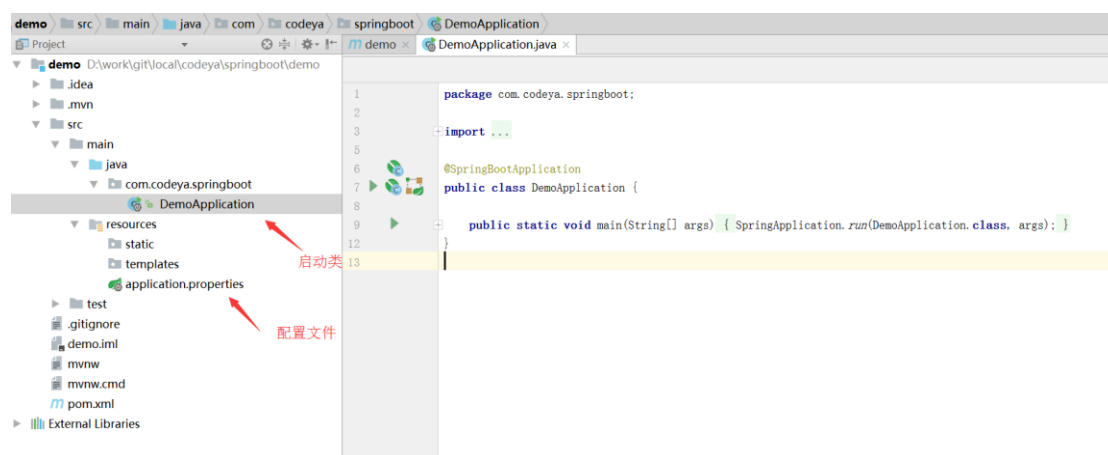


Spring Boot 入门

参考官方文档

<https://docs.spring.io/spring-boot/docs/1.5.8.RELEASE/reference/htmlsingle>

1 介绍



SpringBoot 可以帮助开发者更容易的创建基于 Spring 的应用程序和服务。

默认集成了 Tomcat 容器，所以可以独立运行，可以不依赖于外部容器。当然也可以达成 war 包部署提供了一种从 main()方法启动的便捷方式,也就说 :只需要在 main 方法所在的类右键 Run 即可启动应用。相比于传统的 spring 应用里需要各种配置，比如 web.xml 加载 spring、spring mvc，扫描路径等，SpringBoot 只需要很少的几个配置或者注解就可以达到。整合了很多框架，pom.xml 里只需要引入需要的 starter 依赖就可以了。

1.1 版本

Spring Boot 1.5.8 (目前我们使用的):

此版本默认使用 1.7 的 jdk (官方建议使用 1.8), 4.3.12.RELEASE 版本的 spring 框架。

内嵌 Tomcat 8

Apache Maven 3.2 或更高版本

最新的 Spring Boot 2.0.0.BUILD-SNAPSHOT :
[Java 8](#) and Spring Framework 5.0.2.RELEASE or above.

1.2 启动类

```
@SpringBootApplication
public class DemoApplication {

    public static void main(String[] args) { SpringApplication.run(DemoApplication.class, args); }
}
```

使用@SpringBootApplication 注解

1.3 配置文件

默认使用的 application.properties/ application.yml

yml 文件书写时注意,冒号和 value 之间有个空格

```
server:
  port: 8081
  context-path: /demo
```

必须要有个空格

上面的写法相当于 properties 文件里的：

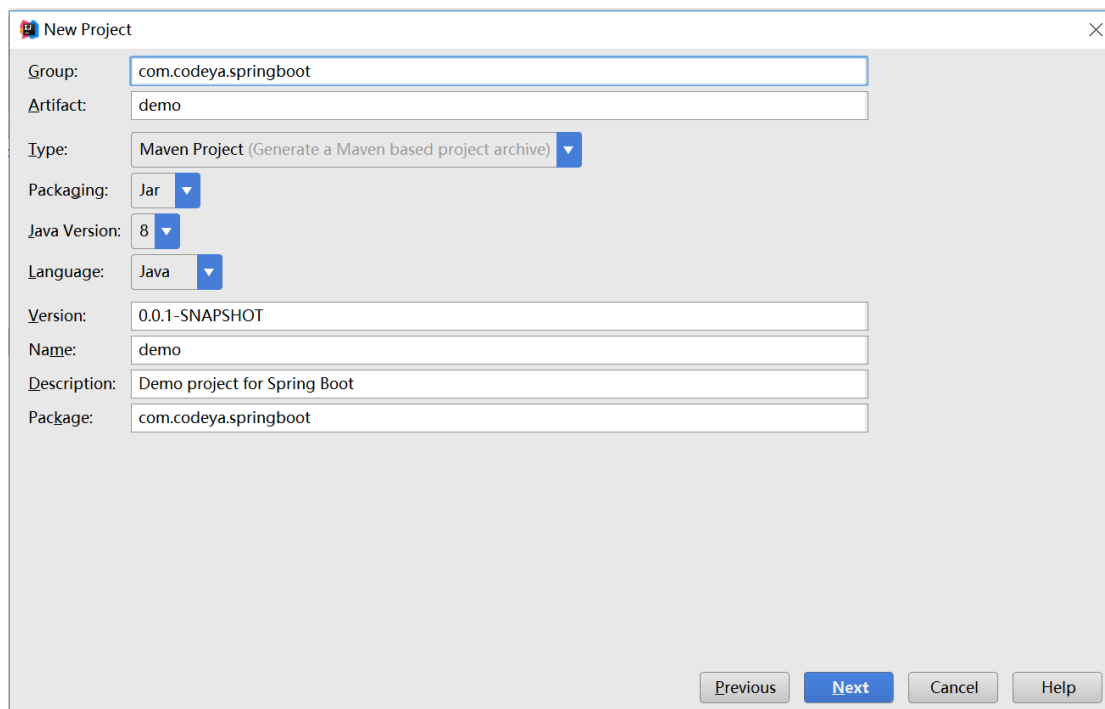
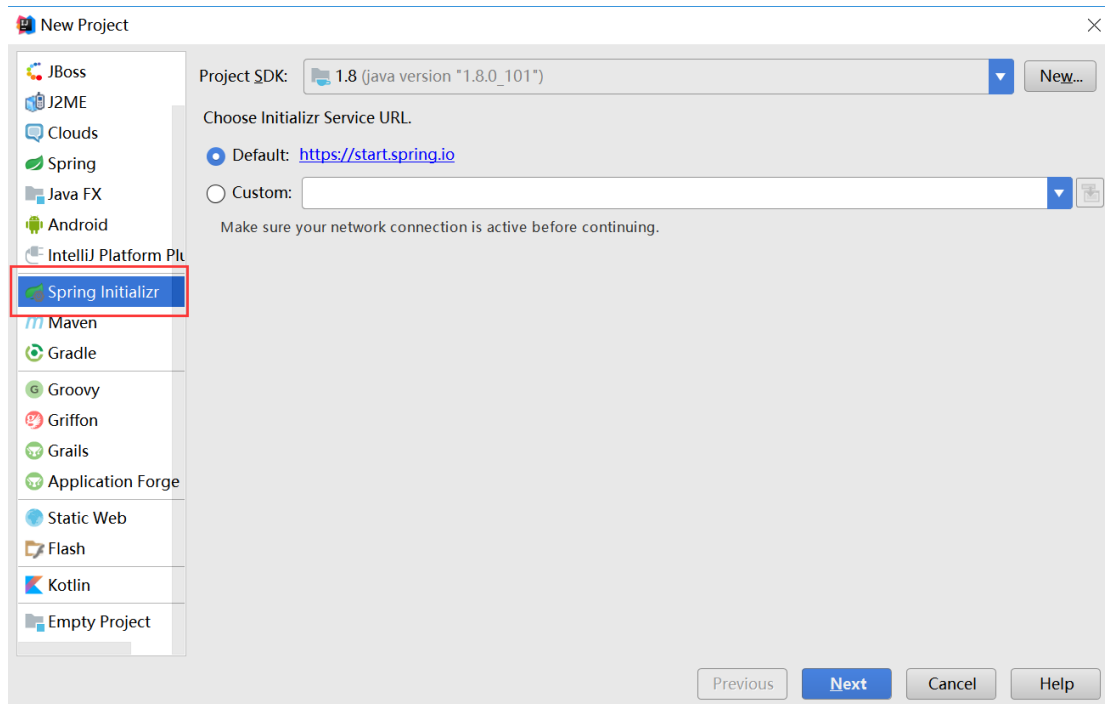
server.port=8081

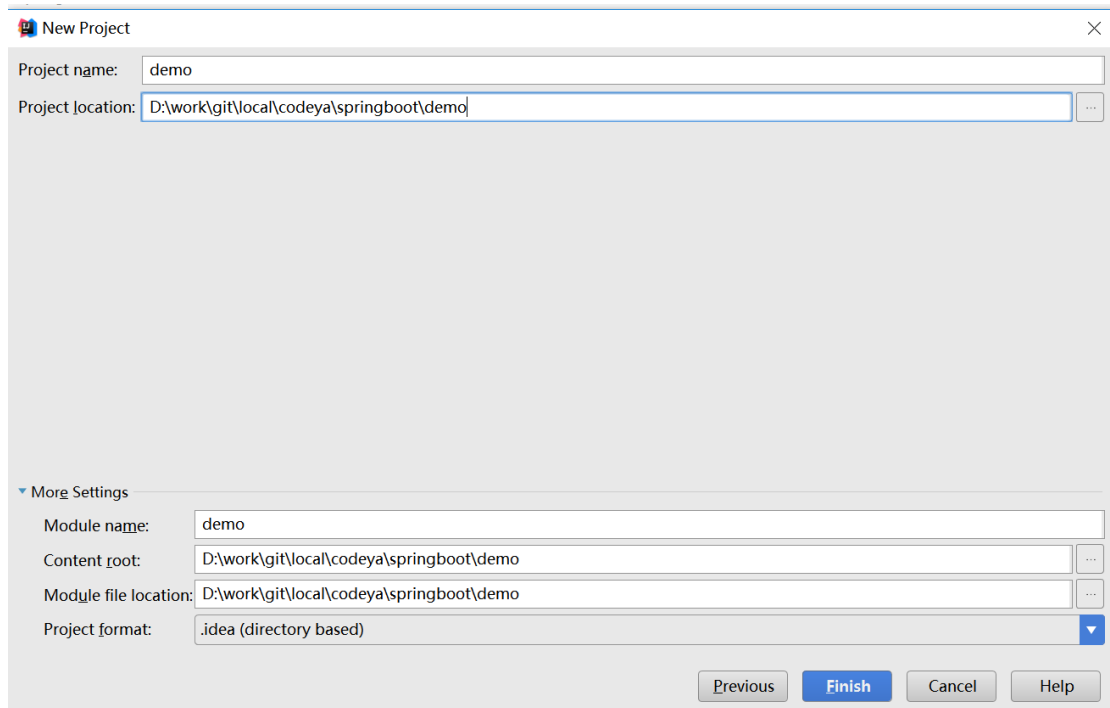
server.context-path=/demo

2 第一个 Spring Boot web 应用

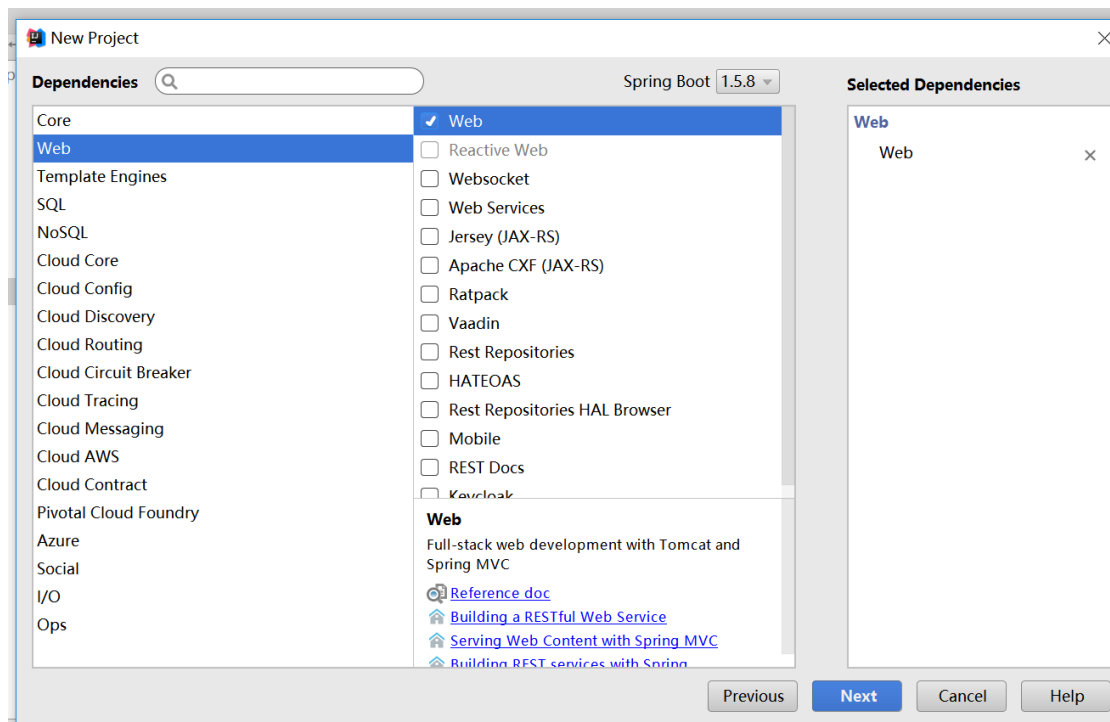
2.1 idea 创建 demo 应用

idea 自带 file->new->project->Spring Initializr 然后 next。。 next。。。

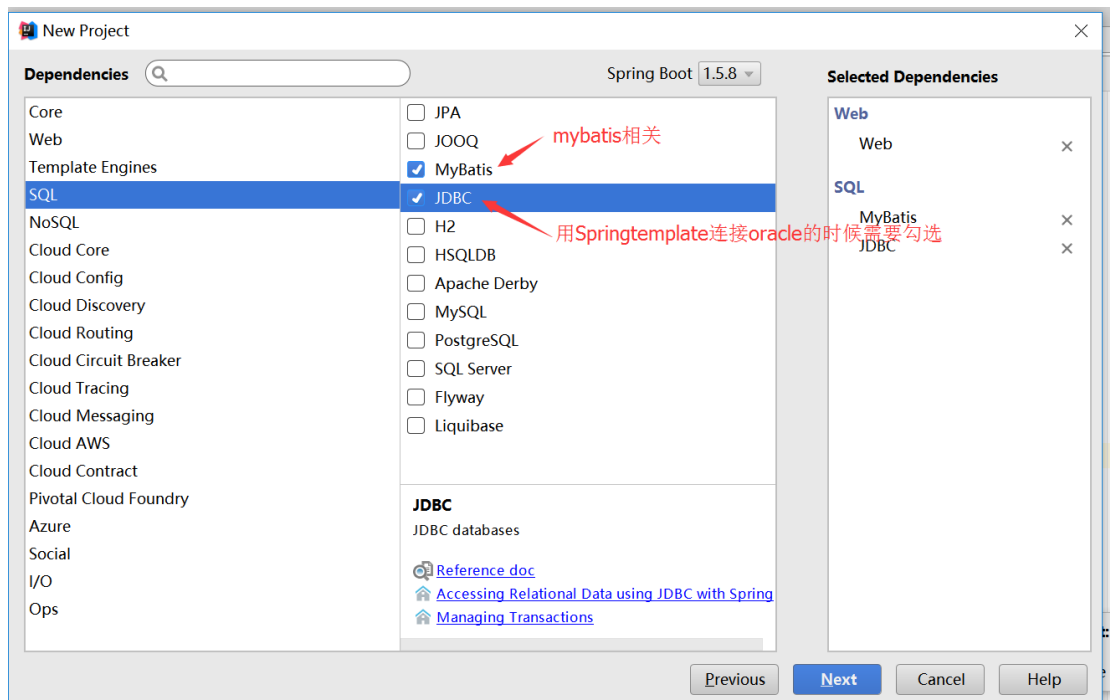




如果是搭一个最简单的 web 应用。只需要勾选如下 Web 即可：

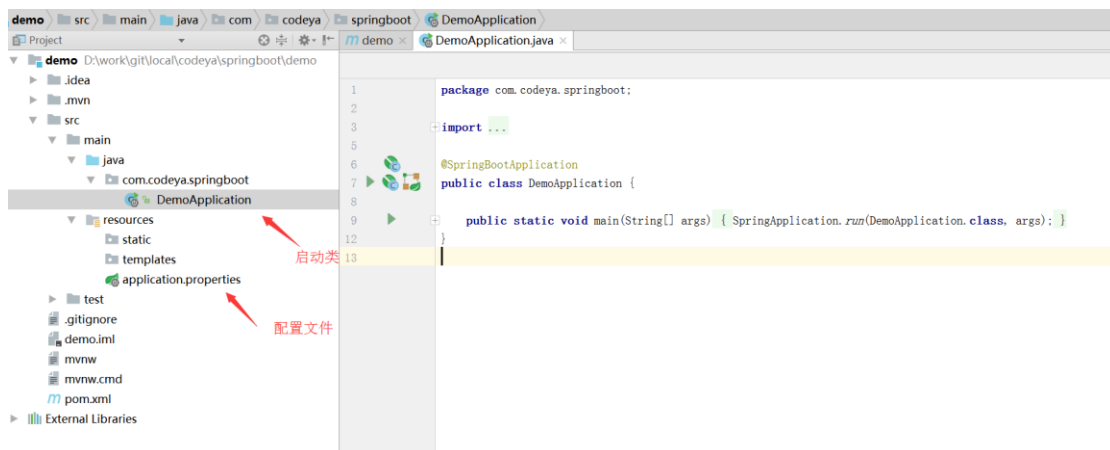


如果使用 mybatis 则需要勾选下图的 MyBatis，如果使用 jdbcTemplate，则需要勾选下图的 JDBC---此时需要在创建好的工程的配置文件里设置相应的数据源。



也可以不勾选，直接在 pom 文件里添加对应的依赖即可。

2.2 建好的应用结构



正常启动类在相对于其他目录的较高层目录。。如果修改启动类的路径，需要修改 `@SpringBootApplication` 的 `scanBasePackages`，具体可参考[后面的章节](#)

2.3 Pom.xml

```
<parent>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-parent</artifactId>
  <version>1.5.8.RELEASE</version>
  <relativePath/> <!-- lookup parent from repository -->
</parent>

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

<dependencies>

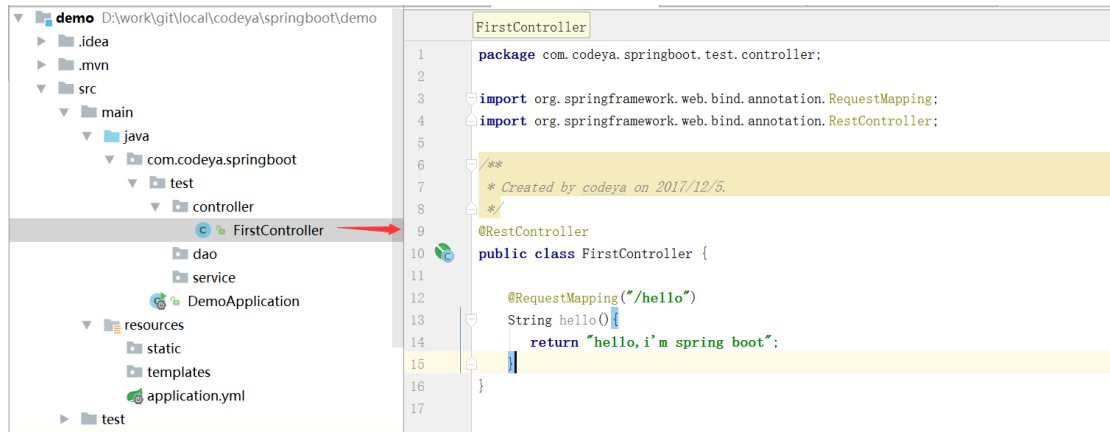
  <dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-web</artifactId>
  </dependency>
  <dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-test</artifactId>
    <scope>test</scope>
  </dependency>
</dependencies>

<build>
  <plugins>
    <plugin>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-maven-plugin</artifactId>
    </plugin>
  </plugins>
</build>

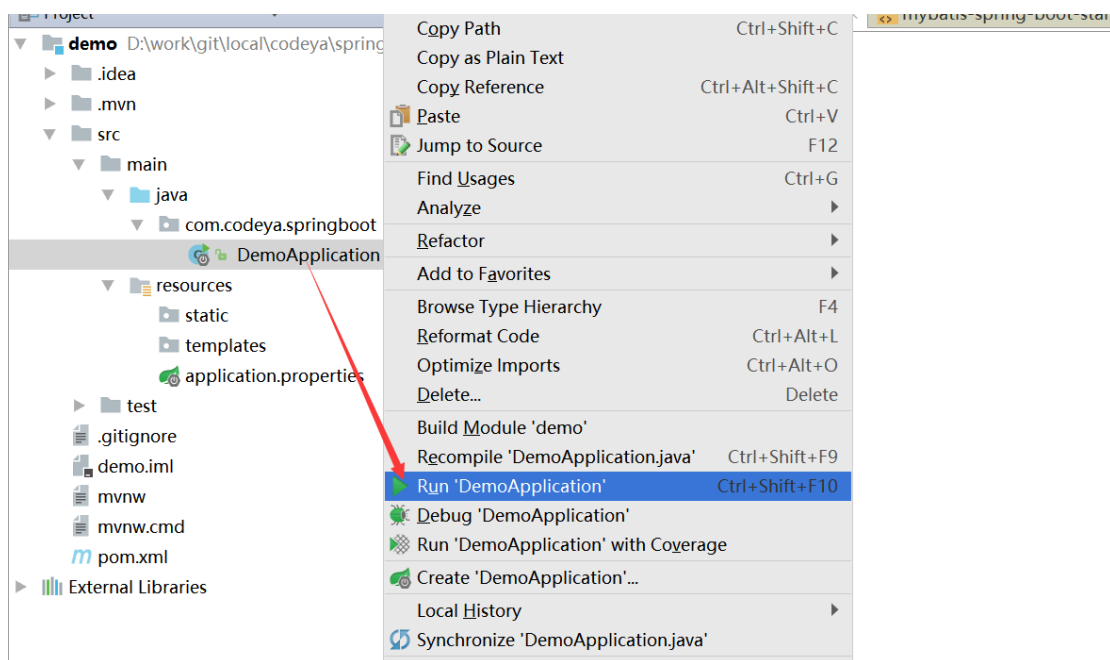
</project>
```

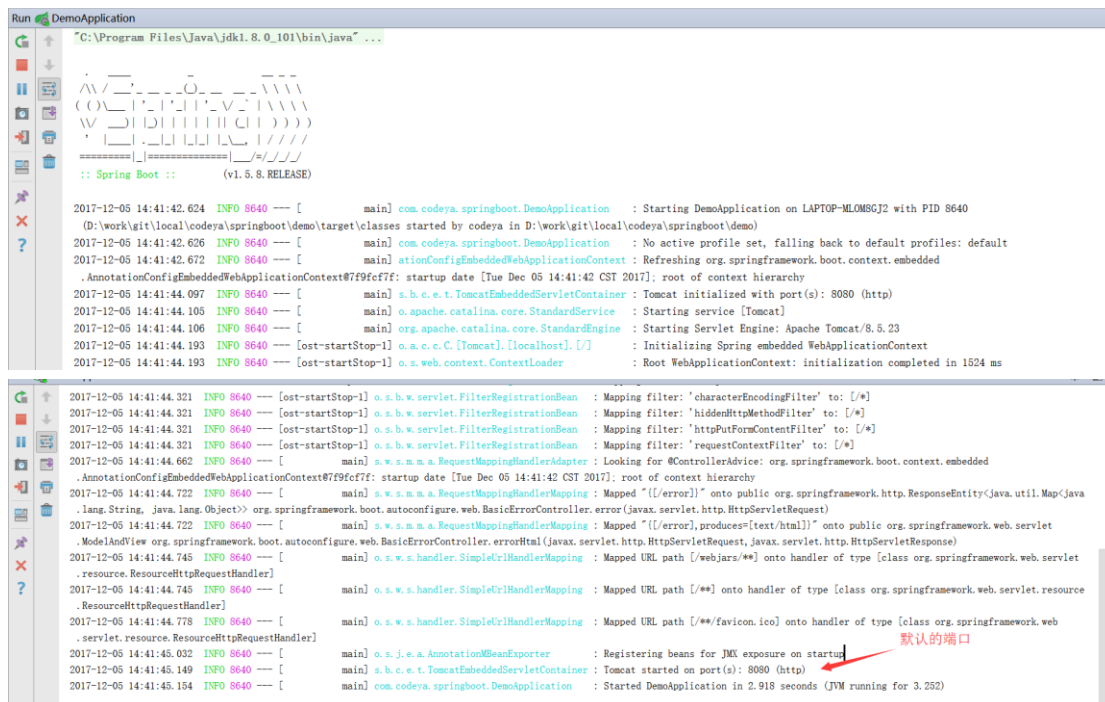
注意：如果网络不好，可能会打不开上面的界面。也可以自行创建一个普通的 maven 工程。。然后参考上面的 pom 文件进行修改：1.修改<parent>节点 2.增加 springboot 的相关依赖 3.<build>节点里增加 spring-boot-maven-plugin，这样打出来的才是一个可执行的 jar

2.4 增加一个 controller 进行测试



2.5 启动应用





```
Run DemoApplication
"C:\Program Files\Java\jdk1.8.0_101\bin\java" ...

:: Spring Boot :: (v1.5.8.RELEASE)

2017-12-05 14:41:42.624 INFO 9640 --- [main] com.codeya.springboot.DemoApplication : Starting DemoApplication on LAPTOP-MLOM8GJ2 with PID 9640
(D:\work\git\local\codeya\springboot\demo\target\classes started by codeya in D:\work\git\local\codeya\springboot\demo)
2017-12-05 14:41:42.626 INFO 9640 --- [main] com.codeya.springboot.DemoApplication : No active profile set, falling back to default profiles: default
2017-12-05 14:41:42.672 INFO 9640 --- [main] ationConfigEmbeddedWebApplicationContext : Refreshing org.springframework.boot.context.embedded
.AnnotationConfigEmbeddedWebApplicationContext@7f9fc7f7: startup date [Tue Dec 05 14:41:42 CST 2017]; root of context hierarchy
2017-12-05 14:41:44.097 INFO 9640 --- [main] s.b.c.e.t.TomcatEmbeddedServletContainer : Tomcat initialized with port(s): 8080 (http)
2017-12-05 14:41:44.105 INFO 9640 --- [main] o.apache.catalina.core.StandardService : Starting service [Tomcat]
2017-12-05 14:41:44.106 INFO 9640 --- [main] org.apache.catalina.core.StandardEngine : Starting Servlet Engine: Apache Tomcat/8.5.23
2017-12-05 14:41:44.193 INFO 9640 --- [ost-startStop-1] o.a.c.c.C.[Tomcat].[localhost].[/] : Initializing Spring embedded WebApplicationContext
2017-12-05 14:41:44.193 INFO 9640 --- [ost-startStop-1] o.s.web.context.ContextLoader : Root WebApplicationContext: initialization completed in 1524 ms

2017-12-05 14:41:44.321 INFO 9640 --- [ost-startStop-1] o.s.b.w.servlet.FilterRegistrationBean : Mapping filter: 'characterEncodingFilter' to: [//*]
2017-12-05 14:41:44.321 INFO 9640 --- [ost-startStop-1] o.s.b.w.servlet.FilterRegistrationBean : Mapping filter: 'hiddenHttpMethodFilter' to: [//*]
2017-12-05 14:41:44.321 INFO 9640 --- [ost-startStop-1] o.s.b.w.servlet.FilterRegistrationBean : Mapping filter: 'httpPutFormContentFilter' to: [//*]
2017-12-05 14:41:44.321 INFO 9640 --- [ost-startStop-1] o.s.b.w.servlet.FilterRegistrationBean : Mapping filter: 'requestContextFilter' to: [//*]
2017-12-05 14:41:44.662 INFO 9640 --- [main] s.w.s.m.m.a.RequestMappingHandlerAdapter : Looking for @ControllerAdvice: org.springframework.boot.context.embedded
.AnnotationConfigEmbeddedWebApplicationContext@7f9fc7f7: startup date [Tue Dec 05 14:41:42 CST 2017]; root of context hierarchy
2017-12-05 14:41:44.722 INFO 9640 --- [main] s.w.s.m.m.a.RequestMappingHandlerMapping : Mapped "([/error])" onto public org.springframework.http.ResponseEntity<java.util.Map<java
.lang.String, java.lang.Object>> org.springframework.boot.autoconfigure.web.BasicExceptionHandler.error(javax.servlet.http.HttpServletRequest)
2017-12-05 14:41:44.722 INFO 9640 --- [main] s.w.s.m.m.a.RequestMappingHandlerMapping : Mapped "([/error],produces=[text/html])" onto public org.springframework.web.servlet
.ModelAndView org.springframework.boot.autoconfigure.web.BasicExceptionHandler.errorHtml(javax.servlet.http.HttpServletRequest, javax.servlet.http.HttpServletResponse)
2017-12-05 14:41:44.745 INFO 9640 --- [main] o.s.w.s.handler.SimpleUrlHandlerMapping : Mapped URL path [/webjars/*] onto handler of type [class org.springframework.web.servlet
.ResourceHandler]
2017-12-05 14:41:44.745 INFO 9640 --- [main] o.s.w.s.handler.SimpleUrlHandlerMapping : Mapped URL path [/**] onto handler of type [class org.springframework.web.servlet
.ResourceHandler]
2017-12-05 14:41:44.778 INFO 9640 --- [main] o.s.w.s.handler.SimpleUrlHandlerMapping : Mapped URL path [/**/favicon.ico] onto handler of type [class org.springframework.web
.servlet.ResourceHandler]
2017-12-05 14:41:45.032 INFO 9640 --- [main] o.s.j.e.a.AnnotationBeanExporter : Registering beans for JMX exposure on startup
2017-12-05 14:41:45.149 INFO 9640 --- [main] s.b.c.e.t.TomcatEmbeddedServletContainer : Tomcat started on port(s): 8080 (http)
2017-12-05 14:41:45.154 INFO 9640 --- [main] com.codeya.springboot.DemoApplication : Started DemoApplication in 2.918 seconds (JVM running for 3.252)
```

默认的端口是 8080。。如需修改，请在 application.yml 里修改

server:

port: 8080 #如果此处不配置，则使用默认的 8080 端口。。

2.6 测试



3 支持 jdbcTemplate

3.1 Pom.xml 增加依赖

```
<dependency>
<groupId>org.springframework.boot</groupId>
<artifactId>spring-boot-starter-jdbc</artifactId>
```



```

</dependency>
<dependency>
    <groupId>com.oracle</groupId>
    <artifactId>ojdbc7</artifactId>
</dependency>

```

3.2 配置文件里增加数据源信息

```

spring:
  datasource:
    driver-class-name: oracle.jdbc.driver.OracleDriver
    url: jdbc:oracle:thin:@ip:端口:sid
    username: 用户名
    password: 密码

```

恭喜你，下面可以使用 Autowired 注入 JdbcTemplate 了

3.3 在 dao 里使用 JdbcTemplate

```

@Repository
public class StaffDao {

    @Autowired
    JdbcTemplate template;

    public Staff findOne(String id) {
        return template.queryForObject(sql: "select staff_id,name,login_name from staff s where s.staff_id=?", new Object[] {id}, new RowMapper<Staff>() {
            @Override
            public Staff mapRow(ResultSet resultSet, int i) throws SQLException {
                System.out.println("result:" + resultSet.getString(columnLabel: "staff_id"));
                Staff staff = new Staff(resultSet.getString(columnLabel: "staff_id"), resultSet.getString(columnLabel: "name"), resultSet.getString(columnLabel: "login_name"));
                return staff;
            }
        });
    }
}

```

3.4 测试

可以增加自己的 controller 类进行测试

```

@Service
public class StaffService implements StaffAPI {

    @Autowired
    StaffDao dao;

    public Staff findOne_by_jdbctemplate(String staffId) {
        return dao.findOne(staffId);
    }
}

```

```

@RestController
@RequestMapping("/staff")
public class StaffController {
    @Autowired
    StaffAPI service;

    @RequestMapping(value = "/jdbc/{id}")
    Staff findOne_by_jdbctemplate(@PathVariable("id") String staffId)
    {
        return service.findOne_by_jdbctemplate(staffId);
    }
}

```

The screenshot shows a REST client interface with the following details:

- Method:** GET
- URL:** http://localhost:8080/staff/jdbc/11
- Params:** (empty)
- Buttons:** Send, Save
- Authorization:** No Auth
- Body:** {
 - "id": "11",
 - "name": "Admin",
 - "loginName": "Admin"
 }
- Status:** 200 OK
- Time:** 960
- Response Format:** JSON

4 支持 MyBatis

4.1 Pom.xml 里增加依赖

```

<dependency>
    <groupId>org.mybatis.spring.boot</groupId>
    <artifactId>mybatis-spring-boot-starter</artifactId>
    <version>1.3.1</version>
</dependency>

```

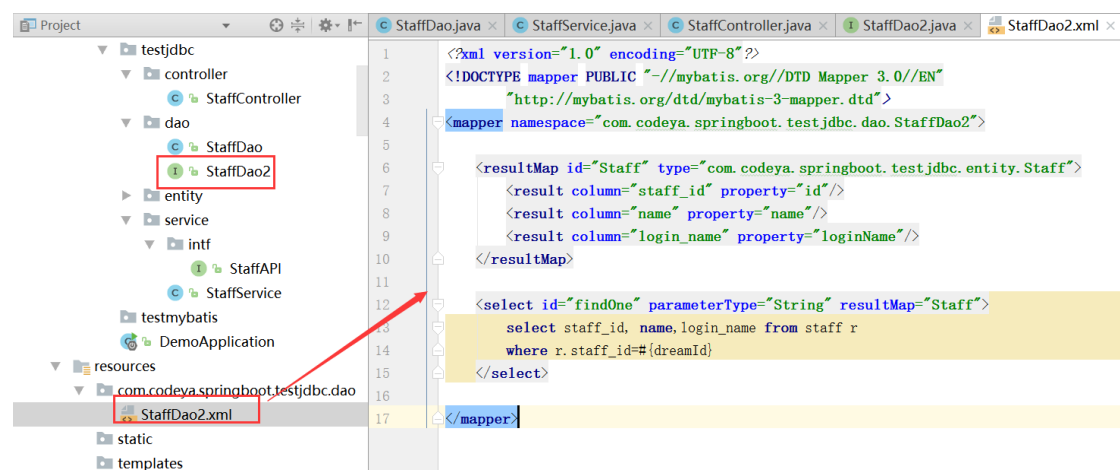
和使用 jdbctemplate 一样，需要在配置文件里存在需要的数据源信息。

4.2 增加 dao 接口

```
public interface StaffDao2 {  
    public Staff findOne(String id);  
}
```

4.3 增加接口对应的 Mapper

在 resources 的同级目录增加同名的 xml 文件。。当 mapper xml 与 mapper class 不在同一个目录下时需要 在 配置 文件 指定 mybatis.mapper-locations ， 比如：
mybatis.mapper-locations=classpath*:com/codeya/springboot/**/dao/*.xml



4.4 启动类里增加@MapperScan

basePackage 指定 mybatis 接口对应的包的位置

```
@SpringBootApplication  
@MapperScan(basePackages="com.codeya.springboot.**.dao")  
public class DemoApplication {  
  
    public static void main(String[] args) { SpringApplication.run(DemoApplication.class,  
    }  
}
```

4.5 测试

```
@Service
public class StaffService implements StaffAPI {

    @Autowired
    StaffDao dao;

    public Staff findOne_by_jdbctemplate(String staffId) {
        return dao.findOne(staffId);
    }

    @Autowired
    StaffDao2 dao2;

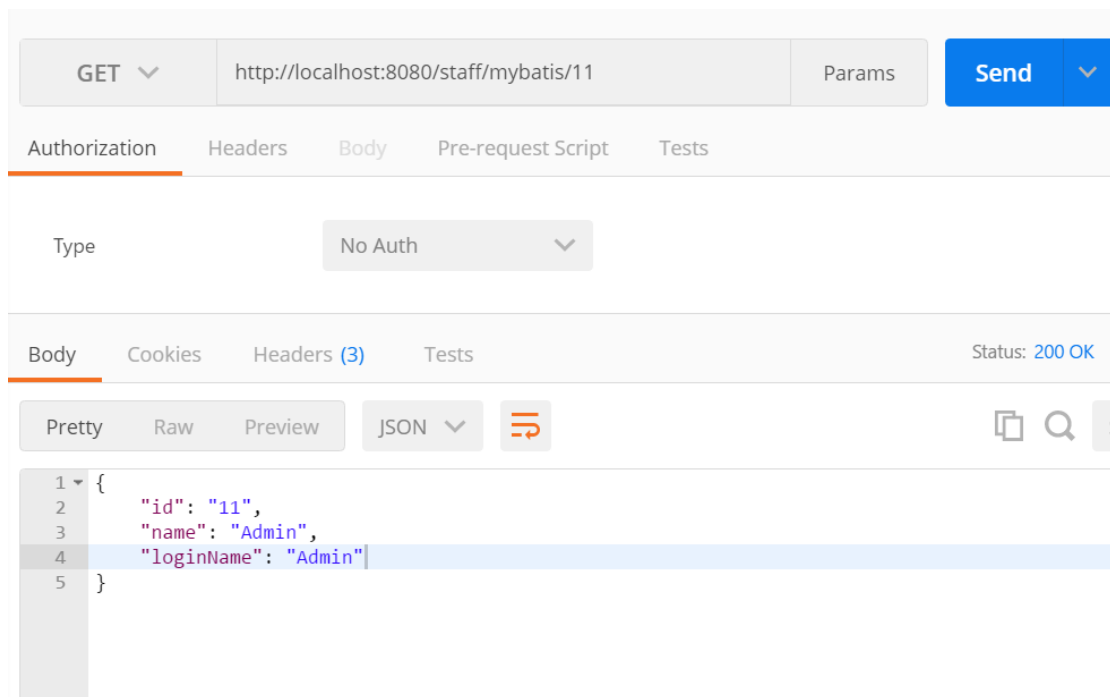
    public Staff findOne_by_mybatis(String staffId) {
        return dao2.findOne(staffId);
    }
}

@RestController
@RequestMapping("/staff")
public class StaffController {

    @Autowired
    StaffAPI service;

    @RequestMapping(value = "/jdbc/{id}")
    Staff findOne_by_jdbctemplate(@PathVariable("id") String staffId)
    {
        return service.findOne_by_jdbctemplate(staffId);
    }

    @RequestMapping(value = "/mybatis/{id}")
    Staff findOne_by_mybatis(@PathVariable("id") String staffId)
    {
        return service.findOne_by_mybatis(staffId);
    }
}
```



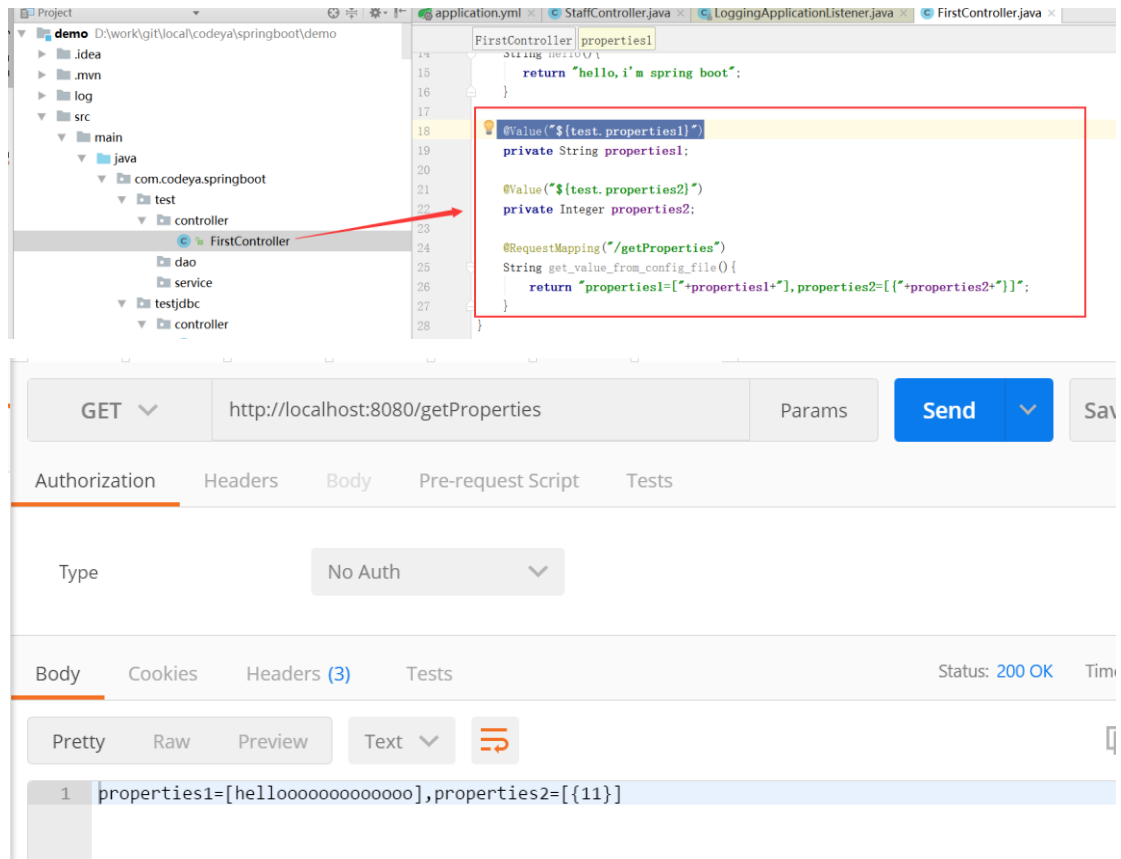
5 读取配置文件里的属性

使用`@Value("${属性名}")`获取配置文件里的属性

举例：

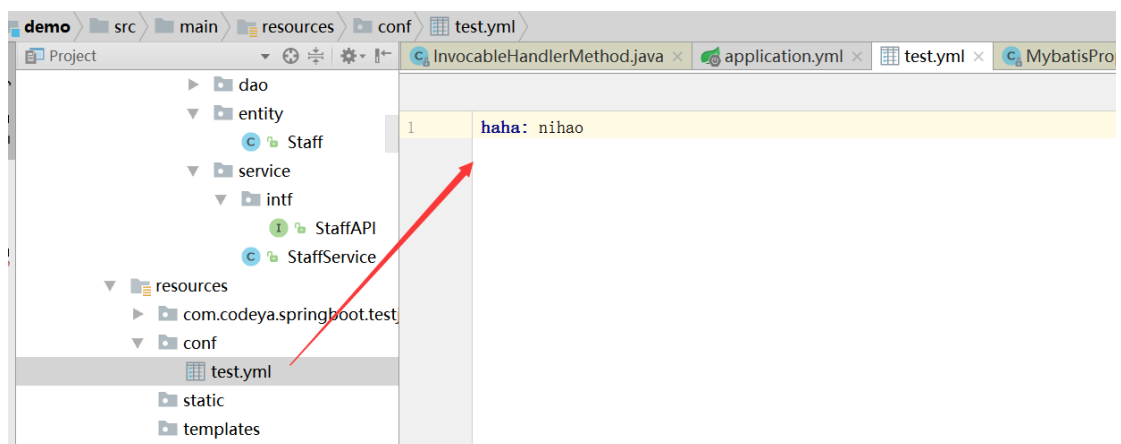
配置文件里增加两个属性

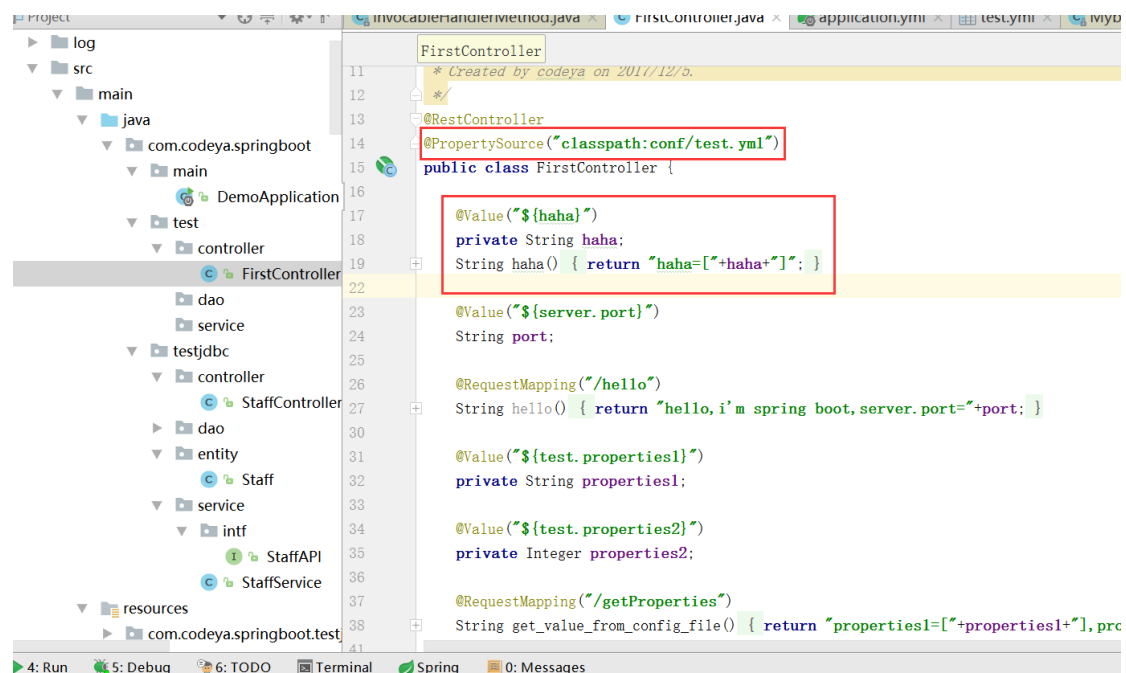




5.1 自定义配置文件

使用@PropertySource 来引入配置文件。。此注解需要与@Configuration 配合使用。





6 日志

SpringBoot 默认使用的 logback。默认记录到 INFO 级别,即记录 ERROR, WARN 和 INFO 级别的信息。如某路径需输出 debug 级别的日志,需要在配置文件里定义 logging.level.包路径=DEBUG,比如下图的 com.example.demo: debug 表示 com.example.demo 包的日志输出级别是 debug logging.level.root 表示该应用总的日志开关,默认是 INFO。

因为输出日志文件需要消耗 I/O 性能,生产环境请将所有日志开关设置为 ERROR。

默认不输出到文件,如需输出文件,需要在配置文件中设定 logging.file 属性,此属性可定义相对路径或绝对路径。

日志文件每达到 10M 就会被分割。

logging.pattern.console 是定义控制台显示日志时的格式

logging.pattern.file 是定义文件记录日志时的格式

这两个值默认不需要设置,但默认的格式不会输出行号。。如需输出行号,可以改成下图里的格式。。

其中 %L 表示行号

```
application.yml x
10
11 logging:
12   file: log/log.log
13   level:
14     root: info
15     com.codeya.springboot: debug
16   pattern:
17     console: -%clr(%d{yyyy-MM-dd HH:mm:ss.SSS}){faint} %clr(${LOG_LEVEL_PATTERN:-%5p}) %clr(${PID:- }){magenta} %clr(---){fain
18     file: -%d{yyyy-MM-dd HH:mm:ss.SSS} ${LOG_LEVEL_PATTERN:-%5p} ${PID:- } --- [%t] %-40.40logger{39}:%L:%m%n${LOG_EXCEPTION_
19
20
```

logging:

file: log/log.log

level:

root: info

com.codeya.springboot: debug

pattern:

console: -%clr(%d{yyyy-MM-dd

HH:mm:ss.SSS}){faint} %clr(\${LOG_LEVEL_PATTERN:-%5p}) %clr(\${PID:- }){magenta} %clr(---){faint} %clr([%15.15t]){faint} %clr(%-40.40logger{39}:%L){cyan} %clr(:){faint} %m%n\${LOG_EXCEPTION_CONVERSION_WORD:-%wEx}

file: -%d{yyyy-MM-dd HH:mm:ss.SSS} \${LOG_LEVEL_PATTERN:-%5p} \${PID:- } ---

[%t] %-40.40logger{39}:%L:%m%n\${LOG_EXCEPTION_CONVERSION_WORD:-%wEx}

7 测试

SpringBootTest 里的 classes 指定的是启动类

```
@RunWith(SpringRunner.class)
@SpringBootTest(classes = DemoApplication.class)
public class StaffServiceTest /*extends DemoApplicationTests*/{
    @Autowired
    StaffService service;

    @Test
    public void findOne_by_jdbctemplate() throws Exception {
        Staff staff = service.findOne_by_jdbctemplate(staffId: "11");
        System.out.println("-----findOne_by_jdbctemplate"+staff.toString());
    }

    @Test
    public void findOne_by_mybatis() throws Exception {
        Staff staff = service.findOne_by_mybatis(staffId: "11");
        System.out.println("-----findOne_by_mybatis"+staff.toString());
    }
}
```

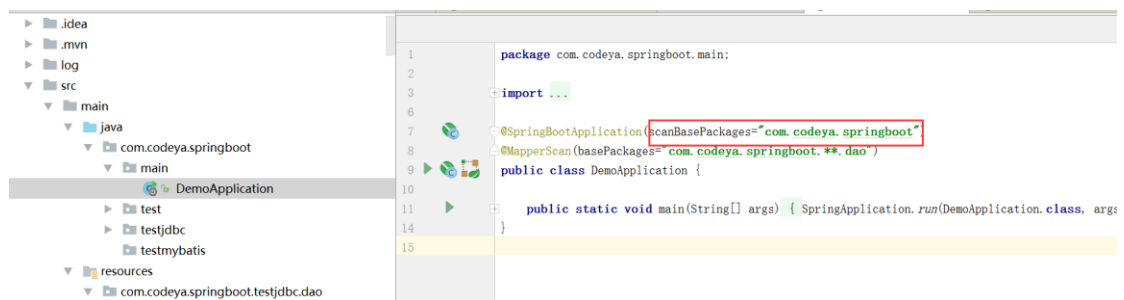

8 扩展

8.1 如果启动类所在目录为较底层目录

正常启动类在相对于其他目录的较高层目录（左图），如果将启动类加到一个底层目录，比如新增的 main 目录下会怎样呢？



如果启动类里不做改动，则原来的路径访问都会报 404.。这是因为默认的扫描路径是启动类的子目录.。此时需要手动指定启动类里的 scanBasePackages

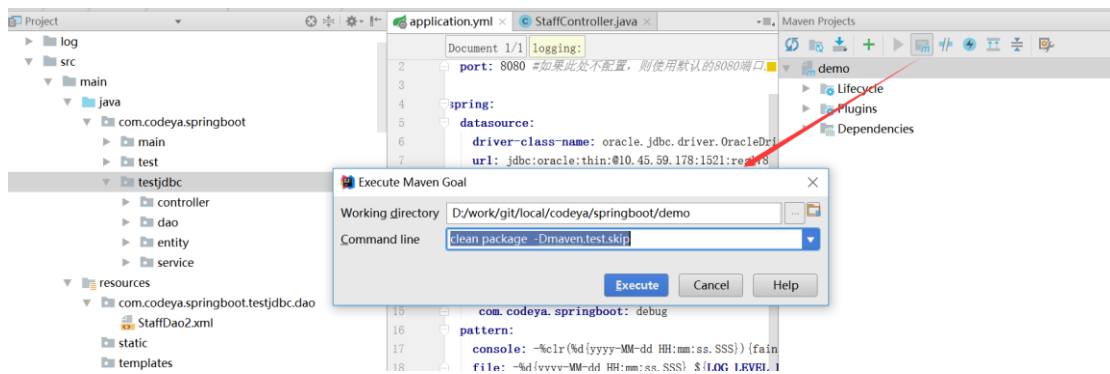


8.2 打包+运行

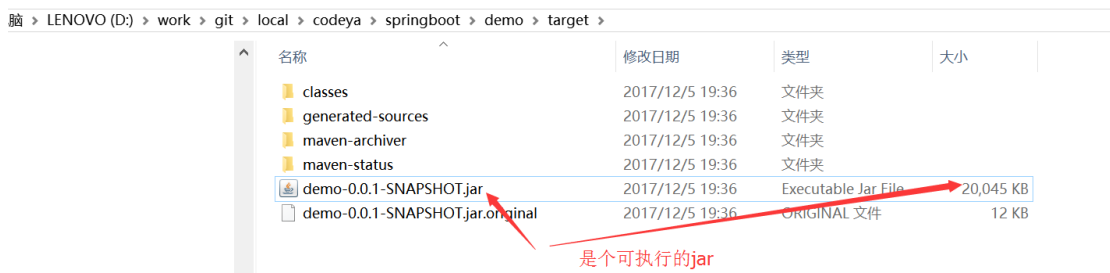
8.2.1 idea 打包

idea 见下图 ,使用的命令 `clean package -Dmaven.test.skip` .。也可以使用命令行 `mvn clean package -Dmaven.test.skip` 进行打包

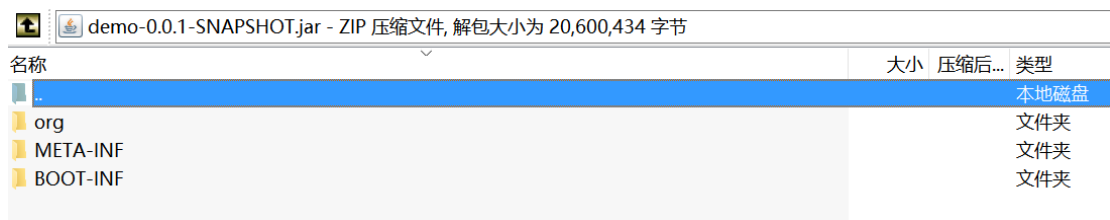
`-Dmaven.test.skip=true` 跳过测试。



最终打出来的包：

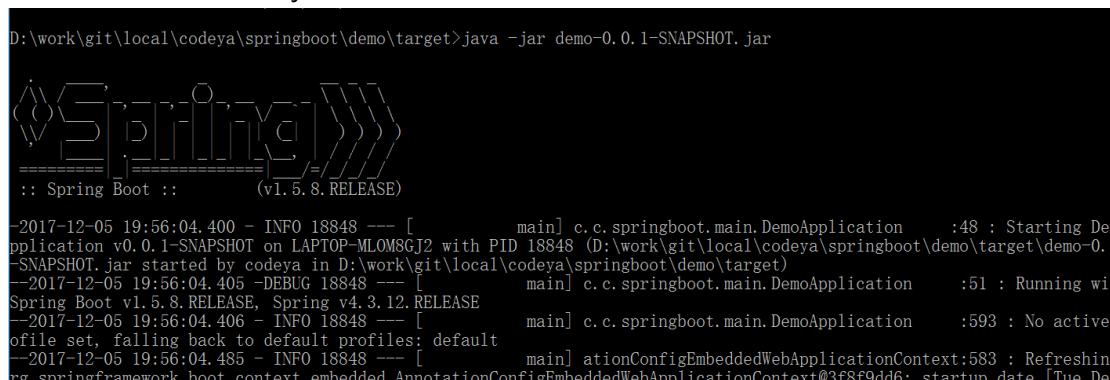


可以自己打开看看。。这个jar的目录和普通的jar不太一样。。里面包含了lib和内嵌的tomcat。。



8.2.2 使用 java -jar 运行此 jar

demo-0.0.1-SNAPSHOT.jar



```
命令提示符 - java -jar demo-0.0.1-SNAPSHOT.jar
[main] o.s.w.s.m.a.RequestMappingHandlerMapping:543 : Mapped "[[/error], produces=[text/html]]" onto public org.springframework.web.servlet.ModelAndView org.springframework.boot.autoconfigure.web.BasicErrorController.error(javax.servlet.http.HttpServletRequest)
[main] o.s.w.s.m.a.RequestMappingHandlerMapping:543 : Mapped "[[/error], produces=[text/html]]" onto public org.springframework.web.servlet.ModelAndView org.springframework.boot.autoconfigure.web.BasicErrorController.errorHtml(javax.servlet.http.HttpServletRequest, javax.servlet.http.HttpServletResponse)
[main] o.s.w.s.handler.SimpleUrlHandlerMapping:362 : Mapped URL path [/webjars/**] onto handler of type [class org.springframework.web.servlet.resource.ResourceHttpRequestHandler]
[main] o.s.w.s.handler.SimpleUrlHandlerMapping:362 : Mapped URL path [/**] onto handler of type [class org.springframework.web.servlet.resource.ResourceHttpRequestHandler]
[main] o.s.w.s.handler.SimpleUrlHandlerMapping:362 : Mapped URL path [/**/favicon.ico] onto handler of type [class org.springframework.web.servlet.resource.ResourceHttpRequestHandler]
[main] o.s.j.e.a.AnnotationMBeanExporter:431 : Registering beans for JMX exposure on startup
[main] s.b.c.e.t.TomcatEmbeddedServletContainer:201 : Tomcat started on port(s): 8080 (http)
[main] c.c.springboot.main.DemoApplication:57 : Started DemoApplication in 4.34 seconds (JVM running for 4.883)
```

8.3 使用 profile 区分配置

默认先加载 application.yml/.properties 文件，然后使用对应的 profile 文件来覆盖其配置。。

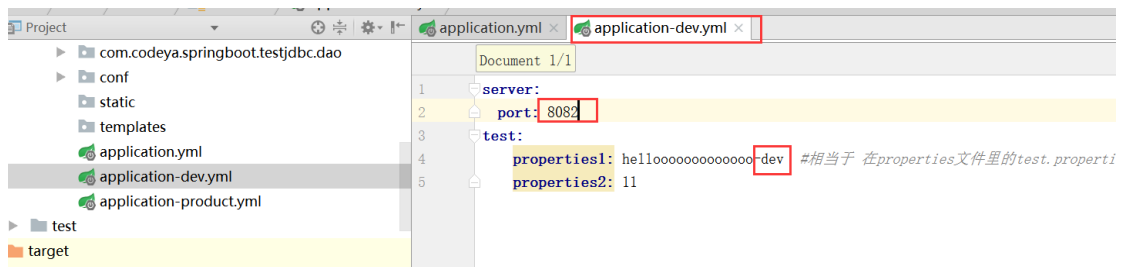
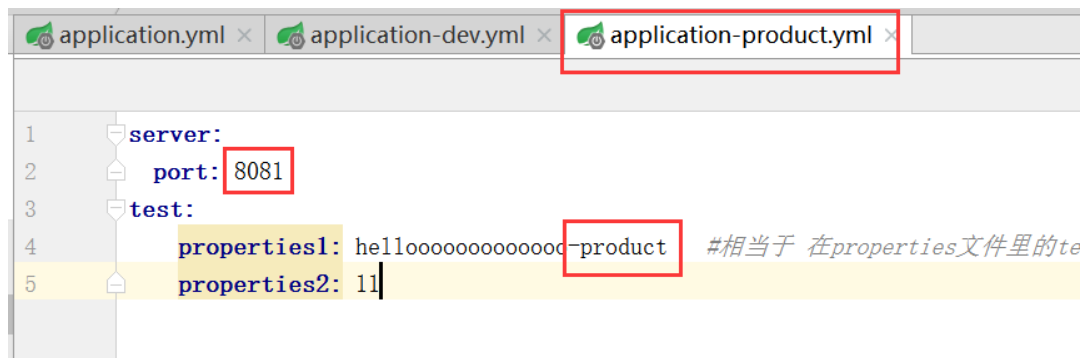
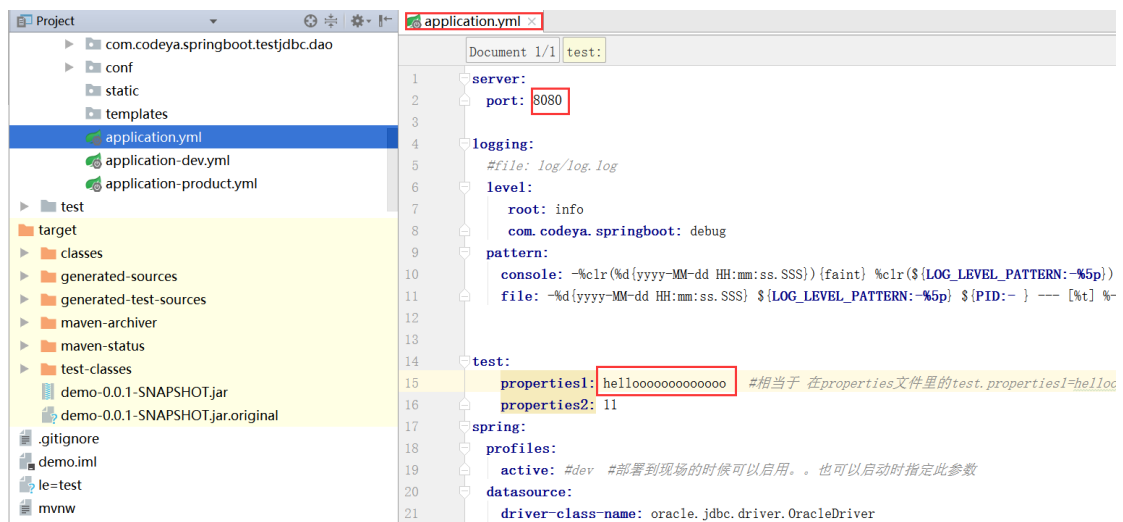
可以在启动的时候指定 profile。也可以在 application 文件里指定 spring.profiles.active(见下图)



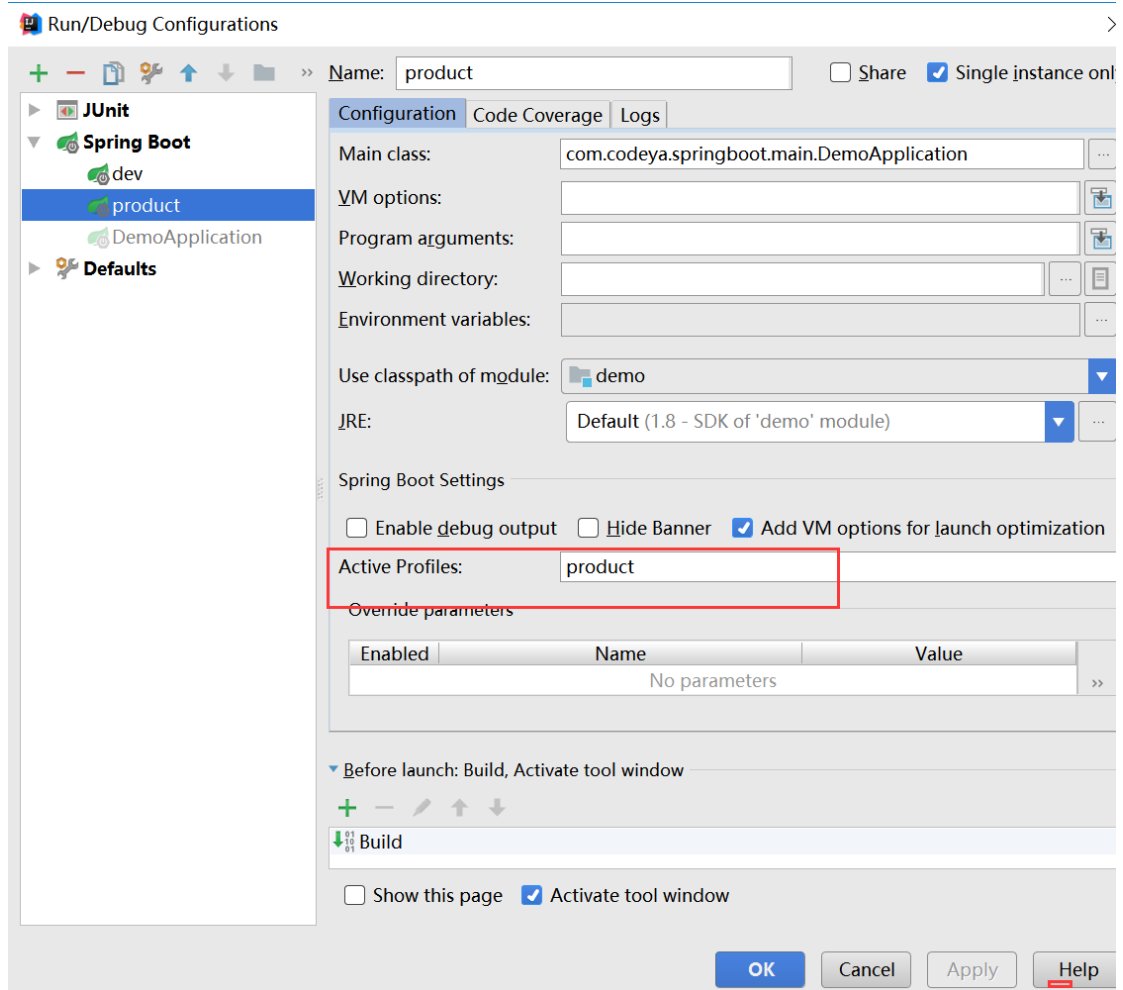
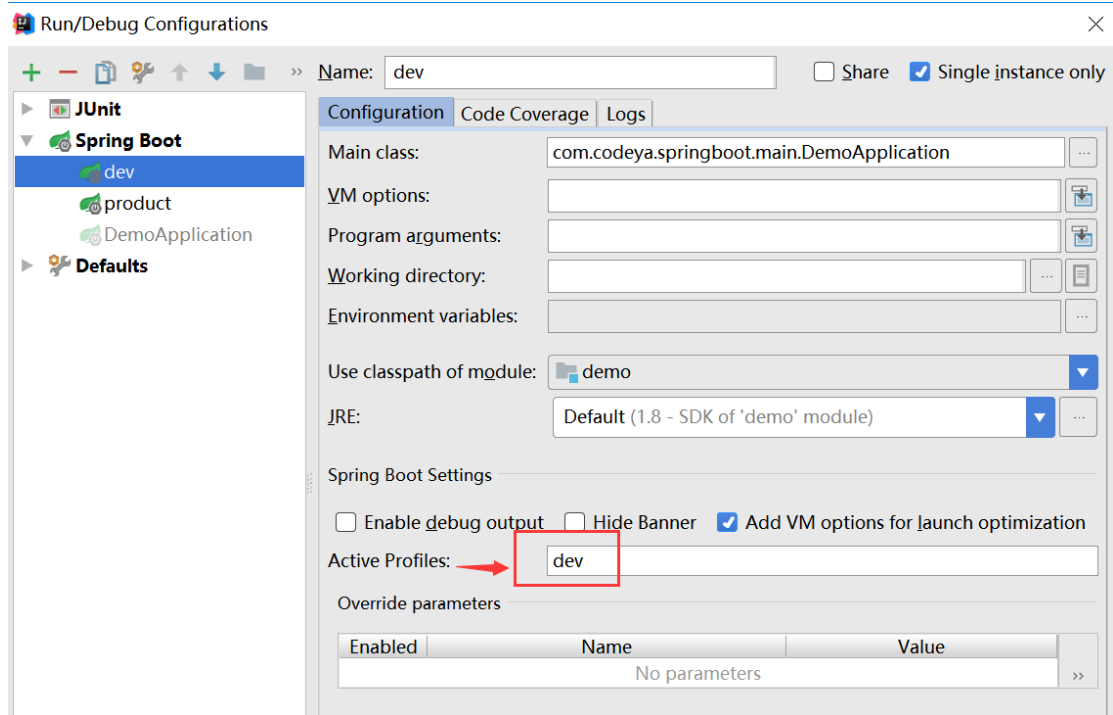
8.3.1 测试 profile

我们在 application.yml 里设置的端口为 8080，test.properties1= hellooooooooooooooooo

新增加两个配置文件 application-product.yml、application-dev.yml，端口分别设为 8081 和 8082，test.properties1 的值分别加上对应的后缀。。见下图



然后启动配置里指定 profile (可以看出是文件的后缀), 参考下图 :



启动测试

