我可以这么告诉大家，我在见过的所有的maven视频课程里，从没见过任何一个人讲解这块内容，profile适配各个发布环境，但是实际上，在公司里工作的时候，如果你的项目没有用profile去适配各个发布环境，那我只能说，你的项目弄的有点low

因为如果你不这么做的话，就可能导致你对每个环境的发布，都要手工去修改各种配置文件

1、公司里的多个环境

一般我们的项目都有多个环境，常见的是这么几个环境

dev：即本地的开发和测试环境，这个就是指的是我们自己本地的笔记本电脑，在上面可以进行开发、单元测试、冒烟测试。比如说你的local本地环境需要有一些基础性的依赖设施，比如说数据库，比如说MQ，比如说redis。这些东西，mysql、rabbit mq、redis，都是部署在公司的一套公共的一个dev环境，有一套服务器，上面会部署各种项目组成员在本地开发，需要的各种依赖设施。保证你在本地开发的时候，指定对应的地址，都是可以连通，笔记本电脑连通的。

beta：beta，一般称作内部测试环境，也就是不对外。集成测试/联调测试的环境，一般自己感觉开发好了之后，就需要将自己的系统部署到一个集成测试环境，有的公司也叫做联调测试环境，说白了，就是一个项目都是多个研发人员搞的，每个人弄好了自己的部分，都要发布到一个环境，大家在上面测一测整个系统多个服务能不能串起来跑的通，至少别报错把。这一块主要是确保主流程要跑通，不要报错。

test：QA测试环境，正经项目，都会有专门的测试人员，如果没有的话，那研发人员只能自己去充当QA角色了。就是需要将通过集成测试的代码，部署到QA测试环境，然后由QA人员进行非常充分而且完善的测试，验证功能，性能，等各个方面都没有问题

staging：预发布环境，通常这个环境会跟生产环境保持基本一致，部署到上面之后，就会使用部分真实的流量或者数据，来让系统运行。比如对外提供服务的网站，app之类的，可以通过流量拷贝的方式，拷贝一小部分流量过来，在staging环境让开发好的系统跑一跑。后者也可以拷贝部分真实的线上数据库的数据下来，跑一跑。然后这个环节QA还是会介入，再次验证一下看系统是否运行正常。同时这个环节，有一个验收的作用，项目如果有产品经理，此时会在这个环节看一下是否符合他的产品预期。

prod：生产环境，最终系统部署到线上生产环境中，完成上线

每个环境，都是完全隔离开来的，都有自己的数据库、mq、缓存等等各种各样的依赖，比如数据库把，使用的就肯定也是不同的数据库

dev环境下，通常是用的开发人员自己本地安装的一个数据库，或者是用公司测试环境中专门用于开发的一个公共测试数据库；dev环境下，有一个专门的集成测试数据库；test环境下，有一个专门的QA测试数据库；staging环境下，有一个专门的预发布环境数据库；prod环境下，就是线上的生产环境数据库

每个环境的数据库的各种配置都是不一样的

如果我们按照现在这种模式去做项目，是很low，无法直视。相当不靠谱，因为这个是依靠人力去大量的重复的反复的修改，很可能哪一次就弄错了配置。比如说，你本来应该上test环境，数据库+mq+缓存，都是应该test环境的地址；结果你不小心忘了配置，就导致带着beta环境的地址上了test环境，结果导致系统各种报错，耗费1天的时间反复排查。

还有更严重的情况，比如说带着staging环境的地址，就直接上prod环境，开除

因此我们的项目，需要一种能力，可以在不同的环境发布的时候，各种资源的配置自动适配各个环境，而不是每次发布一个环境之前，手动去修改那个配置

我们需要在项目里，给每一套环境，都配置好一套预先定好的地址，而不是每次部署的时候去修改一个文件，接着在每次往一个环境去部署的时候，可以手动指定一个参数，那么那个参数会指定用哪个环境下的一套配置

那么可能有的同学会说，是，不用每次部署都修改几十个配置了，但是还是要指定一个环境参数，会不会指定错呢？

也有这种可能，但是问题在于说，已经概率小的多了，本来你每次部署都要修改几十个配置，现在的话就是指定一个环境参数就可以了

另外一个，还可以跟大家普及一下，大公司里面，绝对都是对工程开发流程与发布，等等，一整套东西，都是公司自己研发的软件系统。比如说，可能你在各个环境部署的时候，就是只要保证代码仓库里是正确的代码，接着在界面上直接点击要部署到哪个环境就可以了，此时就会将代码部署到那个环境，同时采用正确的环境参数，激活一套配置

这个的好处，在于说，如果你本来是要上线，部署prod环境；不小心在页面上点错了，点了部署staging环境，也没关系，他其实就会给你部署到staging环境，而且是用staging环境的一套配置，跟prod环境没关系

2、基于资源过滤+profile的方式适配各个环境

比如src/main/resources下，有一个jdbc.properties配置文件

在local环境下，应该是这样的：

database.jdbc.driverClass=com.mysql.jdbc.Driver

database.jdbc.connectionURL=jdbc:mysql://192.168.31.110:3306/oa\_local

database.jdbc.username=local

database.jdbc.password=local

在dev环境下，应该是这样的：

database.jdbc.driverClass=com.mysql.jdbc.Driver

database.jdbc.connectionURL=jdbc:mysql://192.168.31.110:3306/oa\_dev

database.jdbc.username=dev

database.jdbc.password=dev

此时，首先，我们需要将配置对应的值从配置文件里抽取出来，用占位符替代，然后实际的值放到profile里去

database.jdbc.driverClass=${database.jdbc.driverClass}

database.jdbc.connectionURL=${database.jdbc.connectionURL}

database.jdbc.username=${database.jdbc.username}

database.jdbc.password=${database.jdbc.password}

然后在pom.xml里加入各个环境对应的profile配置

<profiles>

<profile>

<id>local</id>

<properties>

<database.jdbc.driverClass>com.mysql.jdbc.Driver</database.jdbc.driverClass>

<database.jdbc.connectionURL>jdbc:mysql://192.168.31.110:3306/oa\_local</database.jdbc.connectionURL>

<database.jdbc.username>local</database.jdbc.username>

<database.jdbc.password>local</database.jdbc.password>

</properties>

</profile>

<profile>

<id>dev</id>

<properties>

<database.jdbc.driverClass>com.mysql.jdbc.Driver</database.jdbc.driverClass>

<database.jdbc.connectionURL>jdbc:mysql://192.168.31.110:3306/oa\_dev</database.jdbc.connectionURL>

<database.jdbc.username>dev</database.jdbc.username>

<database.jdbc.password>dev</database.jdbc.password>

</properties>

</profile>

</profiles>

这个时候大家会发现，太好了，配置文件里的值都是占位符，每个环境的实际值都到了对应的profile里去了

但是此时就是，如何让项目在发布到不同环境的时候激活profile，并且将实际值替换到占位符里去呢？

为src/main/resources目录开启资源过滤功能，让maven resources插件在处理资源的时候自动去解析里面的占位符，然后找到对应profile里的实际值来进行替换

<resources>

<resource>

<directory>${project.basedir}/src/main/resources</directory>

<filtering>true</filtering>

</resource>

</resources>

<testResources>

<testResource>

<directory>${project.basedir}/src/test/resources</directory>

<filtering>true</filtering>

</testResource>

</testResources>

src/main/profiles

<resources>

<resource>

<directory>src/main/profiles/local</directory>

</resource>

</resources>

mvn clean package -Pdev，-P就是说激活dev profile

查看target/classes下面的资源文件，全都替换为了dev profile中的实际值

profile有很多种激活的方式，但是常用的其实就是-P这一种激活方式即可，因为都是部署的时候用不同的-P来激活

3、默认激活

我们一般会设置默认激活一个profile

在profile配置里加一个：

<activation>

<activeByDefault>true</activeByDefault>

</activation>

然后在各个环境部署的时候，肯定会用哪个环境的-P来激活对应的profile配置进行占位符替换

而且还可以用mvn help:active-profiles查看当前激活的是哪个profile

4、不同地方配置profile

profile实际上是可以在不同的地方配置的

比如说在settings.xml里，我们也通过配置profile以及激活一些profile配置了maven私服仓库地址，在maven的超级pom里，也有一些profile激活

5、实验步骤

（1）将jdbc.properties中的属性值都修改为占位符形式

（2）在pom.xml里面，放入profiles，各个环境对应一个profile，在profile里面定义properties

（3）在<build>的<resource>里面，开启src/main/resources目录的资源过滤

（4）在打包的时候，给一个环境参数，激活你要部署的那个环境的profile，同时maven会对src/main/resources下面的文件进行资源过滤，看有没有占位符，如果有，就用profile下面的properties实际属性值去替换

（5）打包实验，给不同的环境参数

mvn clean process-resources，不加任何参数，会发现，jdbc.properties里面都替换成了dev profile默认激活的值

mvn clean process-resources -Pbeta，会激活beta profile，此时，直接到eclipse的eclipse-workspace里面去找，target/classes下面，有处理好占位符的资源文件，此时可以看到，jdbc.properties里面的值都替换成了beta profile下的

（6）进一步，在大型的工程里面，配置文件特别多，你不可能把所有的配置都放profiles里面，那个有点很乱，一般是怎么做呢？src/main/resources下面直接就不放东西了，实际上是在profile里面定义一个单独属于自己的环境的一套配置文件对应的目录，然后在处理资源文件的时候，会把激活的profile对应的一套目录里的资源文件拷贝到src/main/resources下面去，给工程打包使用

再建多个目录：

src/main/profiles/dev

src/main/profiles/beta

src/main/profiles/test

src/main/profiles/staging

src/main/profiles/prod

每个目录下，都有自己的一套完整的配置文件

因为实际上来说，对于不同的环境，不仅仅只是一些变量值等等，可能对于一些框架本身的设置都不太一样，测试环境，可能就会把一些参数调整的低一些，生产环境就会很高。测试环境下，有些框架就不需要配置，生产环境下需要配置。

给每个profile配置一下这个要拷贝到打包目录里面去的resource即可：

<build>

<resources>

<resource>

<directory>src/main/profiles/dev</directory>

<includes>

<include>\*\*/\*.xml</include>

<include>\*\*/\*.properties</include>

</includes>

<filtering>true</filtering>

</resource>

</resources>

</build>

再次用mvn clean process-resources -Pdev激活各个profile，看一下classes里面是什么？

我可以给大家说一个实际的场景：

第一步，oa-organ，oa-auth，oa-flow，三个模块的负责人，此时用的snapshot版本，mvn clean deploy -Pbeta，此时就会用beta环境的东西去打包和部署到私服，给别人去用

第二步，oa-web，要部署beta的tomcat，此时依赖的还是各个工程的snapshot版本，此时各个工程的最新的snapshot版本对应的Jar包里面就都是beta环境的配置文件了

第三步，oa-web，cargo发布到哪个环境的配置，也是可以用profiles下面的properties来进行占位符替换的，每个profile定义一个properties，里面包含自己的环境对应的tomcat地址，然后在cargo插件的配置里面，用占位符来引用即可。

第四步，oa-web，打一个war包，此时也是mvn clean package -Pbeta，用beta环境的配置打成war包，而且此时用的都是另外三个模块的beta环境下的jar包，打在他自己的war包里面

第五步，oa-web，用cargo，mvn cargo:redeploy -Pbeta，此时就可以进行beta环境的部署

第六步，大家在各个环境如法炮制全都测试完了，此时呢就会上线。各个依赖模块，用mvn clean deploy -Pprod，去打一个prod环境的jar包，部署到私服；oa-web，打出来一个war包，此时打包的时候，用的都是各个依赖最新的prod环境的jar包，接着就使用mvn cargo:redeploy -Pprod，此时就会发布到prod环境对应的tomcat地址上去。

在各个公司，可能用法不太一样，但是maven profile肯定是要用的，因为无论你怎么部署，肯定是涉及多个环境的，不同环境的资源文件放一个profile对应的目录下，然后各种部署或者打包的时候，使用对应的profile的参数来激活即可。853769620