基于注解的Spring应用

Spring除了使用xml进行配置以外，还可以使用注解方式进行配置，注解方式慢慢成为xml配置的替代方案。

1. 使用@Component代替<bean>

基于注解的，需要在配置文件添加context:component-scan标签。

bean标签的属性，对应了以下几个标签，

xml配置	注解	描述
<bean scope="">	@Scope	在类上或使用了@Bean注解的方法上，标注Bean的作用范围，取值为singleton或prototype
<bean lazy-init="">	@Lazy	在类上或使用了@Bean注解的方法上，标注Bean是否延迟加载，取值为true或false
<bean init-method="">	@PostConstruct	在方法上使用，标注Bean的实例化后执行的方法
<bean destroy-method="">	@PreDestroy	在方法上使用，标注Bean的销毁前执行方法

由于JavaEE开发是分层的，为了每层Bean标识的注解语义化更加明确，@Component衍生出如下三个注解，

@Component衍生注解	描述
@Repository	在Dao层类上使用
@Service	在Serevice层类上使用
@Controller	在Web层上使用

2. 依赖注入开发

Spring提供了如下的注解，用于在Bean内部进行属性注入：

属性注入注解	描述
@Value	使用在字段或方法上，用于注入普通数据

属性注入注解	描述
@Autowired	使用在字段或方法上，用于根据类型（byType）注入引用数据
@Qualifier	使用在字段或方法上，结合@Autowired，根据名称注入
@Resource	使用在字段或方法上，根据类型或名称注入

- 这些注解可以标识在字段上，也可以标识在set方法上
- @Autowired注解在遇到多个同一类型的Bean时，先根据名字去匹配，如果匹配不成功则报错；如果注入的是个list，会将找到的改类型都注入
- @Autowired和@Qualifier一起使用的例子：

```
@Autowired
@Qualifier("userDao2")
private UserDao userDao;
```

- @Resource，java包的注解，不指定名称参数时，根据类型注入，指定名称根据名称注入

3. 非自定义Bean的注入

非自定义Bean不能像自定义Bean一样，使用@Component进行管理，非自定义Bean要通过工厂的方式进行实例化，使用@Bean注解即可，@Bean的属性为BeanName，如不指定则为当前工厂方法的名称。

```
@Component
public class OtherBean {

    @Bean("dataSource")
    public DataSource dataSource() {
        DruidDataSource druidDataSource = new DruidDataSource();
        // 设置4个基本参数...
        return druidDataSource;
    }

}
```

如果需要注入参数，

```
@Component
public class OtherBean {

    @Bean("dataSource")
    public DataSource dataSource(
            @Value("${jdbc.username}") String username,
            @Autowired UserDao userDao
            ) {
```

```
        DruidDataSource druidDataSource = new DruidDataSource();
        // 设置4个基本参数...
        return druidDataSource;
    }

}
```

4. Bean配置类的开发

完全使用注解开发，摒弃xml的配置，还需要解决的是关于其他配置如何通过注解实现，比如配置@Component的扫描路径，配置properties文件的解析路径。

```
@Configuration  // 标注当前类是一个配置类（替代配置文件）+ @Component
@ComponentScan({"com.example"})  // component扫描路径
@PropertySource({"classpath:jdbc.properties"})  // properties扫描路径
public class SpringConfig {
}
```

测试如下：

```
@Test
public void test03() {
  // 使用注解方式加载Spring配置类
  ApplicationContext applicationContext = new
AnnotationConfigApplicationContext(SpringConfig.class);
  UserService userService = applicationContext.getBean("userService",
UserService.class);
  System.out.println(userService);
}
```

5. 其他注解

@Profile注解的作用与xml配置中的profile属性作用相同，进行环境切换，

```
<beans profile="test">
```

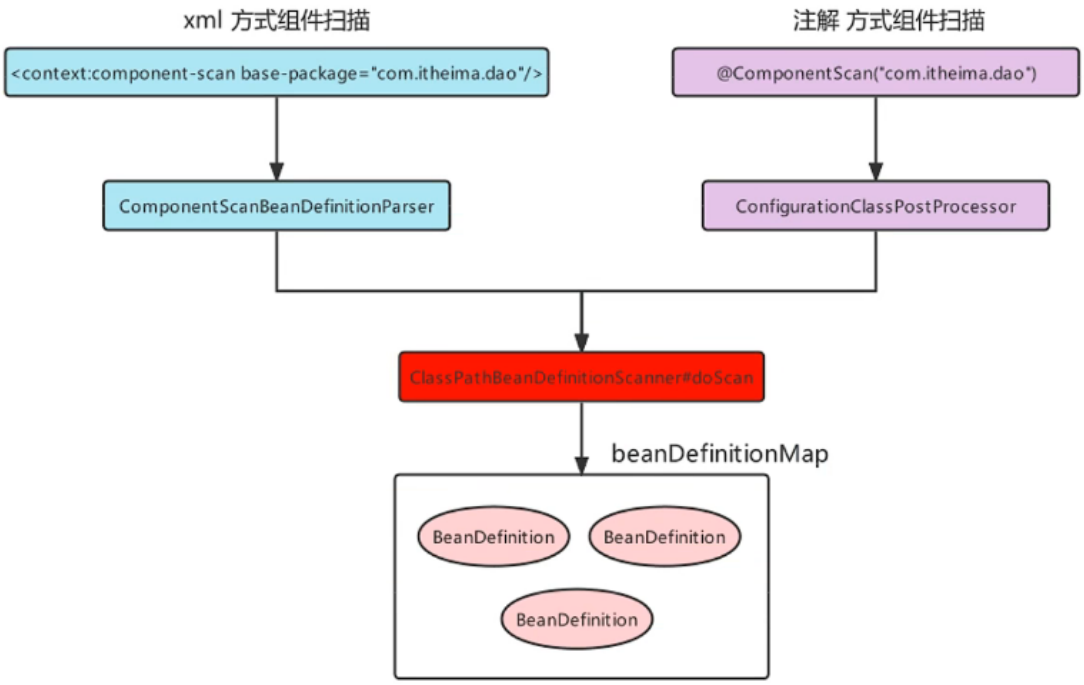
注解@Profile标注在类或方法上，标注当前的Bean从属于哪个环境，只有激活了当前环境，被标注的Bean才能被注册到Spring容器里，不指定环境的Bean，任何环境下都能注册到Spring容器里。

可以使用以下两种方式指定被激活的环境，

- 使用命令行动态参数，-Dspring.profiles.active=test
- 使用代码的方式设置环境变量 System.setProperty("spring.profiles.active", "test")

6. 注解解析原理

- Spring注解的解析原理



7. 整合第三方框架

以MyBatis为例，使用注解的方式替换掉原先的xml配置，xml配置如下，

```
<!--配置SqlSessionFactoryBean，提供SqlSessionFactory-->
<bean class="org.mybatis.spring.SqlSessionFactoryBean">
  <property name="dataSource" ref="dataSource"/>
</bean>

<!--MapperScannerConfigurer，扫描指定的包，产生Mapper对象-->
<bean class="org.mybatis.spring.mapper.MapperScannerConfigurer">
   <property name="basePackage" value="com.example.mapper"/>
</bean>

<!--配置数据源-->
<bean id="dataSource" class="com.alibaba.druid.pool.DruidDataSource">
   <property name="url" value="${jdbc.url}"/>
   <property name="username" value="${jdbc.username}"/>
   <property name="password" value="${jdbc.password}"/>
</bean>
```

使用注解完成配置SqlSessionFactoryBean，

```
@Bean
public SqlSessionFactoryBean sqlSessionFactoryBean(DataSource dataSource) {
  SqlSessionFactoryBean sqlSessionFactoryBean = new SqlSessionFactoryBean();
```

```
    sqlSessionFactoryBean.setDataSource(dataSource);
    return sqlSessionFactoryBean;
  }
```

使用注解完成配置MapperScannerConfigurer，在Spring配置类中添加注解@MapperScan，并指定扫描路径；

使用注解完成dataSource配置，

```
@Bean
public DataSource dataSource(
  @Value("${jdbc.url}") String url,
  @Value("${jdbc.username}") String username,
  @Value("${jdbc.password}") String password
) {
  DruidDataSource druidDataSource = new DruidDataSource();
  druidDataSource.setUrl(url);
  druidDataSource.setUsername(username);
  druidDataSource.setPassword(password);
  return druidDataSource;
}
```