# maven

## 修改maven的setting.xml (.m2目录下), 在 <profiles> 中添加, 指定默认jdk版本

<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>

或者修改 File->ProjectStructure->ProjectSetting->Modules 每个模块的 LanguageLevel 与 ProjectSetting->Project->ProjectSDK和ProjectLanguageLevel一致

## 指定源文件编码, 避免maven complie 的警告

<properties>  
 <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>  
 <project.reporting.outputEncoding>UTF-8</project.reporting.outputEncoding>  
 <java.version>1.8</java.version>  
</properties>

## maven生命周期

maven clear 会删除target目录

maven compile 会编译出class文件

maven package 会生产jar包

maven install 会将生成的jar包和pom文件拷贝到.m2目录

## 运行有依赖的jar包

如果不使用-Xbootclasspath方式, 正确的做法是不要用-jar来运行, 而是将主jar包和其所依赖的jar包都放入-classpath中,然后再指定main类

java.exe –classpath "C:\Users\liyh\.m2\repository\log4j\log4j\1.2.17\log4j-1.2.17.jar;E:\project\test\target\test1-1.0-SNAPSHOT.jar" com.felix.test1.App

而且这样也能避免非要在主的jar包中的MANIFEST.MF文件中指定Main-Class

~~另一种方式是在maven打包时, 将依赖的库也打入这个jar包, 需要使用到maven-shade-plugin, pow.xml 配置如下:~~

~~<build>~~

~~<plugins>~~

~~<plugin>~~

**~~<groupId>org.apache.maven.plugins</groupId>~~**

**~~<artifactId>maven-shade-plugin</artifactId>~~**

**~~<version>3.2.1</version>~~**

~~<executions>~~

~~<execution>~~

**~~<phase>package</phase>~~**

**~~<goals>~~**

**~~<goal>shade</goal>~~**

**~~</goals>~~**

~~</execution>~~

~~</executions>~~

~~</plugin>~~

~~</plugins>~~

~~</build>~~

~~同时 这个 maven-shade-plugin 还能用于指定 该 jar包运行时的主类(它会在打包时修改MANIFEST.MF文件,设定Main-Class), 配置如下~~

~~<build>~~

~~<plugins>~~

~~<plugin>~~

**~~<groupId>org.apache.maven.plugins</groupId>~~**

**~~<artifactId>maven-shade-plugin</artifactId>~~**

**~~<version>3.2.1</version>~~**

~~<executions>~~

~~<execution>~~

~~<phase>package</phase>~~

~~<goals>~~

~~<goal>shade</goal>~~

~~</goals>~~

**~~<configuration>~~**

**~~<transformers>~~**

**~~<transformer~~**

**~~implementation="org.apache.maven.plugins.shade.resource.ManifestResourceTransformer">~~**

**~~<mainClass>com.felix.test1.App</mainClass>~~**

**~~</transformer>~~**

**~~</transformers>~~**

**~~</configuration>~~**

~~</execution>~~

~~</executions>~~

~~</plugin>~~

~~</plugins>~~

~~</build>~~

~~idea-maven 调试或运行时会执行编译(并不会打包), 启动时会把~~

~~%JAVA\_HOME%\jre\lib(\ext)下的几个基本的jar包 和~~

~~编译目录的 target\classes 和~~

~~maven-pom.xml 中的 dependences(在.m2下)的jar包~~

~~加入 -classpath, 运行 java.exe –classpath "xx" 启动类~~

## idea-maven 父模块只是子模块的容器

并不真正编译生成任何包, 子模块是真正编译产生jar包的

父模块的pom文件中使用modules来管理子模块, 编译父模块,会编译其下的所有子模

子模块POM文件中 使用 <parent>来指定父模块, 这里指定的父模块与idea-maven接口可以不一致, 子模块继承父模块的 groupId, dependencies

## 运行有依赖的jar包, 使用spring-boot-maven-plugin

springboot工程可以将所有的dependence包含在jar中, 需要如下POM配置

<parent>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-parent</artifactId>

<version>1.5.3.RELEASE</version>

<relativePath/> <!-- lookup parent from repository -->

</parent>

<build>

<plugins>

<plugin>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-maven-plugin</artifactId>

</plugin>

</plugins>

</build>

打出的jar包, 所有的依赖会在 \BOOT-INF\lib 下, 并且会在\META-INF\MANIFEST.MF中设置好Start-Class

如果不借助 springboot, 可以在POM中如下配置, 并手动设置mainClass (Start-Class)

<build>  
 <plugins>  
 <plugin>  
 <groupId>org.springframework.boot</groupId>  
 <artifactId>spring-boot-maven-plugin</artifactId>  
 <version>2.1.4.RELEASE</version>  
 <configuration>  
 <mainClass>**com.felix.demo.s1.App**</mainClass>  
 <layout>ZIP</layout>  
 </configuration>  
 <executions>  
 <execution>  
 <goals>  
 <goal>repackage</goal>  
 </goals>  
 </execution>  
 </executions>  
 </plugin>  
 </plugins>  
</build>

maven项目的多继承

POM文件中只能指定一个<parent>, 会继承parent的dependency和groupid等等, 可以将项目中所有的依赖都放到parent的dependencyManagement中

另外一种方式是使用scope-import, 将依赖分组标识在不同的POM中, 然后在需要的地方引用它们, 实现多继承的效果

1. 建立dependency工程, 该类工程没有Java代码
2. 修改POM文件, <packaging>pom</packaging>
3. 将可能用到的依赖放入dependencyManagement,

<dependencyManagement>

<dependencies>

<dependency>

<groupId>log4j</groupId>

<artifactId>log4j</artifactId>

<version>${log4j.version}</version>

</dependency>

</dependencies>

</dependencyManagement>

1. 在需要引用的模块里(子模块), 用dependencyManagement scope import 的方式引用它

<dependencyManagement>

<dependencies>

<dependency>

<groupId>com.felix.demo</groupId>

<artifactId>depends</artifactId>

<version>1.0-SNAPSHOT</version>

**<type>pom</type> // 必须标识为pom**

**<scope>import</scope> // scope 为 import**

</dependency>

</dependencies>

</dependencyManagement>

1. 在需要引用的模块里(子模块), 明确写出需要的依赖(**不用带版本号**)

<dependencies>

<dependency>

<groupId>log4j</groupId>

<artifactId>log4j</artifactId>

</dependency>

</dependencies>

# spring-boot

## 建立spring-boot项目

New->Module->Maven, 然后修改< parent > 为

<parent>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-parent</artifactId>

<version>2.1.3.RELEASE</version>

</parent>

加入 spring-boot-starter-web 依赖

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-web</artifactId>

</dependency>

创建Application类

~~@Configuration~~

~~@EnableAutoConfiguration~~

~~@ComponentScan~~

@SpringBootApplication

public class Application {

public static void main(String[] args) {

SpringApplication.run(Application.class, args);

}

}

创建Controller类

@Controller

public class MainController {

@RequestMapping("/")

@ResponseBody

String home() {

return "show me";

}

}

# spring-cloud

## 配置spring cloud config server

POM添加对spring-cloud-config-server的依赖

<dependency>

<groupId>org.springframework.cloud</groupId>

<artifactId>spring-cloud-config-server</artifactId>

</dependency>

Applicaiton类添加@EnableConfigServer注解

@SpringBootApplication

@EnableConfigServer

public class Application {

public static void main(String[] args) {

SpringApplication.run(Application.class, args);

}

}

配置可以在3种地方: native, git, svn

在resource目录下建立application.properties文件

spring.application.name=config

spring.profiles.active=native // 配置在本地

spring.cloud.config.server.native.search-locations=file:E:/project/sc1/config/src/main/resources // 指定本地路径, 或classpath路径(当前目录是jar根路径, 也就是resource路径)

server.port=8181 // 指定config服务端口

然后在resource/config/下建立[服务]的[profile]配置, 如: service1-dev.yml, service1-prod.yml

启动config server 直接访问<http://localhost:8001/service1/dev>, 或 <http://localhost:8001/service1-dev.yml> 可以获取配置

问题:

1. maven可以执行编译通过, 但idea上build失败, 代码上显示类和包无法找到, 很可能是因为iml文件与maven的pom信息不匹配导致, 可以尝试点击maven面板的刷新, 或在iml的当前目录执行mvn idea:module

## 建立discovery工程

1. POM文件中包含
   1. org.springframework.boot/spring-boot-starter

它的版本来自于

<parent>  
 <groupId>org.springframework.boot</groupId>  
 <artifactId>spring-boot-starter-parent</artifactId>  
 <version>2.1.4.RELEASE</version>  
 <relativePath/> <!-- lookup parent from repository -->  
</parent>

* 1. org.springframework.cloud/ spring-cloud-starter-netflix-eureka-server

它的版本来自于

<dependencyManagement>  
 <dependencies>  
 <!-- https://mvnrepository.com/artifact/org.springframework.cloud/spring-cloud-dependencies -->  
 <dependency>  
 <groupId>org.springframework.cloud</groupId>  
 <artifactId>spring-cloud-dependencies</artifactId>  
 <version>Greenwich.RELEASE</version>  
 <type>pom</type>  
 <scope>import</scope>  
 </dependency>  
 </dependencies>  
</dependencyManagement>

1. Application类加上注解
   1. @SpringBootApplication
   2. @EnableEurekaServer // 标识我是注册中心
2. application.properties(ym)中配置
   1. 端口: server.port:8000
   2. 自己不参与注册: eureka.client.registerWithEureka: false
   3. 不向注册中心拉取注册信息: eureka.client.fetchRegistry: false
3. chrome: <http://localhost:8000>

## 建立工程Server1作为服务提供者

1. POM文件中包含
   1. org.springframework.boot/spring-boot-starter
   2. org.springframework.cloud/ spring-cloud-starter-netflix-eureka-client
2. Application类加上注解
   1. @SpringBootApplication
   2. @ EnableEurekaClient // 标识我是客户端
3. application.properties(ym)中配置
   1. 端口: server.port:8001
   2. 设定名称(域名): spring.application.name: server1
   3. 设定注册中心: eureka.client.serviceUrl.defaultZone: <http://localhost:8000/eureka>
4. 建立一个RestController, chrome: <http://localhost/test>
   1. 需要确保org.springframework.boot /spring-boot-starter-web依赖

## 建立工程Server2作为服务的使用者(以Ribbon方式调用服务)

1. POM文件中包含
   1. org.springframework.boot/spring-boot-starter
   2. org.springframework.cloud/ spring-cloud-starter-netflix-eureka-client
2. Application类加上注解
   1. @SpringBootApplication
   2. @ EnableEurekaClient // 标识我是客户端
3. application.properties(ym)中配置
   1. 端口: server.port:8002
   2. 设定名称(域名): spring.application.name: server2
   3. 设定注册中心: eureka.client.serviceUrl.defaultZone: <http://localhost:8000/eureka>
4. application类中提供 RestTemplate Bean

@Bean

@LoadBalanced

public RestTemplate restTemplate() { return new RestTemplate(); }

1. 在需要调用的地方, 比如一个Controller的某个方法中
   1. @ Autowired RestTemplate restTemplate
   2. 调用 restTemplate.getForObject("http://server1/test/" + id, String.class);
2. chrome 测试这个Server2的这个restful接口

## 也可以使用 feign来调用Server1的方法

1. POM文件中包含
   1. org.springframework.boot/spring-boot-starter
   2. org.springframework.cloud/ spring-cloud-starter-netflix-eureka-client
   3. org.springframework.cloud/ spring-cloud-starter-openfeign
2. Application类加上注解
   1. @SpringBootApplication
   2. @EnableEurekaClient // 标识我是客户端
   3. @EnableFeignClients // 启用 Feign
3. application.properties(ym)中配置
   1. 端口: server.port:8003
   2. 设定名称(域名): spring.application.name: server3
   3. 设定注册中心: eureka.client.serviceUrl.defaultZone: <http://localhost:8000/eureka>
4. ~~application类中提供 RestTemplate Bean~~

~~@Bean~~

~~@LoadBalanced~~

~~public RestTemplate restTemplate() { return new RestTemplate(); }~~

1. 声明接口feign client接口

@FeignClient(name="service1") // 指定server

public interface TestFeignClient {

@RequestMapping(value="/test/{id}", method = RequestMethod.GET)

public String test(@PathVariable("id") String id);

}

1. 在需要调用的地方, 比如一个Controller的某个方法中
   1. 直接调用 testFeignClient.test(id);
2. chrome 测试这个Server3的这个restful接口

## 实现服务的高可用

只需保证这个服务的名称(spring.application.name)不变, 再在不同的端口或主机上启动即可

## 实现注册中心的高可用

1. 为注册中心设定服务名称
2. 服务中心之间互相设置: eureka.client.serviceUrl.defaultZone 为对方, 如果有多个就用逗号分隔
3. registerWithEureka 和 fetchRegistry 取默认值true
4. 其他Server (即注册中心的客户端) eureka.client.serviceUrl.defaultZone 设定多个注册中心, 用逗号分隔