

Spring注解驱动开发第24讲——使用@Profile注解实现开发、测试和生产环境的配置和切换，看完这篇我彻底会了！！

写在前面

在实际的企业 **开发环境** 中，往往都会将环境分为开发环境、测试环境和生产环境，并且每个环境基本上都是互相隔离的，也就是说，开发环境、测试环境和生产环境它们之间是互不相通的。在以前的开发过程中，如果开发人员完成相应的功能模块并通过单元测试后，那么他会通过手动修改配置文件的形式，将项目的配置修改成测试环境，发布到测试环境中进行测试。测试通过后，再将配置修改为生产环境，发布到生产环境中。这样手动修改配置的方式，不仅增加了开发和运维的工作量，而且总是手工修改各项配置文件会很容易出问题。那么，有没有什么方式可以解决这个问题呢？答案是：有！通过@Profile注解就可以完全做到这点。

@Profile注解概述

在容器中如果存在同一类型的多个组件，那么可以使用@Profile注解标识要获取的是哪一个bean。也可以说@Profile注解是Spring为我们提供的可以根据当前环境，动态地激活和切换一系列组件的功能。这个功能在不同的环境使用不同的变量的情景下特别有用，例如，开发环境、**测试环境**、生产环境使用不同的数据源，在不改变代码的情况下，可以使用这个注解来动态地切换要连接的数据库。

接下来，我们来看下@Profile注解的源码，如下所示。

```
Profile.class
2+ * Copyright 2002-2017 the original author or authors.
16
17 package org.springframework.context.annotation;
18
19 import java.lang.annotation.Documented;
20 import java.lang.annotation.ElementType;
21 import java.lang.annotation.Retention;
22 import java.lang.annotation.RetentionPolicy;
23 import java.lang.annotation.Target;
24
25 import org.springframework.core.env.AbstractEnvironment;
26 import org.springframework.core.env.ConfigurableEnvironment;
27
29+ * Indicates that a component is eligible for registration when one or more
93 @Target({ElementType.TYPE, ElementType.METHOD})
94 @Retention(RetentionPolicy.RUNTIME)
95 @Documented
96 @Conditional(ProfileCondition.class)
97 public @interface Profile {
98
99     /**
100      * The set of profiles for which the annotated component should be registered.
101      */
102     String[] value();
103
104 }
```

从其源码中我们可以得出如下三点结论：

1. @Profile注解不仅可以标注在方法上，也可以标注在配置类上。
2. 如果@Profile注解标注在配置类上，那么只有是在指定的环境的时候，整个配置类里面的所有配置才会生效。
3. 如果一个bean上没有使用@Profile注解进行标注，那么这个bean在任何环境下都会被注册到IOC容器中，当然了，前提是在整个配置类生效的情况下。

第一点很容易看出，勿须再说，后面两点如果你要是初次认识@Profile注解的话，那么是肯定看不出来的，这得通过我下面的讲解才能知道。

实战案例

接下来，我们就一起来看一个案例，即使用@Profile注解实现开发、测试和生产环境的配置和切换。这里，我们以开发过程中要用到的数据源为例（数据源也是一种组件哟😊）。我们希望在开发环境中，数据源是连向A数据库的；在测试环境中，数据源是连向B数据库的，而且在这一过程中，测试人员压根就不需要改动任何代码；最终项目上线之后，数据源连向C数据库，而且最重要的一点是在整个过程中，我们不希望改动大量的代码，而实现数据源的切换。

环境搭建

首先，我们需要在pom.xml文件中添加c3p0数据源和MySQL驱动的依赖，如下所示。

```
1 <dependency>
2   <groupId>c3p0</groupId>
3   <artifactId>c3p0</artifactId>
4   <version>0.9.1.2</version>
5 </dependency>
6 <dependency>
7   <groupId>mysql</groupId>
8
```

```
9      <artifactId>mysql-connector-java</artifactId>
10      <version>5.1.44</version>
    </dependency>
    AI写代码xml
```



添加完以上依赖之后，我们还得在项目中新建一个配置类，例如MainConfigOfProfile，并在该配置类中模拟开发、测试、生产环境的数据源，如下所示。

```
1  package com.meimeixia.config;
2
3  import javax.sql.DataSource;
4
5  import org.springframework.context.annotation.Bean;
6  import org.springframework.context.annotation.Configuration;
7
8  import com.mchange.v2.c3p0.ComboPooledDataSource;
9
10 /**
11  *
12  * @author liayun
13  *
14  */
15 @Configuration
16 public class MainConfigOfProfile {
17
18     @Bean("testDataSource")
19     public DataSource dataSourceTest() throws Exception {
20         ComboPooledDataSource dataSource = new ComboPooledDataSource();
21         dataSource.setUser("root");
22         dataSource.setPassword("liayun");
23         dataSource.setJdbcUrl("jdbc:mysql://localhost:3306/test");
24         dataSource.setDriverClass("com.mysql.jdbc.Driver");
25         return dataSource;
26     }
27
28     @Bean("devDataSource")
29     public DataSource dataSourceDev() throws Exception {
30         ComboPooledDataSource dataSource = new ComboPooledDataSource();
31         dataSource.setUser("root");
32         dataSource.setPassword("liayun");
33         dataSource.setJdbcUrl("jdbc:mysql://localhost:3306/ssm_crud");
34         dataSource.setDriverClass("com.mysql.jdbc.Driver");
35         return dataSource;
36     }
37
38     @Bean("prodDataSource")
39     public DataSource dataSourceProd() throws Exception {
40         ComboPooledDataSource dataSource = new ComboPooledDataSource();
41         dataSource.setUser("root");
42         dataSource.setPassword("liayun");
43         dataSource.setJdbcUrl("jdbc:mysql://localhost:3306/scw_0515");
44         dataSource.setDriverClass("com.mysql.jdbc.Driver");
45         return dataSource;
46     }
47
48 }
    AI写代码java运行
```



该配置类这样写，是一点儿问题都没有的，但你有没有想过这一点，在真实项目开发中，那些数据库连接的相关信息，例如用户名、密码以及MySQL数据库驱动类的全名，这些都是要抽取在一个配置文件中的。你想想，是不是这么一回事啊！

因此，我们需要在项目的src/main/resources目录下新建一个配置文件，例如dbconfig.properties，在其中写上数据库连接的相关信息，如下所示。

```
1 db.user=root
2 db.password=liayun
3 db.driverClass=com.mysql.jdbc.Driver
    AI写代码xml
```

那么如何在MainConfigOfProfile配置类中获取以上配置文件中的值呢？这就需要我们大显神通了，看过我前面Spring注解驱动开发系列教程的童鞋们，相信这对你们并不难，而且正好可以复习一下前面所学习的知识点。

不过，我在这儿还是得说一点，该MainConfigOfProfile配置类实现了一个EmbeddedValueResolverAware接口，我们通过该接口能够获取到String值解析器。也就是说，IOC容器启动时会自动地将String值的解析器（即StringValueResolver）传递过来给我们用，咱们可以用它来解析一些字符串。

```

1 package com.meimeixia.config;
2
3 import javax.sql.DataSource;
4
5 import org.springframework.beans.factory.annotation.Value;
6 import org.springframework.context.EmbeddedValueResolverAware;
7 import org.springframework.context.annotation.Bean;
8 import org.springframework.context.annotation.Configuration;
9 import org.springframework.context.annotation.PropertySource;
10 import org.springframework.util.StringValueResolver;
11
12 import com.mchange.v2.c3p0.ComboPooledDataSource;
13
14 /**
15  *
16  * @author liayun
17  *
18  */
19 @PropertySource("classpath:/dbconfig.properties") // 加载外部的配置文件
20 @Configuration
21 public class MainConfigOfProfile implements EmbeddedValueResolverAware {
22
23     @Value("${db.user}")
24     private String user;
25
26     private StringValueResolver valueResolver;
27
28     private String dirverClass;
29
30     @Bean("testDataSource")
31     public DataSource dataSourceTest(@Value("${db.password}") String pwd) throws Exception {
32         ComboPooledDataSource dataSource = new ComboPooledDataSource();
33         dataSource.setUser(user);
34         dataSource.setPassword(pwd);
35         dataSource.setJdbcUrl("jdbc:mysql://localhost:3306/test");
36         dataSource.setDriverClass(dirverClass);
37         return dataSource;
38     }
39
40     @Bean("devDataSource")
41     public DataSource dataSourceDev(@Value("${db.password}") String pwd) throws Exception {
42         ComboPooledDataSource dataSource = new ComboPooledDataSource();
43         dataSource.setUser(user);
44         dataSource.setPassword(pwd);
45         dataSource.setJdbcUrl("jdbc:mysql://localhost:3306/ssm_crud");
46         dataSource.setDriverClass(dirverClass);
47         return dataSource;
48     }
49
50     @Bean("prodDataSource")
51     public DataSource dataSourceProd(@Value("${db.password}") String pwd) throws Exception {
52         ComboPooledDataSource dataSource = new ComboPooledDataSource();
53         dataSource.setUser(user);
54         dataSource.setPassword(pwd);
55         dataSource.setJdbcUrl("jdbc:mysql://localhost:3306/scw_0515");
56         dataSource.setDriverClass(dirverClass);
57         return dataSource;
58     }
59
60     @Override
61     public void setEmbeddedValueResolver(StringValueResolver resolver) {
62         this.valueResolver = resolver;
63         dirverClass = valueResolver.resolveStringValue("${db.driverClass}");
64     }
65 }
66

```

AI写代码java运行



其实，这个配置类相对来说还是比较简单的，其中使用 @Bean("devDataSource") 注解标注的是开发环境使用的数据源；使用 @Bean("testDataSource") 注解标注的是测试环境使用的数据源；使用 @Bean("prodDataSource") 注解标注的是生产环境使用的数据源。

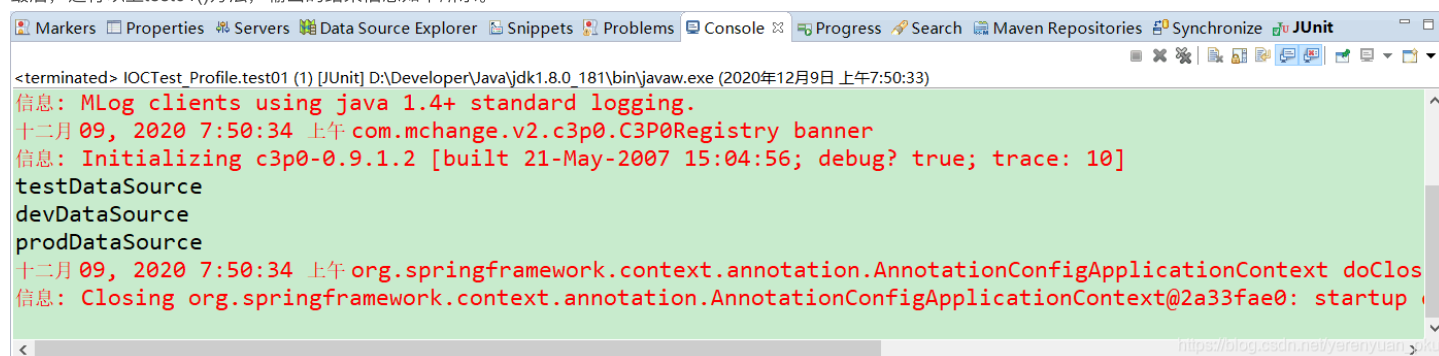
接着，我们创建一个 **单元测试** 类，例如IOCTest_Profile，并在该类中新建一个test01()方法来进行测试，如下所示。

```
1 package com.meimeixia.test;
2
3 import javax.sql.DataSource;
4
5 import org.junit.Test;
6 import org.springframework.context.annotation.AnnotationConfigApplicationContext;
7
8 import com.meimeixia.config.MainConfigOfProfile;
9
10 public class IOCTest_Profile {
11
12     @Test
13     public void test01() {
14         AnnotationConfigApplicationContext applicationContext = new AnnotationConfigApplicationContext(MainConfigOfProfile.class);
15
16         String[] namesForType = applicationContext.getBeanNamesForType(DataSource.class);
17         for (String name : namesForType) {
18             System.out.println(name);
19         }
20
21         // 关闭容器
22         applicationContext.close();
23     }
24 }
25 }
```

AI写代码java运行



最后，运行以上test01()方法，输出的结果信息如下所示。



```
<terminated> IOCTest_Profile.test01 (1) [JUnit] D:\Developer\Java\jdk1.8.0_181\bin\javaw.exe (2020年12月9日 上午7:50:33)
信息: MLog clients using java 1.4+ standard logging.
十二月 09, 2020 7:50:34 上午 com.mchange.v2.c3p0.C3P0Registry banner
信息: Initializing c3p0-0.9.1.2 [built 21-May-2007 15:04:56; debug? true; trace: 10]
testDataSource
devDataSource
prodDataSource
十二月 09, 2020 7:50:34 上午 org.springframework.context.annotation.AnnotationConfigApplicationContext doClose
信息: Closing org.springframework.context.annotation.AnnotationConfigApplicationContext@2a33fae0: startup...
```

可以看到三种不同的数据源成功注册到了IOC容器中，说明我们的环境搭建成功了。

根据环境注册bean

我们成功搭建环境之后，接下来，就是要实现根据不同的环境来向IOC容器中注册相应的bean了。也就是说，我们要实现在开发环境注册开发环境下使用的数据源；在测试环境注册测试环境下使用的数据源；在生产环境注册生产环境下使用的数据源。此时，@Profile注解就显示出其强大的特性了。

我们在MainConfigOfProfile配置类中为每个数据源添加@Profile注解标识，如下所示。

```
1 package com.meimeixia.config;
2
3 import javax.sql.DataSource;
4
5 import org.springframework.beans.factory.annotation.Value;
6 import org.springframework.context.EmbeddedValueResolverAware;
7 import org.springframework.context.annotation.Bean;
8 import org.springframework.context.annotation.Configuration;
9 import org.springframework.context.annotation.Profile;
10 import org.springframework.context.annotation.PropertySource;
11 import org.springframework.util.StringValueResolver;
12
13 import com.mchange.v2.c3p0.ComboPooledDataSource;
14
15 /**
16  *
17  * @author liayun
18  *
19  */
20 @PropertySource("classpath:/dbconfig.properties") // 加载外部的配置文件
21
```

```

22 @Configuration
23 public class MainConfigOfProfile implements EmbeddedValueResolverAware {
24
25     @Value("${db.user}")
26     private String user;
27
28     private StringValueResolver valueResolver;
29
30     private String dirverClass;
31
32     @Profile("test")
33     @Bean("testDataSource")
34     public DataSource dataSourceTest(@Value("${db.password}") String pwd) throws Exception {
35         ComboPooledDataSource dataSource = new ComboPooledDataSource();
36         dataSource.setUser(user);
37         dataSource.setPassword(pwd);
38         dataSource.setJdbcUrl("jdbc:mysql://localhost:3306/test");
39         dataSource.setDriverClass(dirverClass);
40         return dataSource;
41     }
42
43     @Profile("dev") // 定义了一个环境标识，只有当dev环境被激活以后，我们这个bean才能被注册进来
44     @Bean("devDataSource")
45     public DataSource dataSourceDev(@Value("${db.password}") String pwd) throws Exception {
46         ComboPooledDataSource dataSource = new ComboPooledDataSource();
47         dataSource.setUser(user);
48         dataSource.setPassword(pwd);
49         dataSource.setJdbcUrl("jdbc:mysql://localhost:3306/ssm_crud");
50         dataSource.setDriverClass(dirverClass);
51         return dataSource;
52     }
53
54     @Profile("prod")
55     @Bean("prodDataSource")
56     public DataSource dataSourceProd(@Value("${db.password}") String pwd) throws Exception {
57         ComboPooledDataSource dataSource = new ComboPooledDataSource();
58         dataSource.setUser(user);
59         dataSource.setPassword(pwd);
60         dataSource.setJdbcUrl("jdbc:mysql://localhost:3306/scw_0515");
61         dataSource.setDriverClass(dirverClass);
62         return dataSource;
63     }
64
65     @Override
66     public void setEmbeddedValueResolver(StringValueResolver resolver) {
67         this.valueResolver = resolver;
68         dirverClass = valueResolver.resolveStringValue("${db.driverClass}");
69     }
70 }

```

AI写代码java运行



可以看到，我们使用 `@Profile("dev")` 注解来标识在开发环境下注册devDataSource；使用 `@Profile("test")` 注解来标识在测试环境下注册testDataSource；使用 `@Profile("prod")` 注解来标识在生产环境下注册prodDataSource。

此时，我们运行IOCTest_Profile类中的test01()方法，发现Eclipse控制台并未输出任何结果信息。说明我们为不同的数据源添加@Profile注解后，默认是不会向IOC容器中注册bean的，需要我们根据环境显示指定向IOC容器中注册相应的bean。

换句话说，通过@Profile注解加了环境标识的bean，只有这个环境被激活的时候，相应的bean才会被注册到IOC容器中。

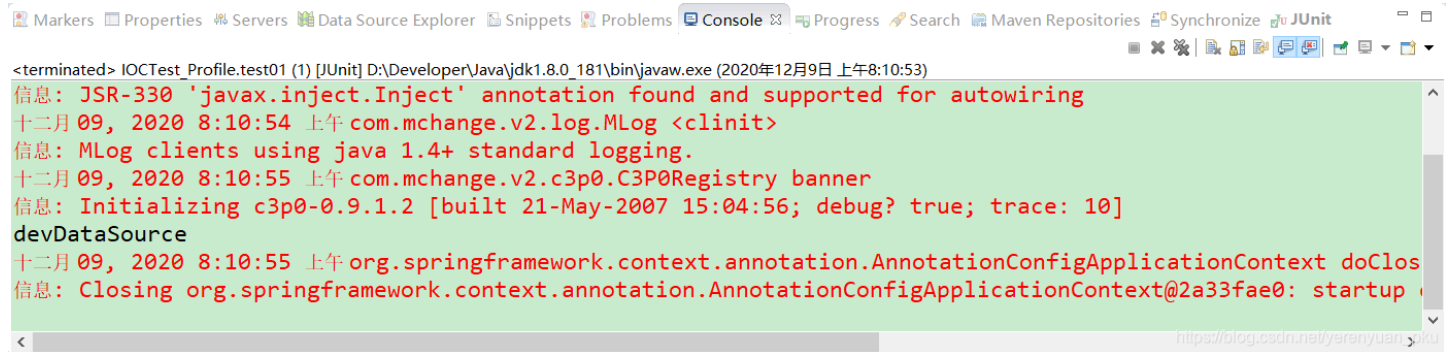
如果我们需要一个默认的环境，那么该怎么办呢？此时，我们可以通过 `@Profile("default")` 注解来标识一个默认的环境，例如，我们将devDataSource环境标识为默认环境，如下所示。

```

1  @Profile("default")
2  // @Profile("dev") // 定义了一个环境标识，只有当dev环境被激活以后，我们这个bean才能被注册进来
3  @Bean("devDataSource")
4  public DataSource dataSourceDev(@Value("${db.password}") String pwd) throws Exception {
5      ComboPooledDataSource dataSource = new ComboPooledDataSource();
6      dataSource.setUser(user);
7      dataSource.setPassword(pwd);
8      dataSource.setJdbcUrl("jdbc:mysql://localhost:3306/ssm_crud");
9      dataSource.setDriverClass(dirverClass);
10     return dataSource;
11 }

```

此时，我们运行IOCTest_Profile类中的test01()方法，输出的结果信息如下所示。



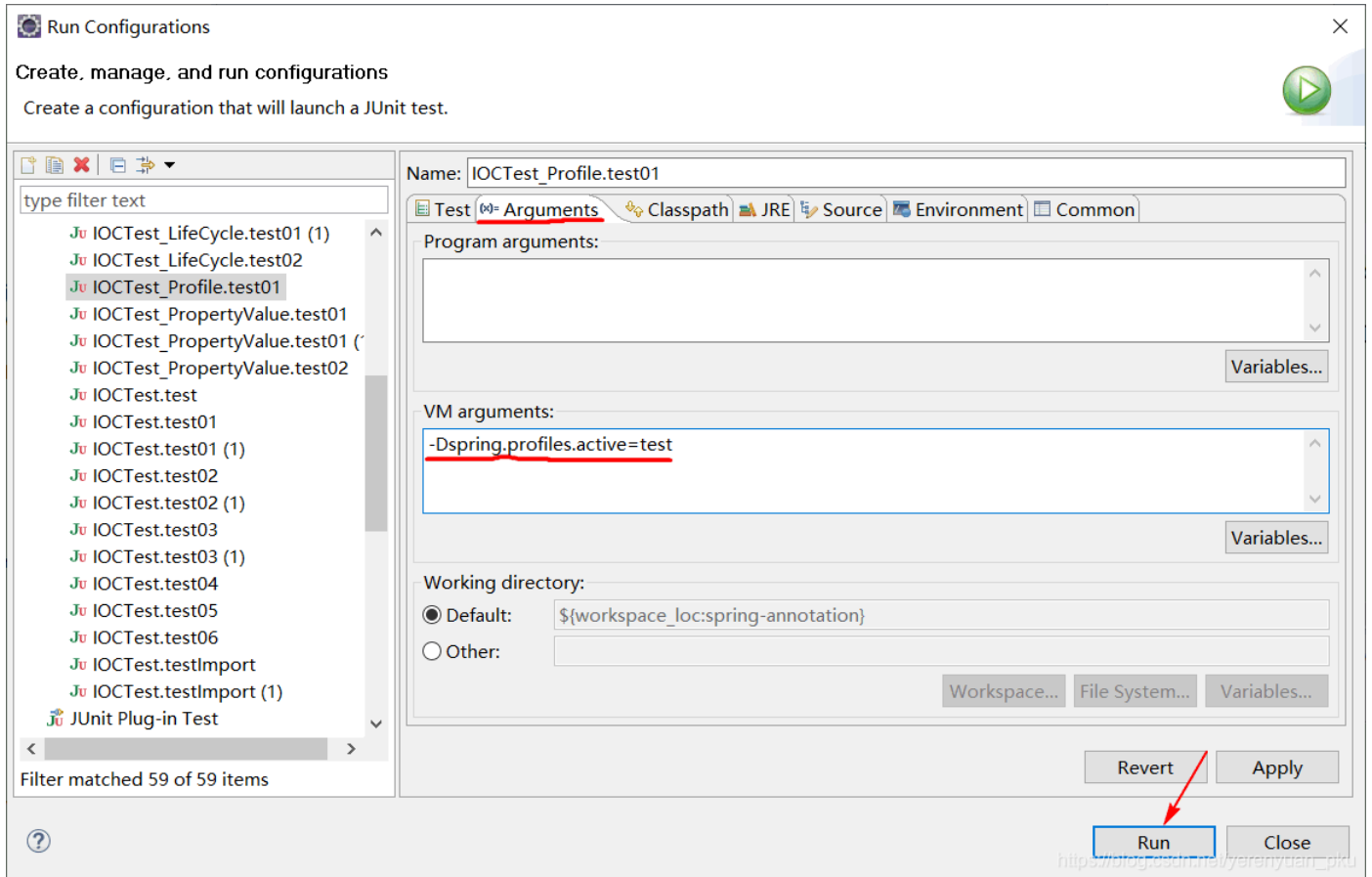
```
<terminated> IOCTest_Profile.test01 (1) [JUnit] D:\Developer\Java\jdk1.8.0_181\bin\javaw.exe (2020年12月9日 上午8:10:53)
信息: JSR-330 'javax.inject.Inject' annotation found and supported for autowiring
十二月 09, 2020 8:10:54 上午 com.mchange.v2.log.MLog <clinit>
信息: MLog clients using java 1.4+ standard logging.
十二月 09, 2020 8:10:55 上午 com.mchange.v2.c3p0.C3P0Registry banner
信息: Initializing c3p0-0.9.1.2 [built 21-May-2007 15:04:56; debug? true; trace: 10]
devDataSource
十二月 09, 2020 8:10:55 上午 org.springframework.context.annotation.AnnotationConfigApplicationContext doClose
信息: Closing org.springframework.context.annotation.AnnotationConfigApplicationContext@2a33fae0: startup
```

可以看到，我们在devDataSource数据源上使用 `@Profile("default")` 注解将其设置为默认的数据源，运行测试方法时Eclipse 控制台会输出devDataSource。

接下来，我们将devDataSource数据源上的 `@Profile("default")` 注解还原成 `@Profile("dev")` 注解，重新标识它为一个开发环境下注册的数据源，好方便下面的测试。

那么，我们如何根据不同的环境来注册相应的bean呢？例如，我们想在程序运行的时候，将其切换到测试环境下。

第一种方式就是根据命令行参数来确定环境，我们在运行程序的时候可以添加相应的命令行参数。例如，如果我们现在的环境是测试环境，那么可以在运行程序的时候添加如下命令行参数。



此时，点击 **Run** 按钮运行IOCTest_Profile类中的test01()方法，输出的结果信息如下所示。


```
<terminated> IOCTest_Profile.test01 [JUnit] D:\Developer\Java\jdk1.8.0_181\bin\javaw.exe (2020年12月9日 上午8:24:51)
信息: JSR-330 'javax.inject.Inject' annotation found and supported for autowiring
十二月 09, 2020 8:24:51 上午 com.mchange.v2.log.MLog <clinit>
信息: MLog clients using java 1.4+ standard logging.
十二月 09, 2020 8:24:52 上午 com.mchange.v2.c3p0.C3P0Registry banner
信息: Initializing c3p0-0.9.1.2 [built 21-May-2007 15:04:56; debug? true; trace: 10]
testDataSource
十二月 09, 2020 8:24:52 上午 org.springframework.context.annotation.AnnotationConfigApplicationContext doClose
信息: Closing org.springframework.context.annotation.AnnotationConfigApplicationContext@2a33fae0: startup...
```

第二种方式就是通过写代码的方式来激活某种环境，其实主要是通过AnnotationConfigApplicationContext类的无参构造方法来实现，具体步骤如下：

1. 在bean上加@Profile注解，其value属性值为环境标识，可以自定义
2. 使用AnnotationConfigApplicationContext类的无参构造方法创建容器
3. 设置容器环境，其值为第1步设置的环境标识
4. 设置容器的配置类
5. 刷新容器

温馨提示：2、4、5步其实是AnnotationConfigApplicationContext类中带参构造方法的步骤，以上这几个步骤相当于是把其带参构造方法拆开，在其中插入一条语句设置容器环境，这些我们可以在AnnotationConfigApplicationContext类的带参构造方法中看到，如下所示。

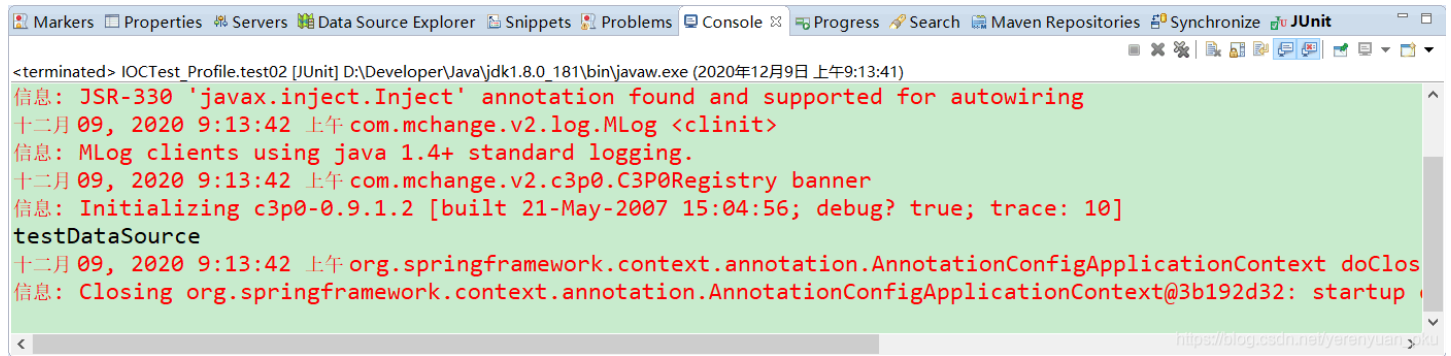
```
MainConfigOfProfile.java IOCTest_Profile.java AnnotationConfigApplicationContext.class
73 }
74
75 /**
76  * Create a new AnnotationConfigApplicationContext, deriving bean definitions
77  * from the given annotated classes and automatically refreshing the context.
78  * @param annotatedClasses one or more annotated classes,
79  * e.g. {@link Configuration @Configuration} classes
80  */
81 public AnnotationConfigApplicationContext(Class<?>... annotatedClasses) {
82     this();
83     register(annotatedClasses);
84     refresh();
85 }
86
87 /**
88  * Create a new AnnotationConfigApplicationContext, scanning for bean definitions
89  * in the given packages and automatically refreshing the context.
90  * @param basePackages the packages to check for annotated classes
91  */
```

好了，我们要开始正式编写代码来激活某种环境了。我们先在程序中调用AnnotationConfigApplicationContext类的无参构造方法来创建一个IOC容器，然后在容器进行初始化之前，为其设置相应的环境，接着再为容器设置主配置类，最后刷新一下容器。例如，我们将IOC容器设置为测试环境，如下所示。

```
1 @Test
2 public void test02() {
3     // 1. 使用无参构造器创建一个IOC容器
4     AnnotationConfigApplicationContext applicationContext = new AnnotationConfigApplicationContext();
5     // 2. 在我们容器还没启动创建其他bean之前，先设置需要激活的环境（可以设置激活多个环境哟）
6     applicationContext.getEnvironment().setActiveProfiles("test");
7     // 3. 注册主配置类
8     applicationContext.register(MainConfigOfProfile.class);
9     // 4. 启动刷新容器
10    applicationContext.refresh();
11
12    String[] namesForType = applicationContext.getBeanNamesForType(DataSource.class);
13    for (String name : namesForType) {
14        System.out.println(name);
15    }
16
17    // 关闭容器
18    applicationContext.close();
19 }
```

AI写代码java运行

此时，我们运行以上test02()方法，输出的结果信息如下所示。



```
<terminated> IOCTest_Profile.test02 [JUnit] D:\Developer\Java\jdk1.8.0_181\bin\javaw.exe (2020年12月9日 上午9:13:41)
信息: JSR-330 'javax.inject.Inject' annotation found and supported for autowiring
十二月 09, 2020 9:13:42 上午 com.mchange.v2.log.MLog <clinit>
信息: MLog clients using java 1.4+ standard logging.
十二月 09, 2020 9:13:42 上午 com.mchange.v2.c3p0.C3P0Registry banner
信息: Initializing c3p0-0.9.1.2 [built 21-May-2007 15:04:56; debug? true; trace: 10]
testDataSource
十二月 09, 2020 9:13:42 上午 org.springframework.context.annotation.AnnotationConfigApplicationContext doClose
信息: Closing org.springframework.context.annotation.AnnotationConfigApplicationContext@3b192d32: startup...
```

可以看到，Eclipse控制台输出了testDataSource，说明我们成功将IOC容器的环境设置为了测试环境。

如果此时测试环境里面还有一些其他的组件，比如Yellow，

```
1 @Profile("test")
2 @Bean
3 public Yellow yellow() {
4     return new Yellow();
5 }
```

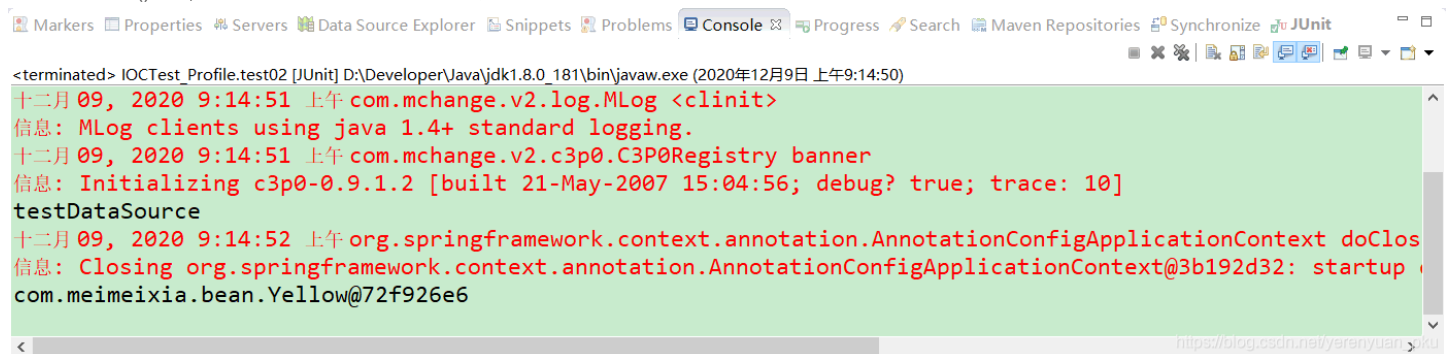
AI写代码java运行

那么在测试环境被激活的情况下，测试环境下的所有bean都会被注册到IOC容器中。如果你要是不信的话，那么你可以试着修改一下IOCTest_Profile类中的test02()方法，即在其中获取Yellow组件并打印看看。

```
1 @Test
2 public void test02() {
3     // 1. 使用无参构造器创建一个IOC容器
4     AnnotationConfigApplicationContext applicationContext = new AnnotationConfigApplicationContext();
5     // 2. 在我们容器还没启动创建其他bean之前，先设置需要激活的环境（可以设置激活多个环境哟）
6     applicationContext.getEnvironment().setActiveProfiles("test");
7     // 3. 注册主配置类
8     applicationContext.register(MainConfigOfProfile.class);
9     // 4. 启动刷新容器
10    applicationContext.refresh();
11
12    String[] namesForType = applicationContext.getBeanNamesForType(DataSource.class);
13    for (String name : namesForType) {
14        System.out.println(name);
15    }
16
17    Yellow yellow = applicationContext.getBean(Yellow.class);
18    System.out.println(yellow);
19
20    // 关闭容器
21    applicationContext.close();
22 }
```

AI写代码java运行

运行以上test02()方法，你将会看到如下所示的结果信息。



```
<terminated> IOCTest_Profile.test02 [JUnit] D:\Developer\Java\jdk1.8.0_181\bin\javaw.exe (2020年12月9日 上午9:14:50)
十二月 09, 2020 9:14:51 上午 com.mchange.v2.log.MLog <clinit>
信息: MLog clients using java 1.4+ standard logging.
十二月 09, 2020 9:14:51 上午 com.mchange.v2.c3p0.C3P0Registry banner
信息: Initializing c3p0-0.9.1.2 [built 21-May-2007 15:04:56; debug? true; trace: 10]
testDataSource
十二月 09, 2020 9:14:52 上午 org.springframework.context.annotation.AnnotationConfigApplicationContext doClose
信息: Closing org.springframework.context.annotation.AnnotationConfigApplicationContext@3b192d32: startup...
com.meimeixia.bean.Yellow@72f926e6
```

这佐证了如果测试环境被激活，那么测试环境下的所有bean都会被注册到IOC容器中的这一结论。

@Profile注解不仅可以标注在方法上，也可以标注在配置类上。如果标注在配置类上，那么只有是在指定的环境的时候，整个配置类里面的所有配置才会生效。例如，我们在MainConfigOfProfile配置类上标注上@Profile("dev")注解，如下所示。


```
1 | @Profile("dev") // @Profile注解除了能写到bean上，还能写到类上
2 | @PropertySource("classpath:/dbconfig.properties") // 加载外部的配置文件
3 | @Configuration
4 | public class MainConfigOfProfile implements EmbeddedValueResolverAware {
5 |
6 |     /*****代码省略*****/
7 |
8 | }
AI写代码java运行
```

然后，我们来运行IOCTest_Profile类中的test02()方法，在运行该方法之前，记得要把获取Yellow组件并打印的两行代码给注释掉，要不然运行test02()方法之后，Eclipse控制台就会报错。这时，咱们再来运行test02()方法，会发现Eclipse控制台中并未输出任何信息。

这是因为我们在test02()方法中指定了当前的环境为测试环境，而MainConfigOfProfile配置类上标注的注解为@Profile("dev")，说明该配置类中的所有配置只有在开发环境下才会生效。所以，此时没有任何数据源注册到IOC容器中，自然Eclipse控制台中就不会输出任何信息了。

还记得我在一开头就说过，如果一个bean上没有使用@Profile注解进行标注，那么这个bean在任何环境下都会被注册到IOC容器中吗？现在咱们就来验证这一点。

首先，我们要将MainConfigOfProfile配置类上标注的@Profile("dev")注解给注释掉，好方便接下来的测试。

然后，再将Yellow组件上的@Profile("test")注解给注释掉，此时，MainConfigOfProfile配置类的完整代码如下所示。

```
1 | package com.meimeixia.config;
2 |
3 | import javax.sql.DataSource;
4 |
5 | import org.springframework.beans.factory.annotation.Value;
6 | import org.springframework.context.EmbeddedValueResolverAware;
7 | import org.springframework.context.annotation.Bean;
8 | import org.springframework.context.annotation.Configuration;
9 | import org.springframework.context.annotation.Profile;
10 | import org.springframework.context.annotation.PropertySource;
11 | import org.springframework.util.StringValueResolver;
12 |
13 | import com.mchange.v2.c3p0.ComboPooledDataSource;
14 | import com.meimeixia.bean.Yellow;
15 |
16 | /**
17 |  *
18 |  * @author liayun
19 |  *
20 |  */
21 | // @Profile("dev") // @Profile注解除了能写到bean上，还能写到类上
22 | @PropertySource("classpath:/dbconfig.properties") // 加载外部的配置文件
23 | @Configuration
24 | public class MainConfigOfProfile implements EmbeddedValueResolverAware {
25 |
26 |     @Value("${db.user}")
27 |     private String user;
28 |
29 |     private StringValueResolver valueResolver;
30 |
31 |     private String dirverClass;
32 |
33 |     // @Profile("test")
34 |     @Bean
35 |     public Yellow yellow() {
36 |         return new Yellow();
37 |     }
38 |
39 |     @Profile("test")
40 |     @Bean("testDataSource")
41 |     public DataSource dataSourceTest(@Value("${db.password}") String pwd) throws Exception {
42 |         ComboPooledDataSource dataSource = new ComboPooledDataSource();
43 |         dataSource.setUser(user);
44 |         dataSource.setPassword(pwd);
45 |         dataSource.setJdbcUrl("jdbc:mysql://localhost:3306/test");
46 |         dataSource.setDriverClass(dirverClass);
47 |         return dataSource;
48 |     }
49 |
50 |     // @Profile("default")
51 |     @Profile("dev") // 定义了一个环境标识，只有当dev环境被激活以后，我们这个bean才能被注册进来
52 |     @Bean("devDataSource")
53 |     public DataSource dataSourceDev(@Value("${db.password}") String pwd) throws Exception {
54 |         ComboPooledDataSource dataSource = new ComboPooledDataSource();
55 |         dataSource.setUser(user);
56 |         dataSource.setPassword(pwd);
57 |     }
58 | }
```

```

57     dataSource.setJdbcUrl("jdbc:mysql://localhost:3306/ssm_crud");
58     dataSource.setDriverClass(dirverClass);
59     return dataSource;
60 }
61
62 @Profile("prod")
63 @Bean("prodDataSource")
64 public DataSource dataSourceProd(@Value("${db.password}") String pwd) throws Exception {
65     ComboPooledDataSource dataSource = new ComboPooledDataSource();
66     dataSource.setUser(user);
67     dataSource.setPassword(pwd);
68     dataSource.setJdbcUrl("jdbc:mysql://localhost:3306/scw_0515");
69     dataSource.setDriverClass(dirverClass);
70     return dataSource;
71 }
72
73 @Override
74 public void setEmbeddedValueResolver(StringValueResolver resolver) {
75     this.valueResolver = resolver;
76     dirverClass = valueResolver.resolveStringValue("${db.driverClass}");
77 }
78 }
79 }

```

AI写代码java运行

接着，修改一下IOCTest_Profile类中的test02()方法，即放开获取Yellow组件并打印的两行代码。

```

1  @Test
2  public void test02() {
3      // 1. 使用无参构造器创建一个IOC容器
4      AnnotationConfigApplicationContext applicationContext = new AnnotationConfigApplicationContext();
5      // 2. 在我们容器还没启动创建其他bean之前，先设置需要激活的环境（可以设置激活多个环境哟）
6      applicationContext.getEnvironment().setActiveProfiles("dev");
7      // 3. 注册主配置类
8      applicationContext.register(MainConfigOfProfile.class);
9      // 4. 启动刷新容器
10     applicationContext.refresh();
11
12     String[] namesForType = applicationContext.getBeanNamesForType(DataSource.class);
13     for (String name : namesForType) {
14         System.out.println(name);
15     }
16
17     Yellow yellow = applicationContext.getBean(Yellow.class);
18     System.out.println(yellow);
19
20     // 关闭容器
21     applicationContext.close();
22 }

```

AI写代码java运行

可以看到，当前的环境指定为了开发环境，那么此时Yellow这个组件会被注册到IOC容器中吗？

紧接着，运行IOCTest_Profile类中的test02()方法，输出的结果信息如下所示。

```

<terminated> IOCTest_Profile.test02 [JUnit] D:\Developer\Java\jdk1.8.0_181\bin\javaw.exe (2020年12月9日 上午9:34:34)
十二月 09, 2020 9:34:35 上午 com.mchange.v2.log.MLog <clinit>
信息: MLog clients using java 1.4+ standard logging.
十二月 09, 2020 9:34:35 上午 com.mchange.v2.c3p0.C3P0Registry banner
信息: Initializing c3p0-0.9.1.2 [built 21-May-2007 15:04:56; debug? true; trace: 10]
devDataSource
十二月 09, 2020 9:34:35 上午 org.springframework.context.annotation.AnnotationConfigApplicationContext doClose
信息: Closing org.springframework.context.annotation.AnnotationConfigApplicationContext@3b192d32: startup
com.meimeixia.bean.Yellow@72f926e6

```

从以上输出结果中可以看到，Yellow组件上并没有使用@Profile注解进行标注，但是它在开发环境下被注册到IOC容器中了。

如果此时将当前的环境指定为生产环境，那么Yellow这个组件还会被注册到IOC容器中吗？

```
1 @Test
2 public void test02() {
3     // 1. 使用无参构造器创建一个IOC容器
4     AnnotationConfigApplicationContext applicationContext = new AnnotationConfigApplicationContext();
5     // 2. 在我们容器还没启动创建其他bean之前，先设置需要激活的环境（可以设置激活多个环境哟）
6     applicationContext.getEnvironment().setActiveProfiles("prod");
7     // 3. 注册主配置类
8     applicationContext.register(MainConfigOfProfile.class);
9     // 4. 启动刷新容器
10    applicationContext.refresh();
11
12    String[] namesForType = applicationContext.getBeanNamesForType(DataSource.class);
13    for (String name : namesForType) {
14        System.out.println(name);
15    }
16
17    Yellow yellow = applicationContext.getBean(Yellow.class);
18    System.out.println(yellow);
19
20    // 关闭容器
21    applicationContext.close();
22 }
```

AI写代码java运行



运行以上test02()方法，发现输出的结果信息如下所示。

```
<terminated> IOCTest_Profile.test02 [JUnit] D:\Developer\Java\jdk1.8.0_181\bin\javaw.exe (2020年12月9日 上午9:45:36)
十二月 09, 2020 9:45:36 上午 com.mchange.v2.log.MLog <clinit>
信息: MLog clients using java 1.4+ standard logging.
十二月 09, 2020 9:45:37 上午 com.mchange.v2.c3p0.C3P0Registry banner
信息: Initializing c3p0-0.9.1.2 [built 21-May-2007 15:04:56; debug? true; trace: 10]
prodDataSource
十二月 09, 2020 9:45:37 上午 org.springframework.context.annotation.AnnotationConfigApplicationContext doClose
信息: Closing org.springframework.context.annotation.AnnotationConfigApplicationContext@3b192d32: startup
com.meimeixia.bean.Yellow@72f926e6
```

这进一步说明了，虽然Yellow组件上并没有使用@Profile注解进行标注，但是它也在生产环境下被注册到IOC容器中。

至此，如果一个bean上没有使用@Profile注解进行标注，那么这个bean在任何环境下都会被注册到IOC容器中这一结论就得到完美证明了。