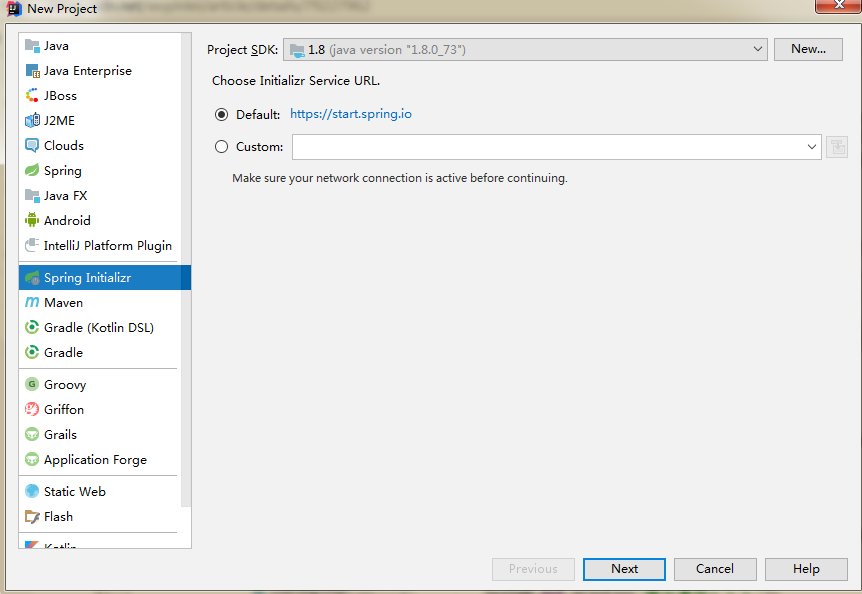
Intellij idea使用Spring Boot构建微服务

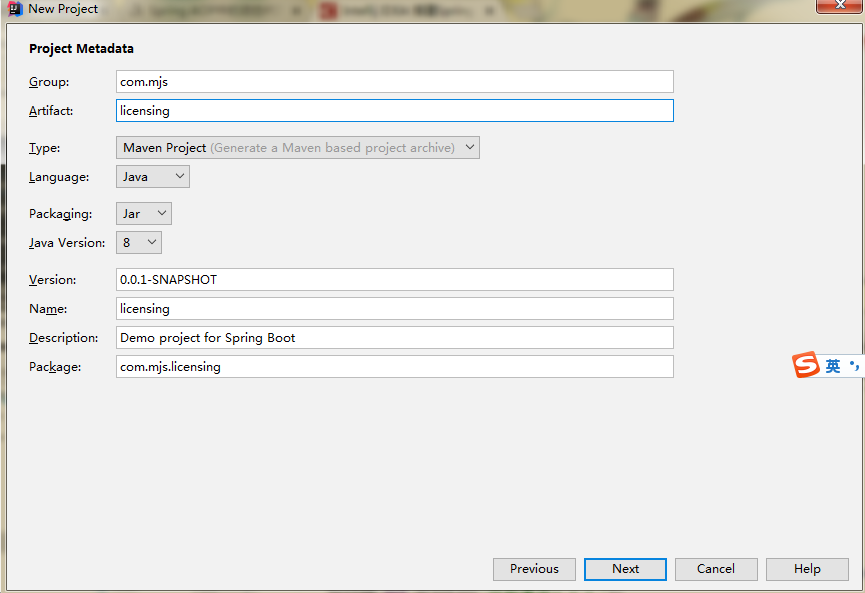
|  |  |  |  |
| --- | --- | --- | --- |
| 版本号 | 日期 | 内容 | 修订人 |
| 0.1 | 2018-08-22 |  | mjs |

# 搭建spring boot项目

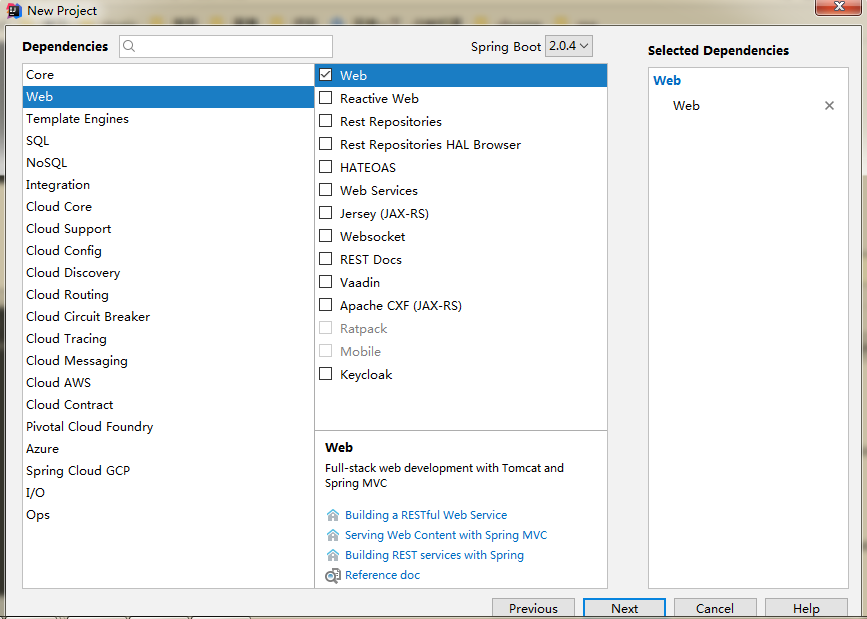
1. File –> New –> Project –>Spring Initialer –> **点击Next** ，如图所示：



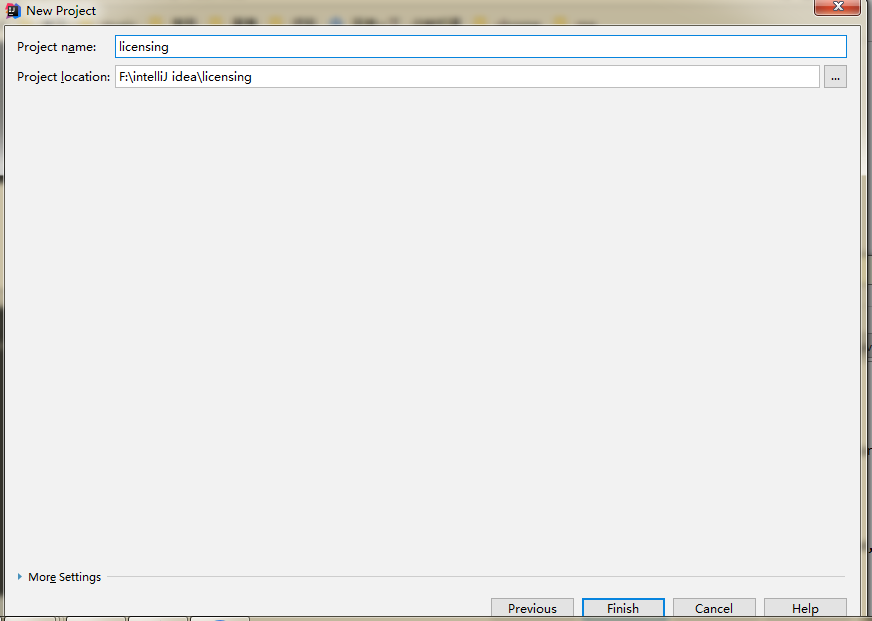
1. 命名**Group** 和 **Artifact**  –>**点击next**



1. 勾选Web—>Web选项，并且选择spring boot版本，点击下一步



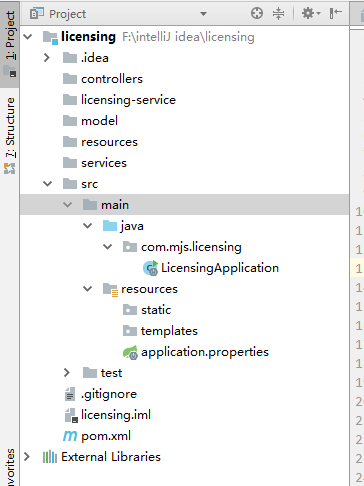
1. 点击下一步，等待下载编译，



1. 删除多余项目目录，[.mvn] [mvnw] [mvnw.cmd]

# 搭建licensing项目

## 1、新增以下目录结构：



## 2、修改pom.xml文件



几个重要部分:

*<!--第一部分 maven拉取spring boot2.0.4版本 Starter 达到简化依赖目的-->*<**parent**>  
 <**groupId**>org.springframework.boot</**groupId**>  
 <**artifactId**>spring-boot-starter-parent</**artifactId**>  
 <**version**>2.0.4.RELEASE</**version**>  
 <**relativePath**/> *<!-- lookup parent from repository -->*</**parent**>

<**dependencies**>  
 *<!--第二部分 拉取spring web工具包  
 包含  
 ■ org.springframework.boot:spring-boot-starter  
 ■ org.springframework.boot:spring-boot-starter-tomcat  
 ■ org.springframework.boot:spring-boot-starter-validation  
 ■ com.fasterxml.jackson.core:jackson-databind  
 ■ org.springframework:spring-web  
 ■ org.springframework:spring-webmvc  
 几部分工具包  
 -->* <**dependency**>  
 <**groupId**>org.springframework.boot</**groupId**>  
 <**artifactId**>spring-boot-starter-web</**artifactId**>  
 </**dependency**>  
 *<!-- 该部分 涵盖测试工具包  
 包含  
 ■ junit:junit  
 ■ org.mockito:mockito-core  
 ■ org.hamcrest:hamcrest-core  
 ■ org.hamcrest:hamcrest-library  
 ■ org.springframework:spring-core(excludes commons-logging:commons-logging)  
 ■ org.springframework:spring-test  
 -->* <**dependency**>  
 <**groupId**>org.springframework.boot</**groupId**>  
 <**artifactId**>spring-boot-starter-test</**artifactId**>  
 <**scope**>test</**scope**>  
 </**dependency**>  
 *<!--第三部分 该部分包含spring boot应用监控工具包 -->* <**dependency**>  
 <**groupId**>org.springframework.boot</**groupId**>  
 <**artifactId**>spring-boot-starter-actuator</**artifactId**>  
 </**dependency**>  
</**dependencies**>  
  
<**build**>  
 <**plugins**>  
 *<!-- maven 插件 简化spring boot构建及部署 -->* <**plugin**>  
 <**groupId**>org.springframework.boot</**groupId**>  
 <**artifactId**>spring-boot-maven-plugin</**artifactId**>  
 </**plugin**>  
 </**plugins**>  
</**build**>

## 3、编写引导类

引导类用户启动和初始化应用程序。



**package** com.mjs.licensing;  
  
**import** org.springframework.boot.SpringApplication;  
**import** org.springframework.boot.autoconfigure.SpringBootApplication;  
  
*/\*\*  
 \** ***@SpringBootApplication*** *告诉spring boot框架，该类为引导类  
 \* 调用该类的main中 SpringApplication.run方法 启动spring boot服务  
 \* 服务核心初始化逻辑都在该类中  
 \*/*@SpringBootApplication  
**public class** LicensingApplication {  
  
 **public static void** main(String[] args) {  
  
 SpringApplication.*run*(LicensingApplication.**class**, args);  
 }  
}

4编写spring boot控制器

控制器类公开了服务端点，并将数据传入的HTTP请求映射到将处理该请求的java方法。

控制类遵循REST 方法 (HTTP请求的一些协议)

控制类使用JSON传输数据



*/\*\*  
 \** ***@RestController*** *告诉spring boot 这是基于REST的服务，它将自动序列化/反序列化服务请求/响应到JSON  
 \** ***@RequestMapping注解*** *告诉spring容器该服务将要公开的HTTP端点 value属性为控制类中公开的所有端点URL的根  
 \*/*@RestController  
@RequestMapping(value = **"/v1/organizations/{organizationId}/licenses"**)  
**public class** LicenseServiceController {  
 */\*\*  
 \* 创建一个Get端点/v1/organizations/{organizationId}/licenses/{licenseId}  
 \*  
 \** ***@param organizationId*** *\** ***@param licenseId*** *\** ***@return*** *\** ***@PathVariable*** *将值映射到方法参数内  
 \*/* @RequestMapping(value = **"/{licenseId}"**, method = RequestMethod.***GET***)  
 **public** License getLicenses(@PathVariable(**"organizationId"**) String organizationId,  
 @PathVariable(**"licenseId"**) String licenseId) {  
 **return new** License()  
 .withId(licenseId)  
 .withProductName(**"Teleco"**)  
 .withLicenseType(**"Seat"**)  
 .withOrganizationId(organizationId);  
 }  
}

4、启动项目

(1)直接执行引导类LicensingApplication的main方法启动;

(2)cmd至项目路径，mvn spring-boot:run 回车启动;

(3)cmd至项目路径，mvn install 回车启动;

命令行构建微服务作为可执行JAR，然后从命令行启动JAR文件

java -jar licensing-0.0.1-SNAPSHOT.jar

# spring boot配置服务器控制配置(confsvr)

1. 引入config相关依赖

<**parent**>  
 <**groupId**>org.springframework.boot</**groupId**>  
 <**artifactId**>spring-boot-starter-parent</**artifactId**>  
 <**version**>1.4.4.RELEASE</**version**>  
 <**relativePath**/> *<!-- lookup parent from repository -->*</**parent**>  
<**dependencyManagement**>  
 <**dependencies**>  
 <**dependency**>  
 <**groupId**>org.springframework.cloud</**groupId**>  
 <**artifactId**>spring-cloud-dependencies</**artifactId**>  
 <**version**>Camden.SR5</**version**>  
 <**type**>pom</**type**>  
 <**scope**>import</**scope**>  
 </**dependency**>  
 </**dependencies**>  
</**dependencyManagement**>  
  
*<!-- 配置服务器将要使用的引导类 -->*<**properties**>  
 <**project.build.sourceEncoding**>UTF-8</**project.build.sourceEncoding**>  
 <**start-class**>com.mjs.confsvr.ConfsvrApplication</**start-class**>  
 <**java.version**>1.8</**java.version**>  
 <**docker.image.name**>johncarnell/tmx-confsvr</**docker.image.name**>  
 <**docker.image.tag**>chapter3</**docker.image.tag**>  
</**properties**>  
<**dependencies**>  
 <**dependency**>  
 <**groupId**>org.springframework.cloud</**groupId**>  
 <**artifactId**>spring-cloud-starter-config</**artifactId**>  
 </**dependency**>  
 <**dependency**>  
 <**groupId**>org.springframework.cloud</**groupId**>  
 <**artifactId**>spring-cloud-config-server</**artifactId**>  
 </**dependency**>  
 <**dependency**>  
 <**groupId**>org.springframework.boot</**groupId**>  
 <**artifactId**>spring-boot-starter-test</**artifactId**>  
 <**scope**>test</**scope**>  
 </**dependency**>  
</**dependencies**>

1. 创建引导类

*/\*\*  
 \** ***@EnableConfigServer*** *使服务成为spring cloud config服务  
 \*/*@SpringBootApplication  
@EnableConfigServer  
**public class** ConfsvrApplication {  
  
 **public static void** main(String[] args) {  
 SpringApplication.*run*(ConfsvrApplication.**class**, args);  
 }  
}

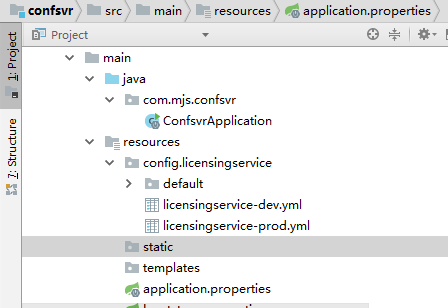
1. 添加config配置至application.properties

**spring.application.name**=**confsvr  
server.port**=**8888***#1、***spring.profiles.active**=**native  
spring.cloud.config.server.native.searchLocations**=**file:F:/intelliJidea/confsvr/src/main/resources/config/licensingservice**

*#spring.profiles.active=native，config server默认从应用的src/main/resource目录下检索配置文件  
#spring.cloud.config.server.native.searchLocations=file:F:/properties/属性来指定配置文件的位置  
#  
#3、git控制方式  
# spring.cloud.config.server.git.uri 配置git仓库位置  
# spring.cloud.config.server.git.search-paths 配置仓库路径下的相对搜索位置，可以配置多个，","隔开  
# spring.cloud.config.label 分支 默认master  
# spring.cloud.config.server.git.username 访问git仓库的用户名  
# spring.cloud.config.server.git.password 访问git仓库的用户密码  
  
# 注：URL与配置文件的映射关系  
# (1)/{application}/{profile}[/{label}]  
# (2)/{application}-{profile}.yml eg：didispace-dev.yml  
# (3)/{label}/{application}-{profile}.yml eg：/master/didispace-dev.yml  
# (4)/{application}-{profile}.properties eg：didispace-dev.properties  
# (5)/{label}/{application}-{profile}.properties eg：/master/didispace-dev.properties  
  
#spring.cloud.config.server.git.uri=https://github.com/mjs154/springbootconfig.git  
#spring.cloud.config.server.git.search-paths=config  
#spring.cloud.config.label=master  
#spring.cloud.config.server.git.username=mjs154@126.com  
#spring.cloud.config.server.git.password=\*\*\*\*\*\**

1. 添加配置文件(说明在3配置文件注释内)

如图所示



1. 启动项目并访问

通过POSTMAN访问URL：<http://localhost:8888/licensingservice/dev>

注：(1) <http://localhost:8888/> 为本地ip及端口

(2) licensingservice *application属性(详细至3注释)*

(3) dev *profile属性(详细至3注释)*

返回结果：

{"name":"licensingservice",

"profiles":["dev"],

"label":null,

"version":null,

"state":null,

"propertySources":

[{"name":"file:F:/intelliJidea/confsvr/src/main/resources/config/licensingservice/licensingservice-dev.yml","source":{"dev.name":"dev"}}]

}

若通过git托管配置文件

访问<http://localhost:8888/didispace/dev>

返回结果:

{"name":"didispace",

"profiles":["dev"],

"label":"master",

"version":"ce99092f0ea025944cd438f6f6c49151f46b69ac",

"state":null,

"propertySources":[{"name":"https://github.com/mjs154/springbootconfig.git/config/didispace-dev.properties","source":{"from":"git-dev-1.0"}},{"name":"https://github.com/mjs154/springbootconfig.git/config/didispace.properties","source":{"from":"git-default-1.0"}}]

}

# Spring boot config和spring boot客户端集成

Licensing服务通过访问confsvr配置服务，confsvr通过存储库获取配置信息，并且将信息返回至licensing服务。

1. licensing添加配置服务的依赖

*<!--第四部分 增加 与confsvr配置服务交互的依赖 -->*<**dependency**>  
 <**groupId**>org.springframework.cloud</**groupId**>  
 <**artifactId**>spring-boot-config-client</**artifactId**>  
</**dependency**>

1. 配置licensing服务可使用confsvr服务

添加bootstrap.properties文件

*#指定licensing服务名称，便于spring cloud config客户端知道正在查找那个服务***spring.application.name**=**didispace***#指定服务应该运行的默认profile oprofile映射到环境***spring.profiles.active**=**default***#指定spring cloud config服务器的位置***spring.cloud.config.url**=**http://localhost:8888**

注：bootstrap.properties和application.properties的区别：

1. bootstrap.properties先于application.properties被加载；
2. 当使用Spring Cloud Config Server时，需要将spring.application.name和spring.cloud.config.server.git.uri写入bootstrap. properties。
3. 一些encryption/decryption（加密/解密）存储在bootstrap. properties里面。
4. bootstrap. properties被Spring ApplicationContext的父类加载，这个类先于加载application. properties的ApplicatonContext启动。
5. 接收配置服务的属性值

略。