Spring02、IoC基础

狂神说 _ SUIP 分类：学习笔记 创建时间：2021/04/13 10:46 ☑字体 ☐皮肤 最后修改于： 2021/04/13 15:27

🏆 2、IoC基础

新建一个空白的maven项目

📢 2.1 分析实现

我们先用我们原来的方式写一段代码 .

先写一个UserDao接口

```
1. public interface UserDao {
2.     public void getUser();
3. }
```

再去写Dao的实现类

```
1. public class UserDaoImpl implements UserDao {
2.     @Override
3.     public void getUser() {
4.         System.out.println("获取用户数据");
5.     }
6. }
```

然后去写UserService的接口

```
1. public interface UserService {
2.     public void getUser();
3. }
```

最后写Service的实现类

```
1. public class UserServiceImpl implements UserService {
2.     private UserDao userDao = new UserDaoImpl();
3.
4.     @Override
5.     public void getUser() {
6.         userDao.getUser();
7.     }
8. }
```

测试一下

```
1. @Test
2. public void test(){
3.     UserService service = new UserServiceImpl();
4.     service.getUser();
5. }
```

这是我们原来的方式，开始大家也都是这么去写的对吧 . 那我们现在修改一下 .

把Userdao的实现类增加一个 .

```java
public class UserDaoMySqlImpl implements UserDao {
    @Override
    public void getUser() {
        System.out.println("MySql获取用户数据");
    }
}
```

紧接着我们要去使用MySql的话，我们就需要去service实现类里面修改对应的实现 .

```java
public class UserServiceImpl implements UserService {
    private UserDao userDao = new UserDaoMySqlImpl();

    @Override
    public void getUser() {
        userDao.getUser();
    }
}
```

在假设, 我们再增加一个Userdao的实现类 .

```java
public class UserDaoOracleImpl implements UserDao {
    @Override
    public void getUser() {
        System.out.println("Oracle获取用户数据");
    }
}
```

那么我们要使用Oracle，又需要去service实现类里面修改对应的实现 . 假设我们的这种需求非常大，这种方式就根本不适用了, 甚至反人类对吧，每次变动，都需要修改大量代码 . 这种设计的耦合性太高了, 牵一发而动全身 .

那我们如何去解决呢？

我们可以在需要用到他的地方，不去实现它，而是留出一个接口，利用set，我们去代码里修改下 .

```java
public class UserServiceImpl implements UserService {
    private UserDao userDao;
    // 利用set实现
    public void setUserDao(UserDao userDao) {
        this.userDao = userDao;
    }

    @Override
    public void getUser() {
        userDao.getUser();
    }
}
```

现在去我们的测试类里，进行测试；

```java
@Test
public void test(){
    UserServiceImpl service = new UserServiceImpl();
    service.setUserDao( new UserDaoMySqlImpl() );
    service.getUser();
    //那我们现在又想用Oracle去实现呢
    service.setUserDao( new UserDaoOracleImpl() );
    service.getUser();
}
```

大家发现了区别没有？可能很多人说没啥区别 . 但是同学们，他们已经发生了根本性的变化，很多地方都不一样了 . 仔细去思考一下，以前所有东西都是由程序去进行控制创建，而现在是由我们自行控制创建对象，把主动权交给了调用者 . 程序不用去管怎么创建,怎么实现了 . 它只负责提供一个接口 .

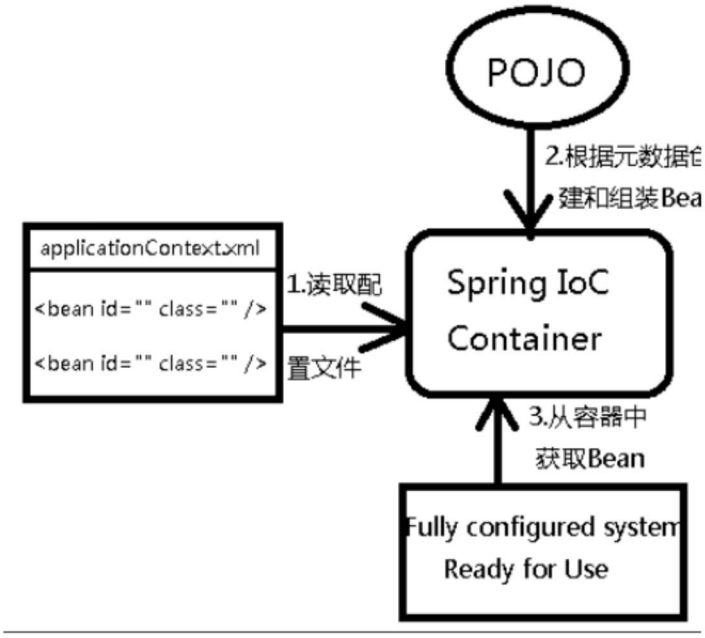
这种思想，从本质上解决了问题，我们程序员不再去管理对象的创建了，更多的去关注业务的实现．耦合性大大降低．这也就是IOC的原型！

2.2 IOC本质

控制反转IoC(Inversion of Control)，是一种设计思想，DI(依赖注入)是实现IoC的一种方法，也有人认为DI只是IoC的另一种说法。没有IoC的程序中，我们使用面向对象编程，对象的创建与对象间的依赖关系完全硬编码在程序中，对象的创建由程序自己控制，控制反转后将对象的创建转移给第三方，个人认为所谓控制反转就是：获得依赖对象的方式反转了。

IoC是Spring框架的核心内容，使用多种方式完美的实现了IoC，可以使用XML配置，也可以使用注解，新版本的Spring也可以零配置实现IoC。

Spring容器在初始化时先读取配置文件，根据配置文件或元数据创建与组织对象存入容器中，程序使用时再从Ioc容器中取出需要的对象。



采用XML方式配置Bean的时候，Bean的定义信息是和实现分离的，而采用注解的方式可以把两者合为一体，Bean的定义信息直接以注解的形式定义在实现类中，从而达到了零配置的目的。

控制反转是一种通过描述（XML或注解）并通过第三方去生产或获取特定对象的方式。在Spring中实现控制反转的是IoC容器，其实现方法是依赖注入（Dependency Injection,DI）。