通过IDEA新建一个普通微服务项目

1. 建Module
2. 改POM

```
<?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
https://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>
    <parent>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-parent</artifactId>
        <version>2.5.6</version>
        <relativePath/>
    </parent>

    <groupId>com.atguigu.docker</groupId>
    <artifactId>docker_boot</artifactId>
    <version>0.0.1-SNAPSHOT</version>

    <properties>
        <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
        <maven.compiler.source>1.8</maven.compiler.source>
        <maven.compiler.target>1.8</maven.compiler.target>
        <junit.version>4.12</junit.version>
        <log4j.version>1.2.17</log4j.version>
        <lombok.version>1.16.18</lombok.version>
        <mysql.version>5.1.47</mysql.version>
        <druid.version>1.1.16</druid.version>
        <mapper.version>4.1.5</mapper.version>
        <mybatis.spring.boot.version>1.3.0</mybatis.spring.boot.version>
    </properties>

    <dependencies>
        <!--SpringBoot通用依赖模块-->
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-web</artifactId>
        </dependency>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-actuator</artifactId>
        </dependency>
        <!--test-->
        <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>
            <plugin>
                <groupId>org.apache.maven.plugins</groupId>
                <artifactId>maven-resources-plugin</artifactId>
                <version>3.1.0</version>
            </plugin>
        </plugins>
    </build>

</project>
```

3. 写YML

```
server.port=8081
```

4. 主启动

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

}
```

5. 业务类

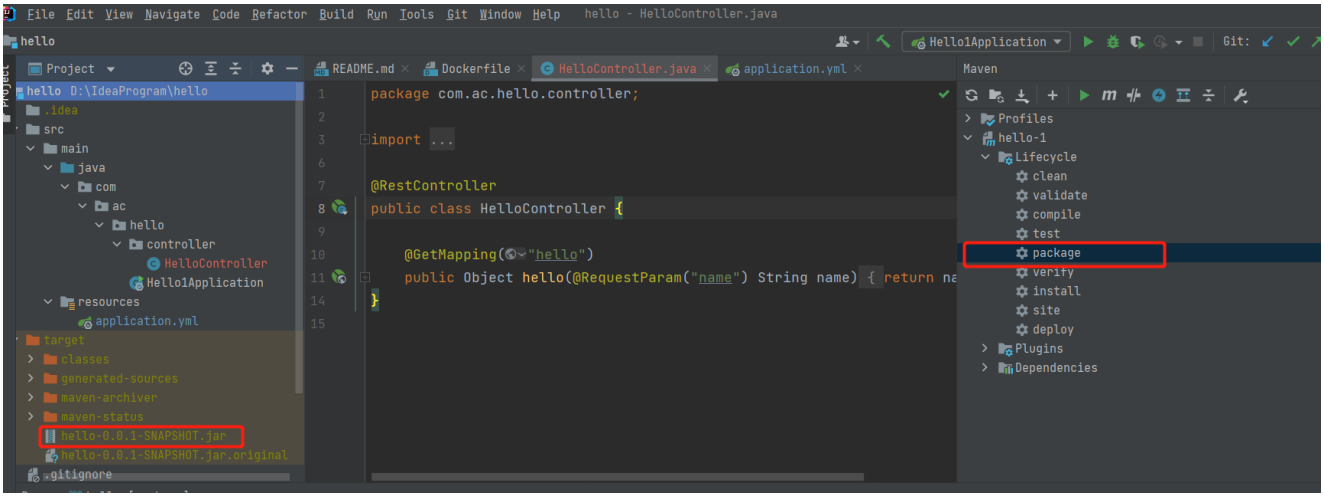
```
@RestController
public class OrderController
{
    @Value("${server.port}")
    private String port;

    @RequestMapping("/order/docker")
    public String helloDocker()
    {
        return "hello docker"+"\t"+port+"\t"+ UUID.randomUUID().toString();
    }

    @RequestMapping(value ="/order/index",method = RequestMethod.GET)
    public String index()
    {
```

```
        return "服务端口号: "+"\t"+port+"\t"+UUID.randomUUID().toString();
    }
}
```

通过dockerfile发布微服务部署到docker容器

1. IDEA里面搞懂微服务jar包

   这里使用一个简单项目



2. 编写Dockerfile

   Dockerfile内容

```
# 基础镜像使用java
FROM java:8
# 作者
MAINTAINER amazecode
# VOLUME 指定临时文件目录为/tmp，在主机/var/lib/docker目录下创建了一个临时文件并链接到容器的/tmp
VOLUME /tmp

# 将jar包添加到容器中并更名为app.jar
ADD hello-0.0.1-SNAPSHOT.jar app.jar

# 运行jar包
RUN bash -c 'touch /app.jar'
ENTRYPOINT ["java","-jar","/app.jar"]
#暴露8081端口作为微服务
EXPOSE 8081
```

3. 构建镜像

```
docker build -t ac_docker:1.0 .
```

```
[root@localhost myservice]# docker build -t ac_docker:1.0 .
[+] Building 142.8s (8/8) FINISHED
 => [internal] load build definition from Dockerfile                                                0.1s
 => => transferring dockerfile: 519B                                                                0.0s
 => [internal] load .dockerignore                                                                   0.1s
 => => transferring context: 2B                                                                     0.0s
 => [internal] load metadata for docker.io/library/java:8                                          16.8s
 => [internal] load build context                                                                   0.6s
 => => transferring context: 17.63MB                                                                0.5s
 => [1/3] FROM docker.io/library/java:8@sha256:c1ff613e8ba25833d2e1940da0940c3824f03f802c449f3d1815a66  122.6s
 => => resolve docker.io/library/java:8@sha256:c1ff613e8ba25833d2e1940da0940c3824f03f802c449f3d1815a66b7  0.0s
 => => sha256:d23bdf5b1b1b1afce5f1d0fd33e7ed8afbc084b594b9ccf742a5b27080d8a4a8 4.73kB / 4.73kB      0.0s
 => => sha256:c1ff613e8ba25833d2e1940da0940c3824f03f802c449f3d1815a66b7f8c0e9d 2.00kB / 2.00kB      0.0s
 => => sha256:5040bd2983909aa8896b9932438c3f1479d25ae837a5f6220242a264d0221f2d 51.36MB / 51.36MB   70.1s
 => => sha256:fce5728aad85a763fe3c419db16885eb6f7a670a42824ea618414b8fb309ccde 18.54MB / 18.54MB   20.8s
 => => sha256:76610ec20bf5892e24cebd4153c7668284aa1d1151b7c3b0c7d50c579aa5ce75 42.50MB / 42.50MB   51.3s
 => => sha256:60170fec2151d2108ed1420625c51138434ba4e0223d3023353d3f32ffe3cfc2 593.15kB / 593.15kB  21.9s
 => => sha256:e98f73de8f0d2ef292f58b004d67bc6e9ee779dcfaff7ebb3964649d4787b872 214B / 214B         22.2s
 => => sha256:11f7af24ed9cf47597dd6cf9963bb3e9109c963f0135e869a9e9b4999fdc12a3 242B / 242B         22.5s
 => => sha256:49e2d6393f32abb1de7c9395c04c822ceb2287383d5a90998f7bd8dbfd43d48c 130.10MB / 130.10MB  110.0s
 => => sha256:bb9cdec9c7f337940f7d872274353b66e118412cbfd433c711361bcf7922aea4 289.05kB / 289.05kB  52.1s
 => => extracting sha256:5040bd2983909aa8896b9932438c3f1479d25ae837a5f6220242a264d0221f2d           6.5s
 => => extracting sha256:fce5728aad85a763fe3c419db16885eb6f7a670a42824ea618414b8fb309ccde           1.8s
 => => extracting sha256:76610ec20bf5892e24cebd4153c7668284aa1d1151b7c3b0c7d50c579aa5ce75           7.1s
 => => extracting sha256:60170fec2151d2108ed1420625c51138434ba4e0223d3023353d3f32ffe3cfc2           0.4s
 => => extracting sha256:e98f73de8f0d2ef292f58b004d67bc6e9ee779dcfaff7ebb3964649d4787b872           0.0s
 => => extracting sha256:11f7af24ed9cf47597dd6cf9963bb3e9109c963f0135e869a9e9b4999fdc12a3           0.0s
 => => extracting sha256:49e2d6393f32abb1de7c9395c04c822ceb2287383d5a90998f7bd8dbfd43d48c          11.5s
 => => extracting sha256:bb9cdec9c7f337940f7d872274353b66e118412cbfd433c711361bcf7922aea4           0.1s
 => [2/3] ADD hello-0.0.1-SNAPSHOT.jar app.jar                                                      2.3s
 => [3/3] RUN bash -c 'touch /app.jar'                                                              0.5s
 => exporting to image                                                                              0.4s
 => => exporting layers                                                                             0.4s
 => => writing image sha256:a8ad0addbc7b43d480129844eb7b8c21ac5d63605c470b8920be0cd307a04c3b        0.0s
 => => naming to docker.io/library/ac_docker:1.0                                                    0.0s
[root@localhost myservice]# docker images
REPOSITORY                TAG       IMAGE ID       CREATED          SIZE
ac_docker                 1.0       a8ad0addbc7b   25 seconds ago   678MB
ubuntu-ifconfig           1.1       338596907cdc   4 days ago       117MB
```

4. 运行容器

```
docker run -d -p 8081:8081 --name my_hello ac_docker:1.0
```

```
[root@localhost myservice]# docker run -d -p 8081:8081 --name my_hello ac_docker:1.0
f7d04c0c4dfca411b47d456a577639a21b29e4d85f47184378469b49fb490cd2
[root@localhost myservice]# curl -XGET http://127.0.0.1:8081/hello?name=li
li[root@localhost myservice]#
```

5. 访问测试

```
# 虚拟机开始8081端口号以免被防火墙限制
firewall-cmd --zone=public --add-port=8081/tcp --permanent
# 重新加载
firewall-cmd --reload
# 查看开放端口
firewall-cmd --reload
```

xiaoli