SpringBoot- 全注解下的 SpringIOC

作者：pleaseok
原文链接：https://ld246.com/article/1544358999132
来源网站：链滴

SpringBoot - 全注解下的SpringIOC

springboot 是基于 springframework的全注解的web框架，它无需配置xml，所以相比较于spring+pringmvc它可以节省更多的开发周期

什么是IOC(Inversion of Controller)?

IOC即是控制反转，把创建对象的控制权交给第三方则叫反转。正转则是一般性的j2ee new对象式的发，多个Bean的容器则叫做IOC容器。反转大大降低了代码的耦合性。

准备 - @Configuration @Bean的简单使用

```
@Configuration
public class Appconfig{
    @Bean(name="user")
    public User initUser(){
        User user = new User();
        user.setId(1L);
        user.setUserName("user_name_1");
        user.setNote("note_1");
        return user;
    }

}
```

@Configuration 代表这是一个java配置文件，Spring的容器会根据它来生成IOC容器去装配Bean

@Bean 代表将initUser方法返回的POJO装配到IOC容器中，而其属性name定义这个bean的名称，果没有配置它，则将方法名称"initUser"作为Bean的名称装配到Spring IOC容器中。

使用实例:

```
public class IOCTest{
    private static Logger log = Logger.getLogger(IOCTest.class);
    public static void main(String[] args){
        ApplicationContext ctx = new AnnotationConfigApplicationContext(AppConfig.class);
        User user = ctx.getBean(User.class);
        log.info(user.getId());
    }

}
```

起步 - 如何去装配Bean

1.扫描装配Bean

因为如果用@Bean去一个个装配的话会显得很繁琐，所以推荐使用扫描装配。

@Component:标明哪个类被扫描进入Spring IOC容器。

@ComponentScan:标明采用何种策略去扫描装配Bean。

```
@Component("user")
public class user{
    @Value("1")
    private Long id;
    @Value("user_name_1")
    private String usename;
    @Value("note_1")
    private String note;
    /**setter and getter**/
}
```

@Component表示这个类将会被SpringIOC容器扫描装配，其中配置的"user"则是作为Bean的名称
为了让Spring IOC容器装配这个类，需要改造类APPConfig

```
@Configuration
@ComponentScan
public class Appconfig{
}
```

这里加入@ComponentScan意味着它会进行扫描，但是它只会扫描类APPConfig所在的当前包和其包(还可以指定扫描包，看下)。所以也要把User.java移动一下位置。

测试

```
ApplicationContext ctx = new AnnotationConfigApplicationContext(AppConfig.class);
User user = ctx.getBean(User.class);
log.info(user.getId());
```

@ComponentScan扫描指定包：

@ComponentScan("com.springboot.demo.*")

@ComponentScan(basePackage={"com.springboot.demo.pojo"})

@ComponentScan(basePackageClasses = {User.class})

Q：当标注了@Service的类在被扫描的包当中，则如何使它不被扫描?

A：还是使用@ComponentScan，不过另加一个参数即可:

```
@ComponentScan("com.springboot.demo.*",excludeFilters = {@Filter(classes = {Service.class}
})
```

2.装配第三方Bean

当我们需要引入第三包的时候，很可能 希望把第三方包的类对象也放入到SpringIOC的容器中，这是用@Bean注解就ok了。

举栗(引入DBCP数据源)：

```
<dependency>
    <groupId>org.apache.commons</groupId>
    <artifactId>commons-dbcp2</artifactId>
</dependency>
<dependency>
    <groupId>mysql</groupId>
    <artifactId>mysql-connector-java</artifactId>
```

```
</dependency>

@Bean(name="dataSource")
public DataSource getDataSource() {
    Properties props = new Properties();
    props.setProperty("driver", "com.mysql.jdbc.Driver");
    props.setProperty("url", "jdbc:mysql://localhost:3306/demo");
    props.setProperty("username", "root");
    props.setProperty("password", "123456");
    DataSource dataSource = null;
    try {
        dataSource = BasicDataSourceFactory.createDataSource(props);
    } catch (Exception e) {
        e.printStackTrace();
    }
    return dataSource;
}
```

加深 - 依赖注入(Dependency Injection)

依赖注入就是将实例变量传入到一个对象中去。(Dependency injection means giving an object its nstance variables)

- 控制反转是一种思想
- 依赖注入是一种设计模式

IoC框架使用依赖注入作为实现控制反转的方式，但是控制反转还有其他的实现方式，例如说ServiceLcator，所以不能将控制反转和依赖注入等同。

1. @Autowired

它是Spring中最常用的注解之一，十分重要，它会根据属性的类型(by type)找到对应对的Bean进行入。

栗子：

```
//动物接口
public interface Animal {
    public void use();
}

//人类接口
public interface Person {
    //使用动物服务
    public void service();
    //设置动物
    public void setAnimal(Animal animal);
}

public class Dog implements Animal{

    @Override
    public void use() {
```

```
        System.out.println("狗【"+Dog.class.getSimpleName()+"】是看门的");
    }
}

public class BussinessPerson implements Person {

    @Autowired
    private Animal animal = null;
    @Override
    public void service() {
        this.animal.use();
    }

    @Override
    public void setAnimal(Animal animal) {
        this.animal = animal;
    }
}
```

如上，spring容器会通过注解@Autowired将Dog注入到BussinessPerson实例中

Q：但是，加入又有一个动物类cat，那么它会选择哪个注入?怎么解决这个错误?

A: 因为 @Autowired首先会根据类型找到对应的Bean，如果类型不是唯一的，那么它会根据其属性称和Bean的名称进行匹配。如果匹配得上，就用该Bean，如果还是无法匹配就会抛出异常。

```
@Autowired
private Animal dog = null;
```

因为将animal修改为了dog，所以它会找到dog类

注：

1. 因为@Autowired是一个默认必须要找到对应Bean的注解，所以如果不能确定其标注属性是否存，或者允许这个被标注的属性为null。则可以配置@Autowired属性required为false。@Autowired(required =false)
2. 它除了可以标注属性外还可以标注方法。甚至还可以使用在方法参数上

```
    @Override
    @Autowired
    public void setAnimal(Animal animal) {
        this.animal = animal;
    }
```

它会使用setAnimal方法从IOC容器中找到对应的动物进行注入。

2. @Primary和@Quelifier

@Primary：它是一个修改优先权的注解，比如上面的例子，当我们有猫有狗时，假设这次使用猫，name只需要在猫类的定义上加入@Primary就可以了

```
@Component
@Primary
public class Cat implements Animal{
```

```
    ......
}
```

这里的@Primary的含义告诉SpringIOC容器，当发现有多个同类型的Bean时，请优先使用我进行注入.

Q: 如果@Primary作用在多个类上,其结果是IOC容器还是无法区分采用哪个Bean的实例进行注入,那该采取什么样的情况呢?

A: 可以使用@Quelifier结合@Autowired一起使用，则可以通过类型和名称一起查找Bean。

```
@Autowired
@Quelifier("dog")
private Animal animal = null;
```

通过上面的代码，即使cat已经标注了@Primary，但是我们还是可以拿到dog提供服务。

3. 带参数的构造方法实现注入

```
public class BussinessPerson implements Person {

    private Animal animal = null;

    public BussinessPerson(@Autowired @Quelifier("dog") Animal animal){
        this.animal = animal;
    }

    @Override
    public void service() {
        this.animal.use();
    }

    @Override
    public void setAnimal(Animal animal) {
        this.animal = animal;
    }
}
```

上面取消了对animal的@Autowired注解而是在构造参数中加入了@Autowired@Quelifier注解