# 整合

## [Spring+Mybatis+SpringMVC+Maven+MySql搭建实例](https://www.cnblogs.com/GarfieldEr007/p/5746263.html)

<https://www.cnblogs.com/GarfieldEr007/p/5746263.html>

# spring与mybatis四种整合方法

<https://blog.csdn.net/xqhys/article/details/53994740>

# Spring+MyBatis双数据库配置

<https://blog.csdn.net/wp1603710463/article/details/50884259>

# [Spring下redis的配置](https://www.cnblogs.com/cuglkb/p/6862609.html)

<https://www.cnblogs.com/cuglkb/p/6862609.html>

# redis多个数据源配置

<https://blog.csdn.net/happy_wu/article/details/73603743>

# Spring集成Redis缓存实现

<https://blog.csdn.net/try_harder_every_day/article/details/78757324>

# Spring配置Mongodb（多个数据库的配置说明）

<http://blog.sina.com.cn/s/blog_b94ee5340102w5qv.html>

# [Spring 一主多从、多主多从 数据库配置](https://www.cnblogs.com/mafly/p/moreMaster.html)

<https://www.cnblogs.com/mafly/p/moreMaster.html>

# Spring中配置DataSource数据源的几种选择

<https://blog.csdn.net/u013490585/article/details/61201840>

# [Spring-Mybatis --- 配置SqlSessionFactoryBean，整合Spring-Mybatis](https://www.cnblogs.com/ClassNotFoundException/p/6425558.html)

<https://www.cnblogs.com/ClassNotFoundException/p/6425558.html>

# [Spring 在xml配置里配置事务](https://www.cnblogs.com/hq233/p/6664798.html)

<https://www.cnblogs.com/hq233/p/6664798.html>

# [spring配置日志](https://www.cnblogs.com/chen-lhx/p/6170298.html)

<https://www.cnblogs.com/chen-lhx/p/6170298.html>

# Spring AOP实现复杂的日志记录（自定义注解）

<https://blog.csdn.net/mlc1218559742/article/details/51778224>

# [spring AOP自定义注解方式实现日志管理](https://www.cnblogs.com/jianjianyang/p/4910851.html)

<https://www.cnblogs.com/jianjianyang/p/4910851.html>

# java实现分布式项目搭建

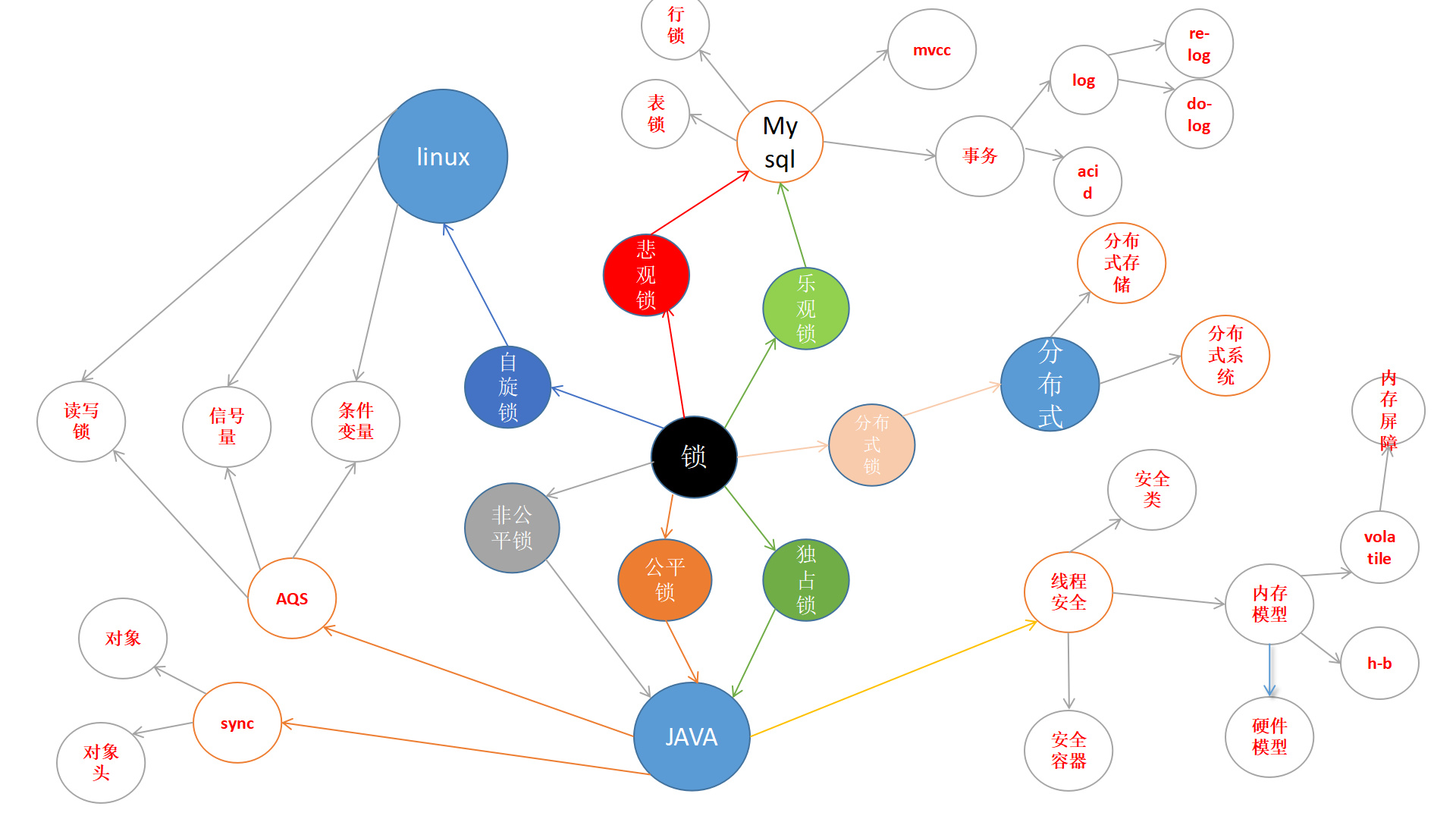
<https://juejin.im/post/5ad85368f265da50472fc95a>

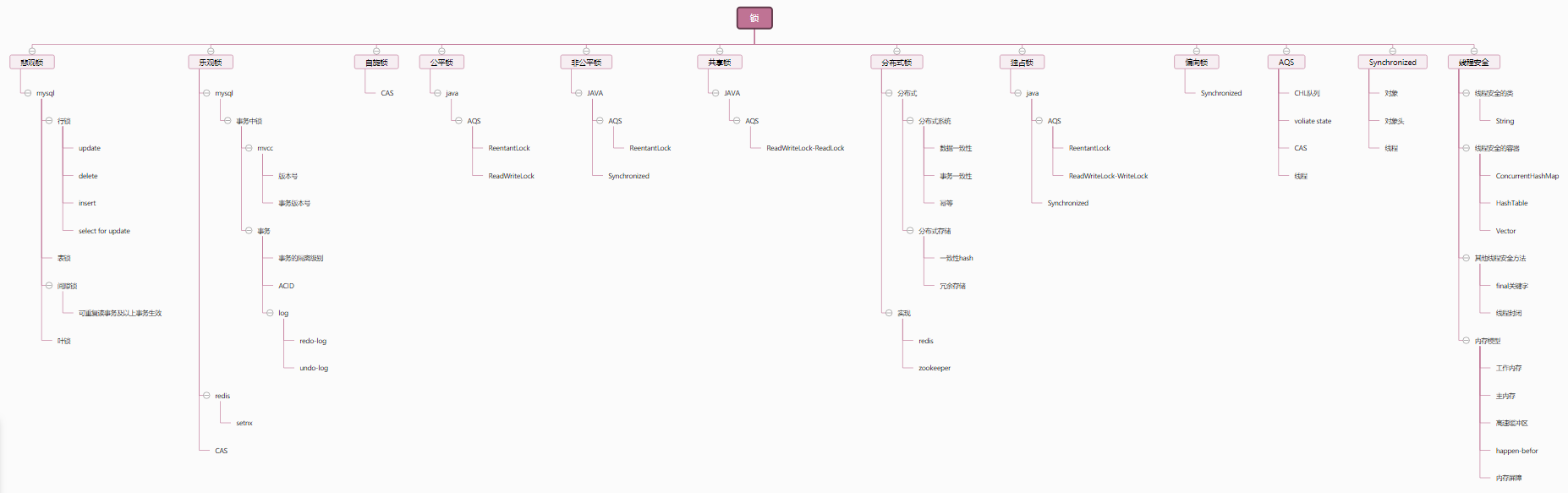
# 28个步骤快速搭建基于SpringBoot的Maven多模块Java Web项目

<https://blog.csdn.net/dalaoliang886/article/details/81160270>

# 两程序员玩“锁”，一人抢救无效身亡

<https://zhuanlan.zhihu.com/p/34510121>





# pom.xml配置文件

<**project xmlns**="http://maven.apache.org/POM/4.0.0" **xmlns:***xsi*="http://www.w3.org/2001/XMLSchema-instance" *xsi***:schemaLocation**="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">  
 <!--父项目的坐标。如果项目中没有规定某个元素的值，那么父项目中的对应值即为项目的默认值。 坐标包括group ID，artifact ID和 version。 -->  
 <**parent**>  
 <!--被继承的父项目的构件标识符 -->  
 <**artifactId** />  
 <!--被继承的父项目的全球唯一标识符 -->  
 <**groupId** />  
 <!--被继承的父项目的版本 -->  
 <**version** />  
 <!--父项目的pom.xml文件的相对路径。相对路径允许你选择一个不同的路径。默认值是../pom.xml。Maven首先在构建当前项目的地方寻找父项目的pom，其次在文件系统的这个位置（relativePath位置），然后在本地仓库，最后在远程仓库寻找父项目的pom。 -->  
 <**relativePath** />  
 </**parent**>  
  
 <!--声明项目描述符遵循哪一个POM模型版本。模型本身的版本很少改变，虽然如此，但它仍然是必不可少的，这是为了当Maven引入了新的特性或者其他模型变更的时候，确保稳定性。 -->  
 <**modelVersion**>4.0.0</**modelVersion**>  
 <!--项目的全球唯一标识符，通常使用全限定的包名区分该项目和其他项目。并且构建时生成的路径也是由此生成， 如com.mycompany.app生成的相对路径为：/com/mycompany/app -->  
 <**groupId**>asia.banseon</**groupId**>  
 <!--构件的标识符，它和group ID一起唯一标识一个构件。换句话说，你不能有两个不同的项目拥有同样的artifact ID和groupID；在某个特定的group  
 ID下，artifact ID也必须是唯一的。构件是项目产生的或使用的一个东西，Maven为项目产生的构件包括：JARs，源码，二进制发布和WARs等。 -->  
 <**artifactId**>banseon-maven2</**artifactId**>  
 <!--项目产生的构件类型，例如jar、war、ear、pom。插件可以创建他们自己的构件类型，所以前面列的不是全部构件类型 -->  
 <**packaging**>jar</**packaging**>  
 <!--项目当前版本，格式为:主版本.次版本.增量版本-限定版本号 -->  
 <**version**>1.0-SNAPSHOT</**version**>  
 <!--项目的名称, Maven产生的文档用 -->  
 <**name**>banseon-maven</**name**>  
 <!--项目主页的URL, Maven产生的文档用 -->  
 <**url**>http://www.baidu.com/banseon</**url**>  
 <!--项目的详细描述, Maven 产生的文档用。 当这个元素能够用HTML格式描述时（例如，CDATA中的文本会被解析器忽略，就可以包含HTML标签），  
 不鼓励使用纯文本描述。如果你需要修改产生的web站点的索引页面，你应该修改你自己的索引页文件，而不是调整这里的文档。 -->  
 <**description**>A maven project to study maven.</**description**>  
  
 <!--描述了这个项目构建环境中的前提条件。 -->  
 <**prerequisites**>  
 <!--构建该项目或使用该插件所需要的Maven的最低版本 -->  
 <**maven** />  
 </**prerequisites**>  
  
 <!--项目的问题管理系统(Bugzilla, Jira, Scarab,或任何你喜欢的问题管理系统)的名称和URL，本例为 jira -->  
 <**issueManagement**>  
 <!--问题管理系统（例如jira）的名字， -->  
 <**system**>jira</**system**>  
 <!--该项目使用的问题管理系统的URL -->  
 <**url**>http://jira.baidu.com/banseon</**url**>  
 </**issueManagement**>  
  
  
 <!--项目持续集成信息 -->  
 <**ciManagement**>  
 <!--持续集成系统的名字，例如continuum -->  
 <**system** />  
 <!--该项目使用的持续集成系统的URL（如果持续集成系统有web接口的话）。 -->  
 <**url** />  
 <!--构建完成时，需要通知的开发者/用户的配置项。包括被通知者信息和通知条件（错误，失败，成功，警告） -->  
 <**notifiers**>  
 <!--配置一种方式，当构建中断时，以该方式通知用户/开发者 -->  
 <**notifier**>  
 <!--传送通知的途径 -->  
 <**type** />  
 <!--发生错误时是否通知 -->  
 <**sendOnError** />  
 <!--构建失败时是否通知 -->  
 <**sendOnFailure** />  
 <!--构建成功时是否通知 -->  
 <**sendOnSuccess** />  
 <!--发生警告时是否通知 -->  
 <**sendOnWarning** />  
 <!--不赞成使用。通知发送到哪里 -->  
 <**address** />  
 <!--扩展配置项 -->  
 <**configuration** />  
 </**notifier**>  
 </**notifiers**>  
 </**ciManagement**>  
  
 <!--项目创建年份，4位数字。当产生版权信息时需要使用这个值。 -->  
 <**inceptionYear** />  
  
 <!--项目相关邮件列表信息 -->  
 <**mailingLists**>  
 <!--该元素描述了项目相关的所有邮件列表。自动产生的网站引用这些信息。 -->  
 <**mailingList**>  
 <!--邮件的名称 -->  
 <**name**>Demo</**name**>  
 <!--发送邮件的地址或链接，如果是邮件地址，创建文档时，mailto: 链接会被自动创建 -->  
 <**post**>banseon@126.com</**post**>  
 <!--订阅邮件的地址或链接，如果是邮件地址，创建文档时，mailto: 链接会被自动创建 -->  
 <**subscribe**>banseon@126.com</**subscribe**>  
 <!--取消订阅邮件的地址或链接，如果是邮件地址，创建文档时，mailto: 链接会被自动创建 -->  
 <**unsubscribe**>banseon@126.com</**unsubscribe**>  
 <!--你可以浏览邮件信息的URL -->  
 <**archive**>http:/hi.baidu.com/banseon/demo/dev/</**archive**>  
 </**mailingList**>  
 </**mailingLists**>  
  
  
 <!--项目开发者列表 -->  
 <**developers**>  
 <!--某个项目开发者的信息 -->  
 <**developer**>  
 <!--SCM里项目开发者的唯一标识符 -->  
 <**id**>HELLO WORLD</**id**>  
 <!--项目开发者的全名 -->  
 <**name**>banseon</**name**>  
 <!--项目开发者的email -->  
 <**email**>banseon@126.com</**email**>  
 <!--项目开发者的主页的URL -->  
 <**url** />  
 <!--项目开发者在项目中扮演的角色，角色元素描述了各种角色 -->  
 <**roles**>  
 <**role**>Project Manager</**role**>  
 <**role**>Architect</**role**>  
 </**roles**>  
 <!--项目开发者所属组织 -->  
 <**organization**>demo</**organization**>  
 <!--项目开发者所属组织的URL -->  
 <**organizationUrl**>http://hi.baidu.com/banseon</**organizationUrl**>  
 <!--项目开发者属性，如即时消息如何处理等 -->  
 <**properties**>  
 <**dept**>No</**dept**>  
 </**properties**>  
 <!--项目开发者所在时区， -11到12范围内的整数。 -->  
 <**timezone**>-5</**timezone**>  
 </**developer**>  
 </**developers**>  
  
  
 <!--项目的其他贡献者列表 -->  
 <**contributors**>  
 <!--项目的其他贡献者。参见developers/developer元素 -->  
 <**contributor**>  
 <**name** />  
 <**email** />  
 <**url** />  
 <**organization** />  
 <**organizationUrl** />  
 <**roles** />  
 <**timezone** />  
 <**properties** />  
 </**contributor**>  
 </**contributors**>  
  
  
 <!--该元素描述了项目所有License列表。 应该只列出该项目的license列表，不要列出依赖项目的 license列表。如果列出多个license，用户可以选择它们中的一个而不是接受所有license。 -->  
 <**licenses**>  
 <!--描述了项目的license，用于生成项目的web站点的license页面，其他一些报表和validation也会用到该元素。 -->  
 <**license**>  
 <!--license用于法律上的名称 -->  
 <**name**>Apache 2</**name**>  
 <!--官方的license正文页面的URL -->  
 <**url**>http://www.baidu.com/banseon/LICENSE-2.0.txt</**url**>  
 <!--项目分发的主要方式： repo，可以从Maven库下载 manual， 用户必须手动下载和安装依赖 -->  
 <**distribution**>repo</**distribution**>  
 <!--关于license的补充信息 -->  
 <**comments**>A business-friendly OSS license</**comments**>  
 </**license**>  
 </**licenses**>  
  
 <!--SCM(Source Control Management)标签允许你配置你的代码库，供Maven web站点和其它插件使用。 -->  
 <**scm**>  
 <!--SCM的URL,该URL描述了版本库和如何连接到版本库。欲知详情，请看SCMs提供的URL格式和列表。该连接只读。 -->  
 <**connection**>  
 scm:svn:http://svn.baidu.com/banseon/maven/banseon/banseon-maven2-trunk(dao-trunk)  
 </**connection**>  
 <!--给开发者使用的，类似connection元素。即该连接不仅仅只读 -->  
 <**developerConnection**>  
 scm:svn:http://svn.baidu.com/banseon/maven/banseon/dao-trunk  
 </**developerConnection**>  
 <!--当前代码的标签，在开发阶段默认为HEAD -->  
 <**tag** />  
 <!--指向项目的可浏览SCM库（例如ViewVC或者Fisheye）的URL。 -->  
 <**url**>http://svn.baidu.com/banseon</**url**>  
 </**scm**>  
  
 <!--描述项目所属组织的各种属性。Maven产生的文档用 -->  
 <**organization**>  
 <!--组织的全名 -->  
 <**name**>demo</**name**>  
 <!--组织主页的URL -->  
 <**url**>http://www.baidu.com/banseon</**url**>  
 </**organization**>  
  
  
 <!--构建项目需要的信息 -->  
 <**build**>  
 <!--该元素设置了项目源码目录，当构建项目的时候，构建系统会编译目录里的源码。该路径是相对于pom.xml的相对路径。 -->  
 <**sourceDirectory** />  
 <!--该元素设置了项目脚本源码目录，该目录和源码目录不同：绝大多数情况下，该目录下的内容 会被拷贝到输出目录(因为脚本是被解释的，而不是被编译的)。 -->  
 <**scriptSourceDirectory** />  
 <!--该元素设置了项目单元测试使用的源码目录，当测试项目的时候，构建系统会编译目录里的源码。该路径是相对于pom.xml的相对路径。 -->  
 <**testSourceDirectory** />  
 <!--被编译过的应用程序class文件存放的目录。 -->  
 <**outputDirectory** />  
 <!--被编译过的测试class文件存放的目录。 -->  
 <**testOutputDirectory** />  
  
 <!--使用来自该项目的一系列构建扩展 -->  
 <**extensions**>  
 <!--描述使用到的构建扩展。 -->  
 <**extension**>  
 <!--构建扩展的groupId -->  
 <**groupId** />  
 <!--构建扩展的artifactId -->  
 <**artifactId** />  
 <!--构建扩展的版本 -->  
 <**version** />  
 </**extension**>  
 </**extensions**>  
  
  
 <!--当项目没有规定目标（Maven2 叫做阶段）时的默认值 -->  
 <**defaultGoal** />  
  
  
 <!--这个元素描述了项目相关的所有资源路径列表，例如和项目相关的属性文件，这些资源被包含在最终的打包文件里。 -->  
 <**resources**>  
 <!--这个元素描述了项目相关或测试相关的所有资源路径 -->  
 <**resource**>  
 <!--描述了资源的目标路径。该路径相对target/classes目录（例如${project.build.outputDirectory}）。举个例子，如果你想资源在特定的包里(org.apache.maven.messages)，你就必须该元素设置为org/apache/maven/messages。然而，如果你只是想把资源放到源码目录结构里，就不需要该配置。 -->  
 <**targetPath** />  
 <!--是否使用参数值代替参数名。参数值取自properties元素或者文件里配置的属性，文件在filters元素里列出。 -->  
 <**filtering** />  
 <!--描述存放资源的目录，该路径相对POM路径 -->  
 <**directory** />  
 <!--包含的模式列表，例如\*\*\*.xml -->  
 <**excludes** />  
 </**resource**>  
 </**resources**>  
  
  
 <!--这个元素描述了单元测试相关的所有资源路径，例如和单元测试相关的属性文件。 -->  
 <**testResources**>  
 <!--这个元素描述了测试相关的所有资源路径，参见build/resources/resource元素的说明 -->  
 <**testResource**>  
 <**targetPath** />  
 <**filtering** />  
 <**directory** />  
 <**includes** />  
 <**excludes** />  
 </**testResource**>  
 </**testResources**>  
  
  
 <!--构建产生的所有文件存放的目录 -->  
 <**directory** />  
 <!--产生的构件的文件名，默认值是${artifactId}-${version}。 -->  
 <**finalName** />  
 <!--当filtering开关打开时，使用到的过滤器属性文件列表 -->  
 <**filters** />  
  
 <!--子项目可以引用的默认插件信息。该插件配置项直到被引用时才会被解析或绑定到生命周期。给定插件的任何本地配置都会覆盖这里的配置 -->  
 <**pluginManagement**>  
 <!--使用的插件列表 。 -->  
 <**plugins**>  
 <!--plugin元素包含描述插件所需要的信息。 -->  
 <**plugin**>  
 <!--插件在仓库里的group ID -->  
 <**groupId** />  
 <!--插件在仓库里的artifact ID -->  
 <**artifactId** />  
 <!--被使用的插件的版本（或版本范围） -->  
 <**version** />  
 <!--是否从该插件下载Maven扩展（例如打包和类型处理器），由于性能原因，只有在真需要下载时，该元素才被设置成enabled。 -->  
 <**extensions** />  
 <!--在构建生命周期中执行一组目标的配置。每个目标可能有不同的配置。 -->  
 <**executions**>  
 <!--execution元素包含了插件执行需要的信息 -->  
 <**execution**>  
 <!--执行目标的标识符，用于标识构建过程中的目标，或者匹配继承过程中需要合并的执行目标 -->  
 <**id** />  
 <!--绑定了目标的构建生命周期阶段，如果省略，目标会被绑定到源数据里配置的默认阶段 -->  
 <**phase** />  
 <!--配置的执行目标 -->  
 <**goals** />  
 <!--配置是否被传播到子POM -->  
 <**inherited** />  
 <!--作为DOM对象的配置 -->  
 <**configuration** />  
 </**execution**>  
  
 </**executions**>  
 <!--项目引入插件所需要的额外依赖 -->  
 <**dependencies**>  
 <!--参见dependencies/dependency元素 -->  
 <**dependency**>  
 ......  
 </**dependency**>  
 </**dependencies**>  
 <!--任何配置是否被传播到子项目 -->  
 <**inherited** />  
 <!--作为DOM对象的配置 -->  
 <**configuration** />  
 </**plugin**>  
 </**plugins**>  
 </**pluginManagement**>  
  
  
 <!--使用的插件列表 -->  
 <**plugins**>  
 <!--参见build/pluginManagement/plugins/plugin元素 -->  
 <**plugin**>  
 <**groupId** />  
 <**artifactId** />  
 <**version** />  
 <**extensions** />  
 <**executions**>  
 <**execution**>  
 <**id** />  
 <**phase** />  
 <**goals** />  
 <**inherited** />  
 <**configuration** />  
 </**execution**>  
 </**executions**>  
 <**dependencies**>  
 <!--参见dependencies/dependency元素 -->  
 <**dependency**>  
 ......  
 </**dependency**>  
 </**dependencies**>  
 <**goals** />  
 <**inherited** />  
 <**configuration** />  
 </**plugin**>  
 </**plugins**>  
 </**build**>  
  
  
 <!--在列的项目构建profile，如果被激活，会修改构建处理 -->  
 <**profiles**>  
 <!--根据环境参数或命令行参数激活某个构建处理 -->  
 <**profile**>  
 <!--构建配置的唯一标识符。即用于命令行激活，也用于在继承时合并具有相同标识符的profile。 -->  
 <**id** />  
 <!--自动触发profile的条件逻辑。Activation是profile的开启钥匙。profile的力量来自于它 能够在某些特定的环境中自动使用某些特定的值；这些环境通过activation元素指定。activation元素并不是激活profile的唯一方式。 -->  
 <**activation**>  
 <!--profile默认是否激活的标志 -->  
 <**activeByDefault** />  
 <!--当匹配的jdk被检测到，profile被激活。例如，1.4激活JDK1.4，1.4.0\_2，而!1.4激活所有版本不是以1.4开头的JDK。 -->  
 <**jdk** />  
 <!--当匹配的操作系统属性被检测到，profile被激活。os元素可以定义一些操作系统相关的属性。 -->  
 <**os**>  
 <!--激活profile的操作系统的名字 -->  
 <**name**>Windows XP</**name**>  
 <!--激活profile的操作系统所属家族(如 'windows') -->  
 <**family**>Windows</**family**>  
 <!--激活profile的操作系统体系结构 -->  
 <**arch**>x86</**arch**>  
 <!--激活profile的操作系统版本 -->  
 <**version**>5.1.2600</**version**>  
 </**os**>  
 <!--如果Maven检测到某一个属性（其值可以在POM中通过${名称}引用），其拥有对应的名称和值，Profile就会被激活。如果值 字段是空的，那么存在属性名称字段就会激活profile，否则按区分大小写方式匹配属性值字段 -->  
 <**property**>  
 <!--激活profile的属性的名称 -->  
 <**name**>mavenVersion</**name**>  
 <!--激活profile的属性的值 -->  
 <**value**>2.0.3</**value**>  
 </**property**>  
 <!--提供一个文件名，通过检测该文件的存在或不存在来激活profile。missing检查文件是否存在，如果不存在则激活 profile。另一方面，exists则会检查文件是否存在，如果存在则激活profile。 -->  
 <**file**>  
 <!--如果指定的文件存在，则激活profile。 -->  
 <**exists**>/usr/local/hudson/hudson-home/jobs/maven-guide-zh-to-production/workspace/  
 </**exists**>  
 <!--如果指定的文件不存在，则激活profile。 -->  
 <**missing**>/usr/local/hudson/hudson-home/jobs/maven-guide-zh-to-production/workspace/  
 </**missing**>  
 </**file**>  
 </**activation**>  
 <!--构建项目所需要的信息。参见build元素 -->  
 <**build**>  
 <**defaultGoal** />  
 <**resources**>  
 <**resource**>  
 <**targetPath** />  
 <**filtering** />  
 <**directory** />  
 <**includes** />  
 <**excludes** />  
 </**resource**>  
 </**resources**>  
 <**testResources**>  
 <**testResource**>  
 <**targetPath** />  
 <**filtering** />  
 <**directory** />  
 <**includes** />  
 <**excludes** />  
 </**testResource**>  
 </**testResources**>  
 <**directory** />  
 <**finalName** />  
 <**filters** />  
 <**pluginManagement**>  
 <**plugins**>  
 <!--参见build/pluginManagement/plugins/plugin元素 -->  
 <**plugin**>  
 <**groupId** />  
 <**artifactId** />  
 <**version** />  
 <**extensions** />  
 <**executions**>  
 <**execution**>  
 <**id** />  
 <**phase** />  
 <**goals** />  
 <**inherited** />  
 <**configuration** />  
 </**execution**>  
 </**executions**>  
 <**dependencies**>  
 <!--参见dependencies/dependency元素 -->  
 <**dependency**>  
 ......  
 </**dependency**>  
 </**dependencies**>  
 <**goals** />  
 <**inherited** />  
 <**configuration** />  
 </**plugin**>  
 </**plugins**>  
 </**pluginManagement**>  
 <**plugins**>  
 <!--参见build/pluginManagement/plugins/plugin元素 -->  
 <**plugin**>  
 <**groupId** />  
 <**artifactId** />  
 <**version** />  
 <**extensions** />  
 <**executions**>  
 <**execution**>  
 <**id** />  
 <**phase** />  
 <**goals** />  
 <**inherited** />  
 <**configuration** />  
 </**execution**>  
 </**executions**>  
 <**dependencies**>  
 <!--参见dependencies/dependency元素 -->  
 <**dependency**>  
 ......  
 </**dependency**>  
 </**dependencies**>  
 <**goals** />  
 <**inherited** />  
 <**configuration** />  
 </**plugin**>  
 </**plugins**>  
 </**build**>  
 <!--模块（有时称作子项目） 被构建成项目的一部分。列出的每个模块元素是指向该模块的目录的相对路径 -->  
 <**modules** />  
 <!--发现依赖和扩展的远程仓库列表。 -->  
 <**repositories**>  
 <!--参见repositories/repository元素 -->  
 <**repository**>  
 <**releases**>  
 <**enabled** />  
 <**updatePolicy** />  
 <**checksumPolicy** />  
 </**releases**>  
 <**snapshots**>  
 <**enabled** />  
 <**updatePolicy** />  
 <**checksumPolicy** />  
 </**snapshots**>  
 <**id** />  
 <**name** />  
 <**url** />  
 <**layout** />  
 </**repository**>  
 </**repositories**>  
 <!--发现插件的远程仓库列表，这些插件用于构建和报表 -->  
 <**pluginRepositories**>  
 <!--包含需要连接到远程插件仓库的信息.参见repositories/repository元素 -->  
 <**pluginRepository**>  
 <**releases**>  
 <**enabled** />  
 <**updatePolicy** />  
 <**checksumPolicy** />  
 </**releases**>  
 <**snapshots**>  
 <**enabled** />  
 <**updatePolicy** />  
 <**checksumPolicy** />  
 </**snapshots**>  
 <**id** />  
 <**name** />  
 <**url** />  
 <**layout** />  
 </**pluginRepository**>  
 </**pluginRepositories**>  
 <!--该元素描述了项目相关的所有依赖。 这些依赖组成了项目构建过程中的一个个环节。它们自动从项目定义的仓库中下载。要获取更多信息，请看项目依赖机制。 -->  
 <**dependencies**>  
 <!--参见dependencies/dependency元素 -->  
 <**dependency**>  
 ......  
 </**dependency**>  
 </**dependencies**>  
 <!--不赞成使用. 现在Maven忽略该元素. -->  
 <**reports** />  
 <!--该元素包括使用报表插件产生报表的规范。当用户执行“mvn site”，这些报表就会运行。 在页面导航栏能看到所有报表的链接。参见reporting元素 -->  
 <**reporting**>  
 ......  
 </**reporting**>  
 <!--参见dependencyManagement元素 -->  
 <**dependencyManagement**>  
 <**dependencies**>  
 <!--参见dependencies/dependency元素 -->  
 <**dependency**>  
 ......  
 </**dependency**>  
 </**dependencies**>  
 </**dependencyManagement**>  
 <!--参见distributionManagement元素 -->  
 <**distributionManagement**>  
 ......  
 </**distributionManagement**>  
 <!--参见properties元素 -->  
 <**properties** />  
 </**profile**>  
 </**profiles**>  
  
  
 <!--模块（有时称作子项目） 被构建成项目的一部分。列出的每个模块元素是指向该模块的目录的相对路径 -->  
 <**modules** />  
  
 <!--发现依赖和扩展的远程仓库列表。 -->  
 <**repositories**>  
 <!--包含需要连接到远程仓库的信息 -->  
 <**repository**>  
 <!--如何处理远程仓库里发布版本的下载 -->  
 <**releases**>  
 <!--true或者false表示该仓库是否为下载某种类型构件（发布版，快照版）开启。 -->  
 <**enabled** />  
 <!--该元素指定更新发生的频率。Maven会比较本地POM和远程POM的时间戳。这里的选项是：always（一直），daily（默认，每日），interval：X（这里X是以分钟为单位的时间间隔），或者never（从不）。 -->  
 <**updatePolicy** />  
 <!--当Maven验证构件校验文件失败时该怎么做：ignore（忽略），fail（失败），或者warn（警告）。 -->  
 <**checksumPolicy** />  
 </**releases**>  
 <!--如何处理远程仓库里快照版本的下载。有了releases和snapshots这两组配置，POM就可以在每个单独的仓库中，为每种类型的构件采取不同的策略。例如，可能有人会决定只为开发目的开启对快照版本下载的支持。参见repositories/repository/releases元素 -->  
 <**snapshots**>  
 <**enabled** />  
 <**updatePolicy** />  
 <**checksumPolicy** />  
 </**snapshots**>  
 <!--远程仓库唯一标识符。可以用来匹配在settings.xml文件里配置的远程仓库 -->  
 <**id**>banseon-repository-proxy</**id**>  
 <!--远程仓库名称 -->  
 <**name**>banseon-repository-proxy</**name**>  
 <!--远程仓库URL，按protocol://hostname/path形式 -->  
 <**url**>http://192.168.1.169:9999/repository/</**url**>  
 <!--用于定位和排序构件的仓库布局类型-可以是default（默认）或者legacy（遗留）。Maven 2为其仓库提供了一个默认的布局；然而，Maven  
 1.x有一种不同的布局。我们可以使用该元素指定布局是default（默认）还是legacy（遗留）。 -->  
 <**layout**>default</**layout**>  
 </**repository**>  
 </**repositories**>  
  
 <!--发现插件的远程仓库列表，这些插件用于构建和报表 -->  
 <**pluginRepositories**>  
 <!--包含需要连接到远程插件仓库的信息.参见repositories/repository元素 -->  
 <**pluginRepository**>  
 ......  
 </**pluginRepository**>  
 </**pluginRepositories**>  
  
 <!--该元素描述了项目相关的所有依赖。 这些依赖组成了项目构建过程中的一个个环节。它们自动从项目定义的仓库中下载。要获取更多信息，请看项目依赖机制。 -->  
 <**dependencies**>  
 <**dependency**>  
 <!--依赖的group ID -->  
 <**groupId**>org.apache.maven</**groupId**>  
 <!--依赖的artifact ID -->  
 <**artifactId**>maven-artifact</**artifactId**>  
 <!--依赖的版本号。 在Maven 2里, 也可以配置成版本号的范围。 -->  
 <**version**>3.8.1</**version**>  
 <!--依赖类型，默认类型是jar。它通常表示依赖的文件的扩展名，但也有例外。一个类型可以被映射成另外一个扩展名或分类器。类型经常和使用的打包方式对应，尽管这也有例外。一些类型的例子：jar，war，ejb-client和test-jar。如果设置extensions为  
 true，就可以在plugin里定义新的类型。所以前面的类型的例子不完整。 -->  
 <**type**>jar</**type**>  
 <!--依赖的分类器。分类器可以区分属于同一个POM，但不同构建方式的构件。分类器名被附加到文件名的版本号后面。例如，如果你想要构建两个单独的构件成JAR，一个使用Java  
 1.4编译器，另一个使用Java 6编译器，你就可以使用分类器来生成两个单独的JAR构件。 -->  
 <**classifier**></**classifier**>  
 <!--依赖范围。在项目发布过程中，帮助决定哪些构件被包括进来。欲知详情请参考依赖机制。 - compile ：默认范围，用于编译 - provided：类似于编译，但支持你期待jdk或者容器提供，类似于classpath  
 - runtime: 在执行时需要使用 - test: 用于test任务时使用 - system: 需要外在提供相应的元素。通过systemPath来取得  
 - systemPath: 仅用于范围为system。提供相应的路径 - optional: 当项目自身被依赖时，标注依赖是否传递。用于连续依赖时使用 -->  
 <**scope**>test</**scope**>  
 <!--仅供system范围使用。注意，不鼓励使用这个元素，并且在新的版本中该元素可能被覆盖掉。该元素为依赖规定了文件系统上的路径。需要绝对路径而不是相对路径。推荐使用属性匹配绝对路径，例如${java.home}。 -->  
 <**systemPath**></**systemPath**>  
 <!--当计算传递依赖时， 从依赖构件列表里，列出被排除的依赖构件集。即告诉maven你只依赖指定的项目，不依赖项目的依赖。此元素主要用于解决版本冲突问题 -->  
 <**exclusions**>  
 <**exclusion**>  
 <**artifactId**>spring-core</**artifactId**>  
 <**groupId**>org.springframework</**groupId**>  
 </**exclusion**>  
 </**exclusions**>  
 <!--可选依赖，如果你在项目B中把C依赖声明为可选，你就需要在依赖于B的项目（例如项目A）中显式的引用对C的依赖。可选依赖阻断依赖的传递性。 -->  
 <**optional**>true</**optional**>  
 </**dependency**>  
 </**dependencies**>  
  
 <!--不赞成使用. 现在Maven忽略该元素. -->  
 <**reports**></**reports**>  
  
 <!--该元素描述使用报表插件产生报表的规范。当用户执行“mvn site”，这些报表就会运行。 在页面导航栏能看到所有报表的链接。 -->  
 <**reporting**>  
 <!--true，则，网站不包括默认的报表。这包括“项目信息”菜单中的报表。 -->  
 <**excludeDefaults** />  
 <!--所有产生的报表存放到哪里。默认值是${project.build.directory}/site。 -->  
 <**outputDirectory** />  
 <!--使用的报表插件和他们的配置。 -->  
 <**plugins**>  
 <!--plugin元素包含描述报表插件需要的信息 -->  
 <**plugin**>  
 <!--报表插件在仓库里的group ID -->  
 <**groupId** />  
 <!--报表插件在仓库里的artifact ID -->  
 <**artifactId** />  
 <!--被使用的报表插件的版本（或版本范围） -->  
 <**version** />  
 <!--任何配置是否被传播到子项目 -->  
 <**inherited** />  
 <!--报表插件的配置 -->  
 <**configuration** />  
 <!--一组报表的多重规范，每个规范可能有不同的配置。一个规范（报表集）对应一个执行目标 。例如，有1，2，3，4，5，6，7，8，9个报表。1，2，5构成A报表集，对应一个执行目标。2，5，8构成B报表集，对应另一个执行目标 -->  
 <**reportSets**>  
 <!--表示报表的一个集合，以及产生该集合的配置 -->  
 <**reportSet**>  
 <!--报表集合的唯一标识符，POM继承时用到 -->  
 <**id** />  
 <!--产生报表集合时，被使用的报表的配置 -->  
 <**configuration** />  
 <!--配置是否被继承到子POMs -->  
 <**inherited** />  
 <!--这个集合里使用到哪些报表 -->  
 <**reports** />  
 </**reportSet**>  
 </**reportSets**>  
 </**plugin**>  
 </**plugins**>  
 </**reporting**>  
  
 <!--继承自该项目的所有子项目的默认依赖信息。这部分的依赖信息不会被立即解析,而是当子项目声明一个依赖（必须描述group ID和artifact ID信息），如果group ID和artifact ID以外的一些信息没有描述，则通过group ID和artifact ID匹配到这里的依赖，并使用这里的依赖信息。 -->  
 <**dependencyManagement**>  
 <**dependencies**>  
 <!--参见dependencies/dependency元素 -->  
 <**dependency**>  
 ......  
 </**dependency**>  
 </**dependencies**>  
 </**dependencyManagement**>  
  
  
 <!--项目分发信息，在执行mvn deploy后表示要发布的位置。有了这些信息就可以把网站部署到远程服务器或者把构件部署到远程仓库。 -->  
 <**distributionManagement**>  
 <!--部署项目产生的构件到远程仓库需要的信息 -->  
 <**repository**>  
 <!--是分配给快照一个唯一的版本号（由时间戳和构建流水号）？还是每次都使用相同的版本号？参见repositories/repository元素 -->  
 <**uniqueVersion** />  
 <**id**>banseon-maven2</**id**>  
 <**name**>banseon maven2</**name**>  
 <**url**>file://${basedir}/target/deploy</**url**>  
 <**layout** />  
 </**repository**>  
 <!--构件的快照部署到哪里？如果没有配置该元素，默认部署到repository元素配置的仓库，参见distributionManagement/repository元素 -->  
 <**snapshotRepository**>  
 <**uniqueVersion** />  
 <**id**>banseon-maven2</**id**>  
 <**name**>Banseon-maven2 Snapshot Repository</**name**>  
 <**url**>scp://svn.baidu.com/banseon:/usr/local/maven-snapshot</**url**>  
 <**layout** />  
 </**snapshotRepository**>  
 <!--部署项目的网站需要的信息 -->  
 <**site**>  
 <!--部署位置的唯一标识符，用来匹配站点和settings.xml文件里的配置 -->  
 <**id**>banseon-site</**id**>  
 <!--部署位置的名称 -->  
 <**name**>business api website</**name**>  
 <!--部署位置的URL，按protocol://hostname/path形式 -->  
 <**url**>  
 scp://svn.baidu.com/banseon:/var/www/localhost/banseon-web  
 </**url**>  
 </**site**>  
 <!--项目下载页面的URL。如果没有该元素，用户应该参考主页。使用该元素的原因是：帮助定位那些不在仓库里的构件（由于license限制）。 -->  
 <**downloadUrl** />  
 <!--如果构件有了新的group ID和artifact ID（构件移到了新的位置），这里列出构件的重定位信息。 -->  
 <**relocation**>  
 <!--构件新的group ID -->  
 <**groupId** />  
 <!--构件新的artifact ID -->  
 <**artifactId** />  
 <!--构件新的版本号 -->  
 <**version** />  
 <!--显示给用户的，关于移动的额外信息，例如原因。 -->  
 <**message** />  
 </**relocation**>  
 <!--给出该构件在远程仓库的状态。不得在本地项目中设置该元素，因为这是工具自动更新的。有效的值有：none（默认），converted（仓库管理员从Maven  
 1 POM转换过来），partner（直接从伙伴Maven 2仓库同步过来），deployed（从Maven 2实例部署），verified（被核实时正确的和最终的）。 -->  
 <**status** />  
 </**distributionManagement**>  
  
 <!--以值替代名称，Properties可以在整个POM中使用，也可以作为触发条件（见settings.xml配置文件里activation元素的说明）。格式是<name>value</name>。 -->  
 <**properties** />  
</**project**>

应用举例

<**project xmlns**="http://maven.apache.org/POM/4.0.0" **xmlns:***xsi*="http://www.w3.org/2001/XMLSchema-instance"  
 *xsi***:schemaLocation**="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">  
 <**modelVersion**>4.0.0</**modelVersion**>  
 <**parent**>  
 <**groupId**>com.hgame</**groupId**>  
 <**artifactId**>ATam</**artifactId**>  
 <**version**>0.0.1-SNAPSHOT</**version**>  
 </**parent**>  
 <**artifactId**>mybatis-generator-plugin</**artifactId**>  
  
 <**name**>mybatis-generator-plugin</**name**>  
 <**url**>http://maven.apache.org</**url**>  
  
 <**properties**>  
 <**project.build.sourceEncoding**>UTF-8</**project.build.sourceEncoding**>  
 </**properties**>  
  
 <**dependencies**>  
 <**dependency**>  
 <**groupId**>org.mybatis.generator</**groupId**>  
 <**artifactId**>mybatis-generator-core</**artifactId**>  
 <**version**>1.3.2</**version**>  
 <**type**>jar</**type**>  
 </**dependency**>  
  
 <**dependency**>  
 <**groupId**>junit</**groupId**>  
 <**artifactId**>junit</**artifactId**>  
 <**version**>3.8.1</**version**>  
 <**scope**>test</**scope**>  
 </**dependency**>  
 </**dependencies**>  
 <**build**>  
 <**plugins**>  
 <**plugin**>  
 <**groupId**>org.apache.maven.plugins</**groupId**>  
 <**artifactId**>maven-compiler-plugin</**artifactId**>  
 <**configuration**>  
 <**source**>1.8</**source**>  
 <**target**>1.8</**target**>  
 <**verbose**>true</**verbose**>  
 <**encoding**>${project.build.sourceEncoding}</**encoding**>  
 <**compilerArguments**>  
 <**sourcepath**>${project.basedir}/src/main/java</**sourcepath**>  
 </**compilerArguments**>  
 </**configuration**>  
 </**plugin**>  
  
 <**plugin**>  
 <**groupId**>org.mybatis.generator</**groupId**>  
 <**artifactId**>mybatis-generator-maven-plugin</**artifactId**>  
 <**version**>1.3.2</**version**>  
 <**configuration**>  
 <**verbose**>true</**verbose**>  
 <**overwrite**>true</**overwrite**>  
 </**configuration**>  
 </**plugin**>  
  
 </**plugins**>  
 </**build**>  
</**project**>

# maven全局配置文件settings.xml详解

<https://www.cnblogs.com/jingmoxukong/p/6050172.html?utm_source=gold_browser_extension>

# 设置通过Maven创建的工程的JDK版本—一劳永逸

方法一:在项目中的pom.xml指定jdk版本

这个方法只能保证该项目是jdk1.8版本，每次新建项目都得加上面代码，不推荐使用，推荐第二种方法。

|  |
| --- |
| <build>  <plugins>  <plugin>  <groupId>org.apache.maven.plugins</groupId>  <artifactId>maven-compiler-plugin</artifactId>  <version>3.1</version>  <configuration>  <source>1.8</source>  <target>1.8</target>  </configuration>  </plugin>  </plugins>  </build> |

方法二:在maven的安装目录找到settings.xml文件,在里面添加如下代码

1:打开settings.xml

2:找到profiles标签

3:加入如下配置

|  |
| --- |
| <profile>  <id>jdk-1.8</id>  <activation>  <activeByDefault>true</activeByDefault>  <jdk>1.8</jdk>  </activation>  <properties>  <maven.compiler.source>1.8</maven.compiler.source>  <maven.compiler.target>1.8</maven.compiler.target>  <maven.compiler.compilerVersion>1.8</maven.compiler.compilerVersion>  </properties>  </profile> |

依赖范围<scope></scope>中的test和provided

可以这么理解:只给当前工程自己用的,不能传递给其他工程~

# 创建一个标准的单工程maven web项目

new project->Maven ->Maven Project

->勾选create a simple project

->packaging=war

->properties->project facets->

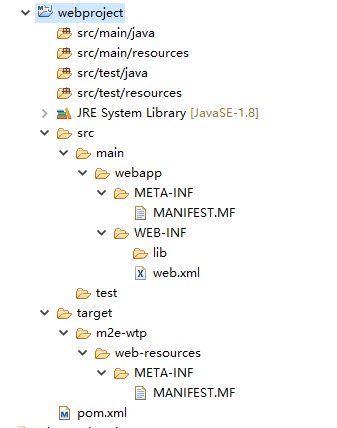
->Dynamic Web Module 2.5 (去掉勾)->apply

->Dynamic Web Module 2.5 (加上勾)->further configuration available

->修改content directory = src/main/webapp

->勾选Generate web.xml deployment descriptor

=finish



# 创建一个标准的多工程maven项目

|  |
| --- |
| 父工程---pom工程  子工程1---jar工程  子工程2---jar工程  子工程3---jar工程  web工程---war工程 |

1:创建pom父工程项目

new project->Maven ->Maven Project

->勾选create a simple project

->packaging=pom

=finish

parent/pom.xml

|  |
| --- |
| <project xmlns=*"http://maven.apache.org/POM/4.0.0"* xmlns:xsi=*"http://www.w3.org/2001/XMLSchema-instance"*  xsi:schemaLocation=*"http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd"*>  <modelVersion>4.0.0</modelVersion>  <groupId>com.tianqing</groupId>  <artifactId>parent</artifactId>  <version>0.0.1-SNAPSHOT</version>  <packaging>pom</packaging>  <modules>  <module>common-api</module>  </modules>  </project> |

2:创建子模块jar工程

在parent项目上右键new project->Maven ->Maven Module

->parent project的名称应该为:parent

->勾选create a simple project

->packaging = jar

=finish

son1/pom.xml

|  |
| --- |
| <project xmlns=*"http://maven.apache.org/POM/4.0.0"* xmlns:xsi=*"http://www.w3.org/2001/XMLSchema-instance"*  xsi:schemaLocation=*"http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd"*>  <modelVersion>4.0.0</modelVersion>  <parent>  <groupId>com.tianqing</groupId>  <artifactId>parent</artifactId>  <version>0.0.1-SNAPSHOT</version>  </parent>  <artifactId>son1</artifactId>  </project> |

同理创建son2,son3

son2/pom.xml

|  |
| --- |
| <project xmlns=*"http://maven.apache.org/POM/4.0.0"* xmlns:xsi=*"http://www.w3.org/2001/XMLSchema-instance"*  xsi:schemaLocation=*"http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd"*>  <modelVersion>4.0.0</modelVersion>  <parent>  <groupId>com.tianqing</groupId>  <artifactId>parent</artifactId>  <version>0.0.1-SNAPSHOT</version>  </parent>  <artifactId>son2</artifactId>  </project> |

son3/pom.xml

|  |
| --- |
| <project xmlns=*"http://maven.apache.org/POM/4.0.0"* xmlns:xsi=*"http://www.w3.org/2001/XMLSchema-instance"*  xsi:schemaLocation=*"http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd"*>  <modelVersion>4.0.0</modelVersion>  <parent>  <groupId>com.tianqing</groupId>  <artifactId>parent</artifactId>  <version>0.0.1-SNAPSHOT</version>  </parent>  <artifactId>son3</artifactId>  </project> |

如果son1,son2,son3之间存在依赖关系,则可以给他们之间添加依赖;

如何给模块与模块之间添加依赖?

点开pom.xml->dependencies->add

->Enter groupId...输入已经建好的工程,选中

->finish

[1]:如果是树型依赖关系

son2->son1

son3->son1

添加完成后,查看pom文件->

son/pom.xml是不改变的,因为它不依赖其他的工程文件~

son2/pom.xml

|  |
| --- |
| <project xmlns=*"http://maven.apache.org/POM/4.0.0"* xmlns:xsi=*"http://www.w3.org/2001/XMLSchema-instance"*  xsi:schemaLocation=*"http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd"*>  <modelVersion>4.0.0</modelVersion>  <parent>  <groupId>com.tianqing</groupId>  <artifactId>parent</artifactId>  <version>0.0.1-SNAPSHOT</version>  </parent>  <artifactId>son2</artifactId>  <dependencies>  <dependency>  <groupId>com.tianqing</groupId>  <artifactId>son1</artifactId>  <version>0.0.1-SNAPSHOT</version>  </dependency>  </dependencies>  </project> |

son3/pom.xml

|  |
| --- |
| <project xmlns=*"http://maven.apache.org/POM/4.0.0"* xmlns:xsi=*"http://www.w3.org/2001/XMLSchema-instance"*  xsi:schemaLocation=*"http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd"*>  <modelVersion>4.0.0</modelVersion>  <parent>  <groupId>com.tianqing</groupId>  <artifactId>parent</artifactId>  <version>0.0.1-SNAPSHOT</version>  </parent>  <artifactId>son3</artifactId>  <dependencies>  <dependency>  <groupId>com.tianqing</groupId>  <artifactId>son1</artifactId>  <version>0.0.1-SNAPSHOT</version>  </dependency>  </dependencies>  </project> |

[2]: 如果是串型依赖关系

son3->son2->son1

添加完成后,查看pom文件->

son2/pom.xml

|  |
| --- |
| <project xmlns=*"http://maven.apache.org/POM/4.0.0"* xmlns:xsi=*"http://www.w3.org/2001/XMLSchema-instance"*  xsi:schemaLocation=*"http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd"*>  <modelVersion>4.0.0</modelVersion>  <parent>  <groupId>com.tianqing</groupId>  <artifactId>parent</artifactId>  <version>0.0.1-SNAPSHOT</version>  </parent>  <artifactId>son2</artifactId>  <dependencies>  <dependency>  <groupId>com.tianqing</groupId>  <artifactId>son1</artifactId>  <version>0.0.1-SNAPSHOT</version>  </dependency>  </dependencies>  </project> |

son3/pom.xml

|  |
| --- |
| <project xmlns=*"http://maven.apache.org/POM/4.0.0"* xmlns:xsi=*"http://www.w3.org/2001/XMLSchema-instance"*  xsi:schemaLocation=*"http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd"*>  <modelVersion>4.0.0</modelVersion>  <parent>  <groupId>com.tianqing</groupId>  <artifactId>parent</artifactId>  <version>0.0.1-SNAPSHOT</version>  </parent>  <artifactId>son3</artifactId>  <dependencies>  <dependency>  <groupId>com.tianqing</groupId>  <artifactId>son2</artifactId>  <version>0.0.1-SNAPSHOT</version>  </dependency>  </dependencies>  </project> |

3:创建web模块war工程

在parent项目上右键new project->Maven ->Maven Module

->parent project的名称应该为:parent

->勾选create a simple project

->packaging = war

->properties->project facets->

->Dynamic Web Module 2.5 (去掉勾)->apply

->Dynamic Web Module 2.5 (加上勾)->further configuration available

->修改content directory = src/main/webapp

->勾选Generate web.xml deployment descriptor

=finish

web/pom.xml

|  |
| --- |
| <project xmlns=*"http://maven.apache.org/POM/4.0.0"* xmlns:xsi=*"http://www.w3.org/2001/XMLSchema-instance"*  xsi:schemaLocation=*"http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd"*>  <modelVersion>4.0.0</modelVersion>  <parent>  <groupId>com.tianqing</groupId>  <artifactId>parent</artifactId>  <version>0.0.1-SNAPSHOT</version>  </parent>  <artifactId>web</artifactId>  <packaging>war</packaging>  </project> |

所有的parent,son1,son2,son3和web模块建立完成后,parent/pom.xml会出现变化

parent/pom.xml

|  |
| --- |
| <project xmlns=*"http://maven.apache.org/POM/4.0.0"* xmlns:xsi=*"http://www.w3.org/2001/XMLSchema-instance"*  xsi:schemaLocation=*"http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd"*>  <modelVersion>4.0.0</modelVersion>  <groupId>com.tianqing</groupId>  <artifactId>parent</artifactId>  <version>0.0.1-SNAPSHOT</version>  <packaging>pom</packaging>  <modules>  <module>common-api</module>  <module>son1</module>  <module>son2</module>  <module>son3</module>  <module>web</module>  </modules>  </project> |

4:如果想可以编译,运行项目,则还需要做一些工作~

1):在webapp下创建一个index.jsp

2):在web/pom.xml中添加**servlet-api.jar的依赖,且为provided范围的依赖**

|  |
| --- |
| <dependencies>  <!-- https://mvnrepository.com/artifact/org.apache.tomcat/servlet-api -->  <dependency>  <groupId>org.apache.tomcat</groupId>  <artifactId>servlet-api</artifactId>  <version>6.0.53</version>  <scope>provided</scope>  </dependency>  </dependencies> |

3):在web/pom.xml中添加编译插件+tomcat运行插件

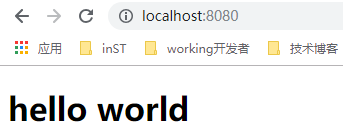
|  |
| --- |
| <build>  <plugins>  <!-- 编码和编译和JDK版本 -->  <plugin>  <groupId>org.apache.maven.plugins</groupId>  <artifactId>maven-compiler-plugin</artifactId>  <version>2.3.2</version>  <configuration>  <source>1.8</source>  <target>1.8</target>  <encoding>utf8</encoding>  </configuration>  </plugin>  <!-- tomcat插件 -->  <plugin>  <groupId>org.apache.tomcat.maven</groupId>  <artifactId>tomcat7-maven-plugin</artifactId>  <version>2.2</version>  <configuration>  <url>http://localhost/manager/text</url>  <server>tomcat7</server>  <username>tomcat</username>  <password>tomcat</password>  <port>8080</port>  <path>/</path>  </configuration>  </plugin>  </plugins>  </build> |

4):在parent目录下创建clean install

5):在web目录下创建tomcat7:run

6):编译运行,访问<http://localhost:8080>

->finish



# 使用命令创建模块化多工程项目

1:创建父工程-parent模块

mvn archetype:create -DgroupId=me.gacl -DartifactId=system-parent -DarchetypeArtifactId=maven-archetype-quickstart -DinteractiveMode=false

需要修改pom.xml中**<packaging>jar</packaging>**修改为**<packaging>pom</packaging>**

2:创建子模块-domain模块

mvn archetype:create -DgroupId=me.gacl -DartifactId=system-domain -DarchetypeArtifactId=maven-archetype-quickstart -DinteractiveMode=false

需要修改pom.xml中**<groupId>me.gacl</groupId>**和**<version>1.0-SNAPSHOT</version>**去掉，加上**<packaging>jar</packaging>**

3:创建子模块-dao模块

mvn archetype:create -DgroupId=me.gacl -DartifactId=system-dao -DarchetypeArtifactId=maven-archetype-quickstart -DinteractiveMode=false

需要修改pom.xml中**<groupId>me.gacl</groupId>**和**<version>1.0-SNAPSHOT</version>**去掉，加上**<packaging>jar</packaging>,同时添加对system-domain模块的依赖**

3:创建子模块-service模块

mvn archetype:create -DgroupId=me.gacl -DartifactId=system-service -DarchetypeArtifactId=maven-archetype-quickstart -DinteractiveMode=false

需要修改pom.xml中**<groupId>me.gacl</groupId>**和**<version>1.0-SNAPSHOT</version>**去掉，加上**<packaging>jar</packaging>，同时添加对system-dao模块的依赖**，system-service依赖system-dao和system-domain，但是我们只需添加system-dao的依赖即可，因为system-dao已经依赖了system-domain

4:创建web模块

mvn archetype:create -DgroupId=me.gacl -DartifactId=system-web -DarchetypeArtifactId=**maven-archetype-webapp** -DinteractiveMode=false

需要修改pom.xml中**<groupId>me.gacl</groupId>**和**<version>1.0-SNAPSHOT</version>**去掉，**同时添加对system-service模块的依赖**

# 如何使用aliyun maven镜像?

找到settings.xml文件

找到<mirror>标签

|  |
| --- |
| <mirrors>  <mirror>  <id>alimaven</id>  <name>aliyun maven</name>  <url>http://maven.aliyun.com/nexus/content/groups/public/</url>  <mirrorOf>central</mirrorOf>  </mirror>  </mirrors> |

保存即可使用

# Jetty Maven插件

使用容器运行的前提是:都需要编译插件:

|  |
| --- |
| <!-- 编码和编译和JDK版本 -->  <plugin>  <groupId>org.apache.maven.plugins</groupId>  <artifactId>maven-compiler-plugin</artifactId>  <version>2.3.2</version>  <configuration>  <source>1.8</source>  <target>1.8</target>  <encoding>utf8</encoding>  </configuration>  </plugin> |

以前的tomcat7插件

|  |
| --- |
| <!-- tomcat插件 -->  <plugin>  <groupId>org.apache.tomcat.maven</groupId>  <artifactId>tomcat7-maven-plugin</artifactId>  <version>2.2</version>  <configuration>  <url>http://localhost/manager/text</url>  <server>tomcat7</server>  <username>tomcat</username>  <password>tomcat</password>  <port>8080</port>  <path>/</path>  </configuration>  </plugin> |

tomcat7:run

现在使用Jetty插件

|  |
| --- |
| <!-- jetty插件 -->  <plugin>  <groupId>org.mortbay.jetty</groupId>  <artifactId>jetty-maven-plugin</artifactId>  <version>8.1.16.v20140903</version>  <configuration>  <scanIntervalSeconds>10</scanIntervalSeconds>  <webApp>  <contextPath>/test</contextPath> <!-- 工程目录 -->  </webApp>  <connectors>  <connector implementation=*"org.eclipse.jetty.server.nio.SelectChannelConnector"*>  <port>8082</port>  </connector>  </connectors>  </configuration>  </plugin> |

jetty:run

# springboot maven聚合工程搭建

1:创建pom父工程项目

new project->Maven ->Maven Project

->勾选create a simple project

->packaging=pom

=finish

在pom.xml中添加springboot的parent标签

parent/pom.xml

|  |
| --- |
| <?xml version=*"1.0"* encoding=*"UTF-8"*?>  <project xmlns=*"http://maven.apache.org/POM/4.0.0"* xmlns:xsi=*"http://www.w3.org/2001/XMLSchema-instance"*  xsi:schemaLocation=*"http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd"*>  <modelVersion>4.0.0</modelVersion>  <groupId>com.tianqing</groupId>  <artifactId>spring-parent</artifactId>  <version>0.0.1-SNAPSHOT</version>  <packaging>pom</packaging>      <!-- 自定义starter都应该继承自该依赖 -->  <!-- 如果自定义starter本身需要继承其它的依赖，可以参考 https://stackoverflow.com/a/21318359 解决 -->  <parent>  <groupId>org.springframework.boot</groupId>  <artifactId>spring-boot-starters</artifactId>  <version>1.5.2.RELEASE</version>  </parent>  </project> |

2:创建子模块jar工程

在parent项目上右键new project->Maven ->Maven Module

->parent project的名称应该为:parent

->不勾选create a simple project

->选择quickstart

=finish

创建

demo-core

demo-api

demo-service

其中依赖关系

demo-api->demo.core

demo-service->demo-api

demo-service->demo.core

demo-core/pom.xml

|  |
| --- |
| <?xml version=*"1.0"* encoding=*"UTF-8"*?>  <project xmlns=*"http://maven.apache.org/POM/4.0.0"* xmlns:xsi=*"http://www.w3.org/2001/XMLSchema-instance"*  xsi:schemaLocation=*"http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd"*>  <modelVersion>4.0.0</modelVersion>  <parent>  <artifactId>spring-parent</artifactId>  <groupId>com.tianqing</groupId>  <version>0.0.1-SNAPSHOT</version>  </parent>  <groupId>com.tianqing</groupId>  <artifactId>demo-core</artifactId>  <version>0.0.1-SNAPSHOT</version>  <name>demo-core</name>  </project> |

demo-api/pom.xml

|  |
| --- |
| <?xml version=*"1.0"* encoding=*"UTF-8"*?>  <project xmlns=*"http://maven.apache.org/POM/4.0.0"* xmlns:xsi=*"http://www.w3.org/2001/XMLSchema-instance"*  xsi:schemaLocation=*"http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd"*>  <modelVersion>4.0.0</modelVersion>  <parent>  <artifactId>spring-parent</artifactId>  <groupId>com.tianqing</groupId>  <version>0.0.1-SNAPSHOT</version>  </parent>  <groupId>com.tianqing</groupId>  <artifactId>demo-api</artifactId>  <version>0.0.1-SNAPSHOT</version>  <name>demo-api</name>  <dependencies>  <dependency>  <groupId>com.tianqing</groupId>  <artifactId>demo-core</artifactId>  <version>0.0.1-SNAPSHOT</version>  </dependency>  </dependencies>  </project> |

在pom.xml中添加springboot的spring-boot-starter

demo-service/pom.xml

|  |
| --- |
| <?xml version=*"1.0"* encoding=*"UTF-8"*?>  <project xmlns=*"http://maven.apache.org/POM/4.0.0"* xmlns:xsi=*"http://www.w3.org/2001/XMLSchema-instance"*  xsi:schemaLocation=*"http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd"*>  <modelVersion>4.0.0</modelVersion>  <parent>  <artifactId>spring-parent</artifactId>  <groupId>com.tianqing</groupId>  <version>0.0.1-SNAPSHOT</version>  </parent>  <groupId>com.tianqing</groupId>  <artifactId>demo-service</artifactId>  <version>0.0.1-SNAPSHOT</version>  <name>demo-service</name>  <dependencies>  <dependency>  <groupId>com.tianqing</groupId>  <artifactId>demo-api</artifactId>  <version>0.0.1-SNAPSHOT</version>  </dependency>  <dependency>  <groupId>com.tianqing</groupId>  <artifactId>demo-core</artifactId>  <version>0.0.1-SNAPSHOT</version>  </dependency>  <dependency>  <groupId>org.springframework.boot</groupId>  <artifactId>spring-boot-starter</artifactId>  <version>1.5.12.RELEASE</version>  </dependency>  </dependencies>  </project> |

demo-web/pom.xml

|  |
| --- |
| <project xmlns=*"http://maven.apache.org/POM/4.0.0"* xmlns:xsi=*"http://www.w3.org/2001/XMLSchema-instance"*  xsi:schemaLocation=*"http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd"*>  <modelVersion>4.0.0</modelVersion>  <parent>  <groupId>com.tianqing</groupId>  <artifactId>spring-parent</artifactId>  <version>0.0.1-SNAPSHOT</version>  </parent>  <artifactId>demo-web</artifactId>  <packaging>war</packaging>  <dependencies>  <dependency>  <groupId>javax.servlet</groupId>  <artifactId>servlet-api</artifactId>  <version>2.5</version>  <scope>provided</scope>  </dependency>  <dependency>  <groupId>org.springframework.boot</groupId>  <artifactId>spring-boot-starter-web</artifactId>  <exclusions>  <exclusion>  <groupId>com.fasterxml.jackson.core</groupId>  <artifactId>jackson-databind</artifactId>  </exclusion>  <exclusion>  <groupId>org.springframework.boot</groupId>  <artifactId>spring-boot-starter-tomcat</artifactId>  </exclusion>  </exclusions>  </dependency>  </dependencies>  <build>  <plugins>  <!-- 编码和编译和JDK版本 -->  <plugin>  <groupId>org.apache.maven.plugins</groupId>  <artifactId>maven-compiler-plugin</artifactId>  <version>2.3.2</version>  <configuration>  <source>1.8</source>  <target>1.8</target>  <encoding>utf8</encoding>  </configuration>  </plugin>  <!-- jetty插件 -->  <plugin>  <groupId>org.mortbay.jetty</groupId>  <artifactId>jetty-maven-plugin</artifactId>  <version>8.1.16.v20140903</version>  <configuration>  <scanIntervalSeconds>10</scanIntervalSeconds>  <webApp>  <contextPath>/</contextPath> <!-- 工程目录 -->  </webApp>  <connectors>  <connector implementation=*"org.eclipse.jetty.server.nio.SelectChannelConnector"*>  <port>8082</port>  </connector>  </connectors>  </configuration>  </plugin>  </plugins>  </build>  </project> |

3:在demo-web工程

在src/main/webapp目录下

创建/WEB-INFO/lib

创建/WEB-INFO/web.xml

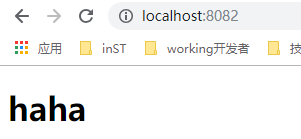
创建index.jsp

就可以

编译parent

运行web了

然后访问容器



# 多模块JavaWeb项目结构设计：

|  |
| --- |
| 简约式分层结构：  system-parent（项目父目录）  |----pom.xml  |----system-domain（域模型层）  |----pom.xml  |----system-dao（数据库访问层）  |----pom.xml  |----system-service（业务逻辑层）  |----pom.xml  |----system-web（表现层）  |----pom.xml  分布式分层结构：  system（项目父目录，一般写项目名作为父目录、根目录）  |----pom.xml  |----system-model（数据模型层、实体层，可用entity、domain表示）  |----pom.xml  |----system-dao（数据库访问层）  |----pom.xml  |----system-dao-impl（数据库访问实现层，如果用mybatis这里框架可以省略这层）  |----pom.xml  |----system-service（业务逻辑接口层）  |----pom.xml  |----system-service-impl（业务逻辑接口实现层，使用dubbo时，要发布该层）  |----pom.xml  |----system-web（表现层、可用controller控制表现层表示）  |----pom.xml  |----system-common（公共通用层、存放些base类等）  |----pom.xml  |----system-utils（工具层、存放些工具类等）  |----pom.xml  |----system-rabbitmq（独立模块功能层，根据需要可以多个功能模块层，例如cas, redis等）  |----pom.xml |

# 在web工程中添加profile配置

web模块的pom.xml文件如下：

|  |
| --- |
| 1. *<!-- 配置各种环境的profile，方便各种环境配置的切换 -->* 2. <profiles> 3. <profile> 4. <id>dev</id> 5. <properties> 6. <env.type>dev</env.type> 7. </properties> 8. *<!-- 默认选择哪个配置，需要的话把activation设置在哪个profile里 -->* 9. <activation> 10. <activeByDefault>true</activeByDefault> 11. </activation> 12. </profile> 13. <profile> 14. <id>test</id> 15. <properties> 16. <env.type>test</env.type> 17. </properties> 18. </profile> 19. <profile> 20. <id>prod</id> 21. <properties> 22. <env.type>prod</env.type> 23. </properties> 24. </profile> 25. </profiles> 27. <build> 28. <resources> 29. *<!-- 项目相关的所有资源路径列表，例如和项目相关的配置文件、属性文件，这些资源被包含在最终的打包文件里 -->* 30. <resource> 31. *<!--描述存放资源的目录，该路径相对POM路径 -->* 32. <directory>src/main/resources</directory> 33. <includes> 34. *<!--包含的模式列表, profile配置文件所在路径 -->* 35. <include>/properties/\*\*/\*.properties</include> 36. </includes> 37. </resource> 38. <resource> 39. *<!--描述存放资源的目录，该路径相对POM路径 -->* 40. <directory>src/main/resources</directory> 41. *<!--是否使用参数值代替参数名。参数值取自properties元素或者文件里配置的属性，文件在filters元素里列出，必须加上 -->* 42. <filtering>true</filtering> 43. </resource> 44. </resources> 45. <plugins> 46. <plugin> 47. <groupId>org.apache.maven.plugins</groupId> 48. <artifactId>maven-compiler-plugin</artifactId> 49. <version>3.2</version> 50. <configuration> 51. <source>1.8</source> 52. <target>1.8</target> 53. </configuration> 54. </plugin> 56. *<!-- 引入tomcat中间件的maven插件，方便打包发布运行war包 -->* 57. <plugin> 58. <groupId>org.apache.tomcat.maven</groupId> 59. <artifactId>tomcat7-maven-plugin</artifactId> 60. <version>2.2</version> 61. <configuration> 62. <port>8181</port> 63. <charset>UTF-8</charset> 64. <server>tomcat7</server> 65. </configuration> 66. </plugin> 67. </plugins> 69. *<!-- 最终打包出来的war包名称 -->* 70. <finalName>ji-book-web</finalName> 71. </build> |

# 自动化编译打包的指令为：(由参数 -P区分开)

|  |
| --- |
| 1. mvn clean install -Dmaven.test.skip=true -Ptest 2. mvn clean install -Dmaven.test.skip=true -Pdev 3. mvn clean install -Dmaven.test.skip=true -Pprod |

# Spring的多事务配置（多个Transaction Manager）和使用方法

大多数项目只需要一个事务管理器。然而，有些项目为了提高效率、或者有多个完全不同又不相干的数据源，最好用多个事务管理器。

机智的Spring的Transactional管理已经考虑到了这一点，首先分别定义多个transactional manager，并为qualifier属性指定不同的值；

然后在需要使用@Transactional注解的时候指定TransactionManager的qualifier属性值或者直接使用bean名称。配置和代码使用的例子：

|  |
| --- |
| 1. <tx:annotation-driven/> 3. <bean id="transactionManager1" class="org.springframework.jdbc.datasource.DataSourceTransactionManager"> 4. <property name="dataSource" ref="datasource1"></property> 5. <qualifier value="datasource1Tx"/> 6. </bean> 8. <bean id="transactionManager2" class="org.springframework.jdbc.datasource.DataSourceTransactionManager"> 9. <property name="dataSource" ref="datasource2"></property> 10. <qualifier value="datasource2Tx"/> 11. </bean> |
| 1. public class TransactionalService { 3. @Transactional("datasource1Tx") 4. public void setSomethingInDatasource1() { ... } 6. @Transactional("datasource2Tx") 7. public void doSomethingInDatasource2() { ... } 8. } |
| 或者，直接使用transaction manager 的bean名字：  @Transactional("transactionManager1")  如果是使用@Transactional()，相当于使用缺省的transaction mananger名字，即：@Transactional("transactionManager") |

# 1:SpringMVC配置

web.xml的配置

SpringMVC主要还是要依赖Servlet，所以需要进行配置web.xml(顺带配置一个过滤器，让SpringMVC所有操作都是UTF-8格式): (必须的)

|  |
| --- |
| <!-- springMVC配置 -->  <servlet>  <servlet-name>springmvc</servlet-name>  <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>  <init-param>  <!-- Springmvc 配置文件所在的位置 -->  <param-name>contextConfigLocation</param-name>  <param-value>classpath:springmvc.xml</param-value>  </init-param>  <load-on-startup>1</load-on-startup>  </servlet>  <servlet-mapping>  <servlet-name>springmvc</servlet-name>  <url-pattern>/</url-pattern>  </servlet-mapping>  <!-- 过滤器:UTF-8 编码 防止插入数据库的数据乱码 -->  <filter>  <filter-name>encodingFilter</filter-name>  <filter-class>org.springframework.web.filter.CharacterEncodingFilter</filter-class>  <init-param>  <param-name>encoding</param-name>  <param-value>UTF-8</param-value>  </init-param>  </filter>  <filter-mapping>  <filter-name>encodingFilter</filter-name>  <url-pattern>/\*</url-pattern>  </filter-mapping> |

springmvc.xml的配置(必须的)

|  |
| --- |
| <!-- 配置SpringMVC -->  <!-- 开启SpringMVC注解模式 -->  <!--简化配置  (1)自动注册DefaultannotationHandlerMapping,AnnotationMethodhandlerAdapter  (2)默认提供了一系列的功能:数据绑定,数字和日期的转换format @NumberFormat,@DataTimeFormat,xml,json的读写支持  -->  <mvc:annotation-driven />  <!-- servlet-mapping 映射路径:"/" -->  <!-- 静态资源默认Servlet配置  1:加入对静态资源的处理:js,gif,png  2:允许使用"/"做整体配置映射  -->  <mvc:default-servlet-handler/>  <!-- 后面的xml和类基本上都要自动包扫描，让Spring认识 -->  <content:component-scan base-package="com.test.all"/> |

springmvc.xml的试图解析器(必须的)

|  |
| --- |
| <!-- 配置springMVC的视图解析器 -->  <bean class=*"org.springframework.web.servlet.view.InternalResourceViewResolver"*>  <property name=*"viewClass"* value=*"org.springframework.web.servlet.view.JstlView"* />  <property name=*"prefix"* value=*"/WEB-INF/"* />  <property name=*"suffix"* value=*".jsp"* />  </bean> |

springmvc.xml配置数据校验器

|  |
| --- |
| <!-- 配置校验器 ，使用的是hibernate框架的validate，但与hibernate无关 -->  <bean id=*"validator"*  class=*"org.springframework.validation.beanvalidation.LocalValidatorFactoryBean"*>  <!-- 校验器，使用hibernate校验器 -->  <property name=*"providerClass"* value=*"org.hibernate.validator.HibernateValidator"* />  <!-- 指定校验使用的资源文件，在文件中配置校验错误信息，如果不指定则默认使用classpath下面的ValidationMessages.properties文件 -->  <property name=*"validationMessageSource"* ref=*"messageSource"* />  </bean> |

springmvc.xml配置校验错误信息

|  |
| --- |
| <!-- 校验错误信息配置文件 -->  <bean id=*"messageSource"* class=*"org.springframework.context.support.ReloadableResourceBundleMessageSource"*>  <!-- 资源文件名 -->  <property name=*"basenames"*>  <list>  <value>classpath:validateMessage</value>  </list>  </property>  <!-- 资源文件编码格式 -->  <property name=*"fileEncodings"* value=*"utf-8"* />  <!-- 对资源文件内容缓存时间，单位秒 -->  <property name=*"cacheSeconds"* value=*"120"* />  </bean> |

springmvc.xml国际化配置

|  |
| --- |
| <!-- 国际化 -->  <bean id="messageSource" class="org.springframework.context.support.ResourceBundleMessageSource">  <!-- 国际化信息所在的文件名 -->  <property name="basename" value="i18n.i18n"/>  <property name="defaultEncoding" value="utf-8"/>  <!-- 如果在国际化资源文件中找不到对应代码的信息，就用这个代码作为名称 -->  <property name="useCodeAsDefaultMessage" value="true"/>  </bean>  <!-- SessionLocalReslover -->  <bean id="localeResolver" class="org.springframework.web.servlet.i18n.SessionLocaleResolver">  <property name="defaultLocale" value="CHINESE"/>  </bean>  <!-- LocalChangeInterceptor -->  <!-- 国际化操作拦截器 如果采用基于（请求/Session/Cookie）则必需配置 -->  <mvc:interceptors>  <bean class="org.springframework.web.servlet.i18n.LocaleChangeInterceptor"/>  </mvc:interceptors> |

springmvc.xml文件上传

|  |
| --- |
| <!-- 文件上传 -->  <bean id=*"multipartResolver"*  class=*"org.springframework.web.multipart.commons.CommonsMultipartResolver"*>  <!-- 上传文件大小上限，单位为字节（10MB） -->  <property name=*"maxUploadSize"*>  <value>10485760</value>  </property>  <!-- 请求的编码格式，必须和jSP的pageEncoding属性一致，以便正确读取表单的内容，默认为ISO-8859-1 -->  <property name=*"defaultEncoding"*>  <value>UTF-8</value>  </property>  </bean> |

# 2:Spring配置

spring的maven依赖

|  |
| --- |
| <properties>  <webVersion>3.0</webVersion>  <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>  <!-- spring版本号 -->  <spring.version>4.0.0.RELEASE</spring.version>  <!-- log4j日志文件管理包版本 -->  <!-- <slf4j.version>1.6.6</slf4j.version> <log4j.version>1.2.12</log4j.version> -->  <!-- mybatis版本号 -->  <mybatis.version>3.1.1</mybatis.version>  <!-- aspectJ版本号 -->  <aspectJ.version>1.7.4</aspectJ.version>  <!-- 数据库连接池版本号 -->  <commons-dbcp.version>1.4</commons-dbcp.version>  <commons-pool.version>1.6</commons-pool.version>  <!-- jackson的版本号 json和xml解析 -->  <jackson.version>2.8.9</jackson.version>  <!-- mybatis/spring包的版本号 -->  <mybatis-spring.version>1.1.1</mybatis-spring.version>  <!-- mysql的版本号 -->  <mysql.version>5.1.29</mysql.version>  <!-- jstl的版本号 -->  <jstl.version>1.2.1</jstl.version>  <!-- 文件上传用的包 的版本号 -->  <commons-fileupload.version>1.3.1</commons-fileupload.version>  <commons-io.version>2.4</commons-io.version>  <!-- 数据校验使用的包的版本号 -->  <validation-api.version>2.0.0.Final</validation-api.version>  <hibernate-validator.version>5.4.1.Final</hibernate-validator.version>  </properties>  <!-- 单元测试 -->  <dependency>  <groupId>junit</groupId>  <artifactId>junit</artifactId>  <version>4.12</version>  <scope>test</scope>  </dependency>  <!-- spring test -->  <dependency>  <groupId>org.springframework</groupId>  <artifactId>spring-test</artifactId>  <version>4.0.0.RELEASE</version>  <scope>test</scope>  </dependency>  <dependency>  <groupId>commons-logging</groupId>  <artifactId>commons-logging</artifactId>  <version>1.1.3</version>  </dependency>  <dependency>  <groupId>commons-collections</groupId>  <artifactId>commons-collections</artifactId>  <version>3.2.1</version>  </dependency>  <!-- jstl -->  <dependency>  <groupId>javax.servlet</groupId>  <artifactId>jstl</artifactId>  <version>1.2</version>  </dependency>  <!-- mybatis -->  <dependency>  <groupId>org.mybatis</groupId>  <artifactId>mybatis</artifactId>  <version>${mybatis.version}</version>  </dependency>  <dependency>  <groupId>org.mybatis</groupId>  <artifactId>mybatis-spring</artifactId>  <version>${mybatis-spring.version}</version>  </dependency>  <dependency>  <groupId>mysql</groupId>  <artifactId>mysql-connector-java</artifactId>  <version>${mysql.version}</version>  </dependency>  <dependency>  <groupId>com.alibaba</groupId>  <artifactId>druid</artifactId>  <version>0.2.23</version>  </dependency>  <!-- spring -->  <dependency>  <groupId>org.springframework</groupId>  <artifactId>spring-core</artifactId>  <version>${spring.version}</version>  </dependency>  <dependency>  <groupId>org.aspectj</groupId>  <artifactId>aspectjrt</artifactId>  <version>${aspectJ.version}</version>  </dependency>  <dependency>  <groupId>org.aspectj</groupId>  <artifactId>aspectjweaver</artifactId>  <version>${aspectJ.version}</version>  </dependency>  <dependency>  <groupId>org.springframework</groupId>  <artifactId>spring-context</artifactId>  <version>${spring.version}</version>  </dependency>  <dependency>  <groupId>org.springframework</groupId>  <artifactId>spring-aop</artifactId>  <version>${spring.version}</version>  </dependency>  <dependency>  <groupId>org.springframework</groupId>  <artifactId>spring-tx</artifactId>  <version>${spring.version}</version>  </dependency>  <dependency>  <groupId>org.springframework</groupId>  <artifactId>spring-jdbc</artifactId>  <version>${spring.version}</version>  </dependency>  <dependency>  <groupId>org.springframework</groupId>  <artifactId>spring-web</artifactId>  <version>${spring.version}</version>  </dependency>  <dependency>  <groupId>org.springframework</groupId>  <artifactId>spring-webmvc</artifactId>  <version>${spring.version}</version>  </dependency>  <dependency>  <groupId>org.springframework</groupId>  <artifactId>spring-context-support</artifactId>  <version>${spring.version}</version>  </dependency>  <!-- Jackson -->  <dependency>  <groupId>com.google.code.gson</groupId>  <artifactId>gson</artifactId>  <version>2.8.5</version>  </dependency>  <dependency>  <groupId>org.codehaus.jackson</groupId>  <artifactId>jackson-mapper-asl</artifactId>  <version>1.9.13</version>  </dependency>  <dependency>  <groupId>com.alibaba</groupId>  <artifactId>fastjson</artifactId>  <version>1.2.3</version>  </dependency>  <dependency>  <groupId>com.fasterxml.jackson.core</groupId>  <artifactId>jackson-annotations</artifactId>  <version>${jackson.version}</version>  </dependency>  <dependency>  <groupId>com.fasterxml.jackson.core</groupId>  <artifactId>jackson-core</artifactId>  <version>${jackson.version}</version>  </dependency>  <dependency>  <groupId>com.fasterxml.jackson.core</groupId>  <artifactId>jackson-databind</artifactId>  <version>${jackson.version}</version>  </dependency>  <!-- 解决.jsp出错 -->  <dependency>  <groupId>javax.servlet</groupId>  <artifactId>javax.servlet-api</artifactId>  <version>3.1.0</version>  <scope>provided</scope>  </dependency>  <!-- 连接池 -->  <dependency>  <groupId>commons-dbcp</groupId>  <artifactId>commons-dbcp</artifactId>  <version>${commons-dbcp.version}</version>  </dependency>  <dependency>  <groupId>commons-pool</groupId>  <artifactId>commons-pool</artifactId>  <version>${commons-pool.version}</version>  </dependency>  <!-- 文件上传用的包 -->  <dependency>  <groupId>commons-fileupload</groupId>  <artifactId>commons-fileupload</artifactId>  <version>${commons-fileupload.version}</version>  </dependency>  <dependency>  <groupId>commons-io</groupId>  <artifactId>commons-io</artifactId>  <version>${commons-io.version}</version>  </dependency>  <!-- 数据校验 -->  <dependency>  <groupId>javax.validation</groupId>  <artifactId>validation-api</artifactId>  <version>${validation-api.version}</version>  </dependency>  <dependency>  <groupId>org.hibernate</groupId>  <artifactId>hibernate-validator</artifactId>  <version>${hibernate-validator.version}</version>  </dependency>  <!-- 发送邮件 -->  <dependency>  <groupId>javax.mail</groupId>  <artifactId>mail</artifactId>  <version>1.4.7</version>  </dependency>  <!-- Freemarker -->  <dependency>  <groupId>org.freemarker</groupId>  <artifactId>freemarker</artifactId>  <version>2.3.23</version>  </dependency>  <!-- velocity模板引擎 -->  <dependency>  <groupId>org.apache.velocity</groupId>  <artifactId>velocity</artifactId>  <version>1.7</version>  </dependency>  <dependency>  <groupId>org.apache.velocity</groupId>  <artifactId>velocity-tools</artifactId>  <version>2.0</version>  </dependency>  <dependency>  <groupId>javax.servlet</groupId>  <artifactId>jsp-api</artifactId>  <version>2.0</version>  <scope>provided</scope>  </dependency> |

web.xml的配置

|  |
| --- |
| <!-- 加载spring容器 ，如果没有配置这个话会出现service无法注入到controller层的情况 -->  <context-param>  <param-name>contextConfigLocation</param-name>  <param-value>classpath:spring/applicationContext-\*.xml</param-value>  </context-param>  <!-- 在启动Web容器时，自动装配Spring applicationContext.xml的配置信息。 -->  <listener>  <listener-class>org.springframework.web.context.ContextLoaderListener</listener-class>  </listener> |

分离配置文件:

spring-dao 主要是jdbc连接，mybatis的sqlSessionFactory等配置，总之和数据库打交道

spring-quartz 很简单，就只是定时器的配置。

spring-redis 是redis的配置

spring-service 扫描service包

spring-transaction 是事务、扫描、注解等配置

spring-mvc 则是springMVC的相关配置,还有拦截器，可以根据自己的需要继续往上面添加

至此，所有的分离基本完成，你就可以根据自己的需要进行配置修改了。

分离的好处

分离之后，明显赶紧结构化清晰很多。相关的配置很清楚在那一块。之后需要新加入配置也很简单。

需要指出的是，每个配置文件上面的协议最好写清楚，不要一股脑全部复制粘贴，用到什么写什么。

尽可能的简化配置文件，让配置文件看的明白。

spring-dao.xml

|  |
| --- |
| <!-- 加载数据库属性文件，为配置数据源做准备 -->  <context:property-placeholder location=*"classpath:db.properties"*  ignore-unresolvable=*"true"* />  <!-- bean采用注解式注入 -->  <context:annotation-config></context:annotation-config>  <!-- 配置数据源 使用dbcp -->  <bean id=*"dataSource"* class=*"org.apache.commons.dbcp.BasicDataSource"*  destroy-method=*"close"*>  <property name=*"driverClassName"* value=*"${jdbc.driverClassName}"* />  <property name=*"url"* value=*"${jdbc.url}"* />  <property name=*"username"* value=*"${jdbc.username}"* />  <property name=*"password"* value=*"${jdbc.password}"* />  </bean>  <!-- =================================MyBatis的整合====================================== -->  <!-- 配置mybatis的SqlSessionFactoryBean -->  <bean id=*"sqlSessionFactory"* class=*"org.mybatis.spring.SqlSessionFactoryBean"*>  <!-- 指定mybatis 全局配置文件的位置 -->  <property name=*"configLocation"* value=*"classpath:mybatis/spring-mybatis.xml"*></property>  <property name=*"dataSource"* ref=*"dataSource"*></property>  </bean>  <!-- mapper配置: mapper批量处理，从mapper包中扫描mapper接口，自动创建代理对象并且在spring容器中注册  遵循规范：将mapper.java和mapper.xml映射文件名称保持一致，且在一个目录中自动扫描出来的mapper的bean的id为mapper类名（首字母小写） -->  <bean class=*"org.mybatis.spring.mapper.MapperScannerConfigurer"*>  <property name=*"basePackage"* value=*"com.snailclimb.dao"*></property>  <property name=*"sqlSessionFactoryBeanName"* value=*"sqlSessionFactory"*></property>  </bean> |

spring-service.xml

|  |
| --- |
| <!-- 用户管理service -->  <context:component-scan base-package=*"com.snailclimb.service.impl"*></context:component-scan> |

spring-transaction.xml

|  |
| --- |
| <!--事务管理器 -->  <bean id=*"transactionManager"* class=*"org.springframework.jdbc.datasource.DataSourceTransactionManager"*>  <property name=*"dataSource"* ref=*"dataSource"* />  </bean>  <!--通知 -->  <tx:advice id=*"txAdvice"* transaction-manager=*"transactionManager"*>  <tx:attributes>  <tx:method name=*"save\*"* propagation=*"REQUIRED"* />  <tx:method name=*"insert\*"* propagation=*"REQUIRED"* />  <tx:method name=*"add\*"* propagation=*"REQUIRED"* />  <tx:method name=*"create\*"* propagation=*"REQUIRED"* />  <tx:method name=*"delete\*"* propagation=*"REQUIRED"* />  <tx:method name=*"update\*"* propagation=*"REQUIRED"* />  <tx:method name=*"find\*"* propagation=*"SUPPORTS"* read-only=*"true"* />  <tx:method name=*"select\*"* propagation=*"SUPPORTS"* read-only=*"true"* />  <tx:method name=*"get\*"* propagation=*"SUPPORTS"* read-only=*"true"* />  </tx:attributes>  </tx:advice>  <aop:config>  <aop:advisor advice-ref=*"txAdvice"* pointcut=*"execution(\* com.snailclimb.service.\*.\*(..))"* />  </aop:config> |

# ssm配置文件分层架构

