1.开发的热部署

1.1 模板热部署

```
#Thymeleaf的配置
spring.thymeleaf.cache=false
#FreeMark的配置
spring.freemarker.cache=false
#Groovy的配置
spring.groovy.template.cache=false
#Velocity的配置
spring.velocity.cache=false
```

1.2 Spring Loaded

spring loaded可实现修改类文件的热部署

1.3 Jrebel

收费,不介绍

1.4 spring-boot-devtools

在Spring Boot项目中添加spring-boot-devtools依赖即可以实现页面和代码的热部署:

```
<dependency>
   <groupId>org.springframework.boot</groupId>
   <artifactId>spring-boot-devtools</artifactId>
   <version>2.1.6.RELEASE</version>
</dependency>
```

2.常规部署

2.1 jar形式

```
#打包
mvn package
#运行
java -jar xx.jar
```

注册为Linux服务,若想使用此用能,需要增加pom.xml文件的插件

```
<plugin>
   <groupId>org.springframework.boot</groupId>
   <artifactId>spring-boot-maven-plugin</artifactId>
   <version>2.1.6.RELEASE</version>
   <configuration>
      <executable>true</executable>
   </configuration>
</plugin>
mvn package
```

主流的Linux大多使用init.d或systemd的命令来注册服务

(1) 基于Linux的init.d部署

```
#注册服务 xx就是我们的服务名
sudo ln -s /var/apps/xx.jar /etc/init.d/xx
#启动服务
service xx start
#停止服务
service xx stop
#服务装填
service xx status
#开机启动
chkconfig xx on
```

(2) 基于Linux的Systemd部署

在/etc/systemd/system/目录下新建文件xx.service,写下如下内容:

```
[Unit]
Description=xx
After=syslog.target

[Service]
ExecStart= /usr/bin/java -jar /var/apps/xx.jar

[Install]
WantedBy=multi-user.target

#启动服务
systemctl start xx
或者:systemctl start xx.service
#停止服务
systemctl stop xx
或者:systemctl stop xx.service
#服务状态
systemctl status xx
或者:systemctl status xx.service
#开机启动
systemctl enable xx
或者:systemctl enable xx.service
#项目日志
journalctl -u xx
或者:journalctl -u xx.service
```

2.2 war形式

新建项目时选择打包方式 为 war

2.打包为jar时

新建Spring Boot项目时选择的打包方式时jar,部署时我们又想要用war包形式部署,

1.修改pom文件

```
<packaging>war</packaging>
```

增加下面依赖:

```
<dependency>
   <groupId>org.springframework.boot</groupId>
   <artifactId>spring-boot-starter-tomcat</artifactId>
   <scope>provided</scope>
</dependency>
```

增加ServletInitializer类:

```
public class ServletInitializer extends SpringBootServletInitializer{
   @Override
   protected SpringApplicationBuilder configure(SpringApplicationBuilder
application){
      return application.sources(ch8Application.class)
   }
}
```

3.云部署-基于docker的部署

3.1 Dockerfile

Dockerfile主要有如下的指令:

(1) FROM指令

FROM指令指明了当前镜像继承的基镜像.编译当前镜像时会自动下载基镜像

```
FROM java
```

(2) MAINTAINER指令

MAINTAINER指令指明了当前镜像的作者

```
MAINTAINER txx
```

(3) RUN指令

RUN指令可以在当前镜像上执行Linux命令形成一个新的层.RUN 是编译时的动作.

```
RUN /bin/bash -c "echo helloworld"
或RUN ["/bin/bash","-c","echo hello"]
CMD 和 ENTRYPOINT也有上面的两种形式
```

(4) CMD指令

CMD指令指明了启动镜像容器时的默认行为，一个Dockefile里只能有一个ＣＭＤ指令.

ＣＭＤ指令里设定的命令可以在运行镜像时使用参数覆盖. ＣＭＤ是运行时(run)的动作

```
CMD echo "this is a test"
```

(5) EXPOSE指令

EXPOST指明了镜像运行时的容器必须监听指定的端口

```
EXPOSE 8080
```

(6) ENV指令

ENV指令可用来设置环境变量

ENV myname=txx
或ENV myname txx

(7) ADD指令

ADD指令是从当前工作目录复制文件到镜像目录中去

ADD ch9_1.jar app.jar

(8) ENTRYPOINT指令

ENTRYPOINT指令可以让容器像一个可执行程序一样运行这样镜像运行时可以像软件一样接收参数执行.ENTRYPOINT是运行时(run)的动作

ENTRYPOINT ["java","-jar","/app.jar"]

3.2 项目目录及文件

Dockerfile示例:

```
FROM java:8
MAINTAINER txx
ADD ch9_1.jar app.jar
EXPOSE 8080
ENTRYPOINT ["java","-jar","/app.jar"]
```

3.3 编译镜像

docker build -t txx/ch9 .
其中txx/ch9为镜像名称,设置txx为前缀,这也是docker镜像的一种命名习惯.
注意最后的".",这是用来指明Dockerfile路径的,"."表示Dockerfile在当前路径下.

3.4 运行

docker run -d -p 8080:8080 txx/ch9
--name:指定运行的容器名称

4.Spring Boot的测试

测试用例:

```
@Runwith()
@SpringApplicationConfiguration(classes=ch9Application.class) //普通项目是
@ContextConfiguration
@WebAppConfiguration
@Transactional
public class ch9ApplicationTests{}}
```