JavaConfig 原来是 Spring 的一个子项目，它通过 Java 类的方式提供 Bean 的定义信息，在 Spring4 的版本， JavaConfig 已正式成为 Spring4 的核心功能 。
测试：

1.12. 基于java的容器配置

- 1.12.1.基本概念: @Bean 和 @Configuration
- 绑定Java与XML配置

```
1
2  /***
3   * @Author 徐庶   QQ:1092002729
4   * @Slogan 致敬大师，致敬未来的你
5   */
6  @Configuration    // 就相当于创建了一个xml 文件 <beans></beans>
7  @ComponentScan("cn.tulingxueyuan")   //<context:component-scan base-package="cn.tulingxue
8  @PropertySource("classpath:db.properties")
9  public class MainConfiration  {
10
11
12     @Value("${mysql.username}")
13     private String name;
14     @Value("${mysql.password}")
15     private String password;
16     @Value("${mysql.url}")
17     private String url;
18     @Value("${mysql.driverClassName}")
19     private String driverName;
20
21
22     // <bean class="com.alibaba.druid.pool.DruidDataSource" id="dataSource"></bean>
23     // 可以干预Bean实例化过程!
24     @Bean
25     public DruidDataSource dataSource(){
26         DruidDataSource dataSource=new DruidDataSource();
27         dataSource.setName(name);
28         dataSource.setPassword(password);
29         dataSource.setUrl(url);
30         dataSource.setDriverClassName(driverName);
31         return dataSource;
32
```

```
    }

    //init-method="initByConfig" destroy-method="destroyByConfig"
    @Bean(initMethod = "initByConfig",destroyMethod = "destroyByConfig")
    public User userconf(){

        return  new User();
    }
}
```

- **1.12.2.使用AnnotationConfigApplicationContext初始化Spring容器**
  - **简单结构**

```
@Test
public void test01(){
    ApplicationContext ioc=new AnnotationConfigApplicationContext(MainConfiration.class)
    UserController bean = ioc.getBean(UserController.class);
    bean.getUser();
}
```

- **1.12.3. @Bean 注解**
  - **声明一个bean**

> @Bean是一个方法级别的注解，它与XML中的 <bean/>元素类似。注解支持 <bean/>提供的一些属性，例如
> * init-method * destroy-method * autowiring * name
> 开发者可以在@Configuration类或@Component类中使用@Bean注解。

```
@Configuration
public class AppConfig {

    @Bean
    public TransferService transferService() {
        return new TransferServiceImpl();
    }
}
```

前面的配置完全等同于以下Spring XML:

```
<beans>
    <bean id="transferService" class="com.acme.TransferServiceImpl"/>
</beans>
```

Bean之间的依赖

- 我们可以使用方法参数来实现该依赖关系，如以下示例所示：

```
@Configuration
public class AppConfig {

    @Bean
    public TransferService transferService(AccountRepository accountRepository) {
        return new TransferServiceImpl(accountRepository);
    }
}
```

接受生命周期回调

```
@Bean(initMethod = "initByConfig",destroyMethod = "destroyByConfig")
public User userconf(){

    return  new User();
}
```

指定 Bean 的作用域

```
@Configuration
public class MyConfiguration {

    @Bean
    @Scope("prototype")
    public Encryptor encryptor() {
        // ...
    }
}
```

自定义Bean的名字

```
//默认情况下，配置类使用@Bean方法的名称作为结果bean的名称。
//但是，可以使用name属性覆盖此功能，如以下示例所示：
@Configuration
public class AppConfig {

    @Bean(name = "myThing")
    //多个别名：@Bean(name = { "dataSource", "subsystemA-dataSource", "subsystemB-dataSou
```

```
    public Thing thing() {
        return new Thing();
    }
}
```

- **1.12.4. @Configuration 注解**
  - **注入内部bean依赖**

```
//当Bean彼此有依赖关系时,表示依赖关系就像调用另一个bean方法一样简单.如下例所示:
@Configuration
public class AppConfig {

    @Bean
    public BeanOne beanOne() {
        return new BeanOne(beanTwo());
    }

    @Bean
    public BeanTwo beanTwo() {
        return new BeanTwo();
    }
}
```

- **1.12.5. 构成基于Java的配置**
  - **@Import 注解**

```
//就像在Spring XML文件中使用<import/>元素来帮助模块化配置一样,
//@Import 注解允许从另一个配置类加载@Bean定义，如下例所示:
@Configuration
public class ConfigA {

    @Bean
    public A a() {
        return new A();
    }
}

@Configuration
```

```java
@Import(ConfigA.class)
public class ConfigB {

    @Bean
    public B b() {
        return new B();
    }
}
```

将一个类注入到ioc中：

1.xml:<bean>

2.@Component (@Controller,@Service,@Repository)

3.@Bean

@Import