)
  - 在容器中建立bean与bean之间的依赖关系的整个过程，称为依赖注入
  - 业务层要用数据层的类对象，以前是自己new的,现在自己不new了，靠别人[外部其实指的就是IOC容器]来给注入进来,这种思想就是依赖注入
- 1.6 如何实现依赖注入呢?
  - 使用IOC容器管理bean
  - 在IOC容器内将有依赖关系的bean进行关系绑定（DI）

二、Spring在maven中的坐标

```
<dependencies>
    <dependency>
        <groupId>org.springframework</groupId>
        <artifactId>spring-context</artifactId>
        <version>5.3.20</version>
    </dependency>
    <dependency>
        <groupId>junit</groupId>
        <artifactId>junit</artifactId>
        <version>4.12</version>
        <scope>test</scope>
    </dependency>
</dependencies>
```

注：在JDK17版本中使用Spring的5.3.20版本，5.2.10.RELEASE与JDK17不兼容

三、注解的使用

- 3.1 注解创建Bean
  - 删除原有的applicationContext.xml文件
  - 在需要创建对象的类上添加注解`@Component/@Controller/@Service/@Repository`

```
@Component("bookDao")
public class BookDaoImpl implements BookDao {
    public void save() {
        System.out.println("book dao save ..." );
    }
}
```

  - **REMARK**
    - `@Component`注解不可以添加在接口上，因为接口是无法创建对象。
    - 一个`@Component`就相当于一个bean结点
    - 可以在`@Component("bookDao")`来指定这个bean的id
    - 注解的bean的名称默认是类名第一个字母小写后的名称
    - `@Component/@Controller/@Service/@Repository`四个注解的作用一致
    - 但是为了规范，一般地
      - `@Repository`用于DAO的类
      - `@Service`用于Service层的类
      - `@Controller`用于Controller层的类
      - `@Component`没有特定的范围

- 3.2 Bean中依赖的注入

> Spring为了使用注解简化开发，并没有提供构造函数注入、setter注入对应的注解，只提供了自动装配的注解实现。

- **3.2.1 在BookServiceImpl类的bookDao属性上添加@Autowired注解**

```
@Service
public class BookServiceImpl implements BookService {
    @Autowired
    private BookDao bookDao;
    public void save() {
        System.out.println("book service save ...");
        bookDao.save();
    }
}
```

- **REMARK**
  - @Autowired可以写在属性上，也可也写在setter方法上，最简单的处理方式是写在属性上并将setter方法删除掉
  - 为什么setter方法可以删除呢?
    - 自动装配基于反射设计创建对象并通过暴力反射为私有属性进行设值
    - 普通反射只能获取public修饰的内容
    - 暴力反射除了获取public修饰的内容还可以获取private修改的内容
    - 所以此处无需提供setter方法
  - @Autowired是按照类型注入的

- **3.2.2 当BookDao接口有多个实现类时**

- @Autowired，按照类型注入，无法区分到底注入哪个对象

  此时引入@Qualifier("bookDao1"),但是此时必须给先给两个DAO类的==bean==起名称

```
@Repository("bookDao")
public class BookDaoImpl implements BookDao {
    public void save() {
        System.out.println("book dao save ..." );
    }
}
@Repository("bookDao2")
public class BookDaoImpl2 implements BookDao {
    public void save() {
        System.out.println("book dao save ...2" );
```

```
        }
    }
```

```
@Service
public class BookServiceImpl implements BookService {

    @Autowired
    private BookDao bookDao;
```

```
@Repository("bookDao1")
public class BookDaoImpl implements BookDao {
    public void save() {
        System.out.println("book dao save ...");
    }
}
```

```
@Repository("bookDao2")
public class BookDaoImpl2 implements BookDao {
    public void save() {
        System.out.println("book dao save ...2");
    }
}
```


> 因为按照类型会找到多个bean对象，此时会按照bookDao名称去找，因为IOC容器只有名称叫bookDao1和bookDao2,所以找不到，会报NoUniqueBeanDefinitionException

- **Service层注解配置如下**

```
@Service
public class BookServiceImpl implements BookService {
    @Autowired
    @Qualifier("bookDao1")
    private BookDao bookDao;

    public void save() {
        System.out.println("book service save ...");
        bookDao.save();
    }
}
```

- **REMARK**
  - 当根据==类型==在容器中找到多个bean时,注入参数的属性名private BookDao bookDao又和容器中bean的名称不一致，这个时候，就需要使用到@Qualifier来指定==注入哪个名称的bean对象==。
  - @Qualifier注解后的值就是需要注入的bean的名称。
  - ***@Qualifier不能独立使用，必须和@Autowired一起使用***

- **3.2.3 简单类型的注入**

```
@Repository("bookDao")
public class BookDaoImpl implements BookDao {
    @Value("itheima")
    private String name;
    public void save() {
        System.out.println("book dao save ..." + name);
```

```
        }
    }
```

- **REMARK**
  - 注意数据格式要匹配，如将"abc"注入给int值，这样程序就会报错。

> 当我们使用注解创建完bean并且注入依赖以后，仅有这些显然是不足以支持程序运行的，Spring并不知道我们对这些类使用了注解,为了让Spring框架能够扫描到写在类上的注解，需要在配置文件上进行包扫描

```
    <?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">
        <context:component-scan base-package="com.itheima"/>
    </beans>
```

以上仍是使用配置文件的方式,这种方式也需要被改进

3.3 纯注解开发

> Spring3.0开启了纯注解开发模式，使用Java类替代配置文件，开启了Spring快速开发赛道

- **3.3.1 删除applicationContext.xml,创建一个配置类`SpringConfig`**

```
    public class SpringConfig {
    }
```

- **3.3.2 标识该类为配置类**

  在配置类上添加`@Configuration`注解，将其标识为一个配置类,替换`applicationContext.xml`

```
    @Configuration
    public class SpringConfig {
    }
```

- **3.3.3 用注解替换包扫描配置**

```
    @Configuration
    @ComponentScan("com.itheima")
```

```
public class SpringConfig {
}
```

- **REMARK**
  - @Configuration注解用于设定当前类为配置类
  - @ComponentScan注解用于设定扫描路径，此注解只能添加一次，多个数据请用数组格式 `@ComponentScan({com.itheima.service","com.itheima.dao"})`
  - 读取Spring核心配置文件初始化容器对象切换为读取Java配置类初始化容器对象

```
    //加载配置文件初始化容器
    ApplicationContext ctx = new
ClassPathXmlApplicationContext("applicationContext.xml");
    //加载配置类初始化容器
    ApplicationContext ctx = new
AnnotationConfigApplicationContext(SpringConfig.class);
```

3.4 IOC/DI注解开发管理第三方bean

> 前面定义bean的时候都是在自己开发的类上面写个注解就完成了，但如果是第三方的类，这些类都是在jar包中，我们没有办法在类上面添加注解，这个时候该怎么办?遇到上述问题，我们就需要有一种更加灵活的方式来定义bean,这种方式不能在原始代码上面书写注解，一样能定义bean,这就用到了一个全新的注解@Bean。

- 案例:**完成对Druid数据源的管理**

- **3.4.1 引入外部配置类**

对于数据源的bean,我们新建一个JdbcConfig配置类，并把数据源配置到该类下。

```
public class JdbcConfig {
    @Bean
    public DataSource dataSource(){
        DruidDataSource ds = new DruidDataSource();
        ds.setDriverClassName("com.mysql.jdbc.Driver");
        ds.setUrl("jdbc:mysql://localhost:3306/spring_db");
        ds.setUsername("root");
        ds.setPassword("root");
        return ds;
    }
}
```

- **REMARK**
  - 注意该方法的返回值就是要创建的Bean对象类型
  - @Bean注解的作用是将方法的返回值制作为Spring管理的一个bean对象
  - 如果有多个bean要被Spring管理，直接在配置类中多些几个方法，方法上添加@Bean注解即可。

> 这个配置类如何能被Spring配置类加载到，并创建DataSource对象在IOC容器中?

- **3.4.2 使用`@Import`引入**

这种方案==JdbcConfig类==可以不用加`@Configuration`注解，但是必须在Spring配置类上使用`@Import`注解手动引入需要加载的配置类

```
@Configuration
@Import({JdbcConfig.class})
public class SpringConfig {

}
```

- **REMARK**
  - `@Import`参数需要的是一个数组，可以引入多个配置类。
  - `@Import`注解在配置类中只能写一次，下面的方式是==不允许的==