目录

[一、准备工作 1](#_Toc518981665)

[（1）开启remote API 1](#_Toc518981666)

[（2）修改后重启服务 2](#_Toc518981667)

[2、使用前一定确保centos7防火墙关闭（每次开启centos7都得进行该操作） 2](#_Toc518981668)

[参考文档1： 2](#_Toc518981669)

[引用依赖 3](#_Toc518981670)

[二、典型实例： 3](#_Toc518981671)

[（1）简单的实例 4](#_Toc518981672)

[（2）java实现 dokcer 建立容器（如tomcat等）（不需指定环境变量） 7](#_Toc518981673)

[（3）java实现docker 创建容器（如msyql等）（需指定环境环境变量） 10](#_Toc518981674)

[（4）java实现docker 创建容器（如nginx等）（需配置容器） 11](#_Toc518981675)

[1、先用命令行测试 11](#_Toc518981676)

[2、java代码实现 12](#_Toc518981677)

[（5）java实现docker 创建容器（如nginx，mysql,tomcat）源码最终版！！！ 13](#_Toc518981678)

[三、docker实际应用，在容器中运行java Web项目 19](#_Toc518981679)

[1、在java container(单个容器)运行 简单的java web项目 19](#_Toc518981680)

[2、在java +mysql container(多个容器)运行 java web项目 25](#_Toc518981681)

[四、报错信息及参考文档 32](#_Toc518981682)

[五、源码：docker-demo 32](#_Toc518981683)

# 一、准备工作

1、

Docker远程访问是默认关闭的，要先打开

## （1）开启remote API

vim /lib/systemd/system/docker.service

ExecStart=/usr/bin/dockerd -H unix:///var/run/docker.sock -H tcp://0.0.0.0:2375

#ExecStart=/usr/bin/dockerd



## （2）修改后重启服务

systemctl daemon-reload

service Docker restart

参考：

Centos7中Docker开启远程访问（Centos7 Docker remote access configure） - CSDN博客

<https://blog.csdn.net/faryang/article/details/75949611>

## 2、使用前一定确保centos7防火墙关闭（每次开启centos7都得进行该操作）

service firewalld stop

后端：

本项目采用的是springboot 框架

与Docker通信，采用了现成的分装

## 参考文档1：

docker-client/user\_manual.md at master · spotify/Docker-client

<https://github.com/spotify/docker-client/blob/master/docs/user_manual.md#unix-socket-support>

spotify/Docker-client: A simple Docker client for the JVM

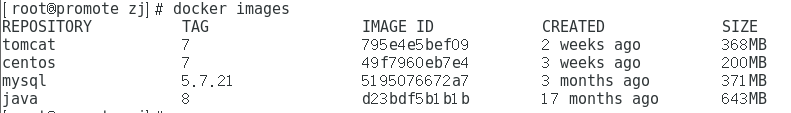
<https://github.com/spotify/docker-client>

官方Docker remote api文档：

[https://docs.docker.com/engine/api/v1.37/#](https://docs.docker.com/engine/api/v1.37/)

ps相关镜像的版本：

[1]



## 引用依赖

<dependency>  
 <groupId>com.spotify</groupId>  
 <artifactId>Docker-client</artifactId>  
 <version>8.11.7</version>  
</dependency>

<!—该依赖不能加，会冲突-->

<!--<dependency>-->  
 <!--<groupId>org.glassfish.hk2</groupId>-->  
 <!--<artifactId>hk2-api</artifactId>-->  
 <!--<version>2.2.0-b10</version>-->  
<!--</dependency>-->  
  
<dependency>  
 <groupId>org.glassfish.jersey.inject</groupId>  
 <artifactId>jersey-hk2</artifactId>  
 <version>2.26</version>  
</dependency>  
<dependency>  
 <groupId>com.spotify</groupId>  
 <artifactId>Docker-client</artifactId>  
 <classifier>shaded</classifier>  
 <version>8.11.7</version>  
</dependency>

# 二、典型实例：

## （1）简单的实例

|  |
| --- |
| */\*\*  \** ***@Auther:*** *zj  \** ***@Date:*** *2018/6/28 09:49  \** ***@Description:create*** *a Docker client  \** ***@return*** *\*/* @GetMapping("/createClient") public String createClient() throws DockerException, InterruptedException {   try {  // Create a client by using the builder 通过builder连接一个客户机  final DockerClient Docker = DefaultDockerClient.*builder*()  .uri( URI.*create*( "http://192.168.23.128:2375" ) ) //2375端口，是centos7打开的远程访问端口，自己可自行设计  // Set various options  .build();   // Pull an image 拉取镜像  //docker.pull("busybox"); 这里我Docker已经存在了，就不拉了   // Bind container ports to host ports 将容器端口绑定到主机端口  final String[] ports = {"81", "23"};  final Map<String, List<PortBinding>> portBindings = new HashMap<>();  for (String port : ports) {  List<PortBinding> hostPorts = new ArrayList<>();  hostPorts.add( PortBinding.*of*( "0.0.0.0", port ) );  portBindings.put( port, hostPorts );  }   // Bind container port 443 to an automatically allocated available host port. 将容器端口443绑定到自动分配的可用主机端口  List<PortBinding> randomPort = new ArrayList<>();  randomPort.add( PortBinding.*randomPort*( "0.0.0.0" ) );  portBindings.put( "443", randomPort );   final HostConfig = HostConfig.*builder*().portBindings( portBindings ).build();   System.*out*.println( hostConfig );   // Create container with exposed ports 创建带有暴露端口的容器  List<String> list = new ArrayList<>();  final ContainerConfig = ContainerConfig.*builder*()  .hostConfig( hostConfig )  .image( "centos:7" ).exposedPorts( ports )  .cmd( "sh", "-c", "while :; do sleep 1; done" )  .build();   final ContainerCreation creation = docker.createContainer( containerConfig );  final String id = creation.id();   System.*out*.println( creation );  System.*out*.println( id );   // Inspect container 检查容器  final ContainerInfo info = docker.inspectContainer( id );  System.*out*.println( info );   // Start container  docker.startContainer( id );   // Exec command inside running container with attached STDOUT and STDERR 执行命令在正运行容器带有附加的STDUT和STDER  final String[] command = {"ls"};  final ExecCreation = docker.execCreate(  id, command, DockerClient.ExecCreateParam.*attachStdout*() );  final LogStream output = docker.execStart( execCreation.id() );  final String execOutput = output.readFully();   System.*out*.println( execOutput );   // Kill container 杀死这个进程  docker.killContainer( id );   // Remove container 删除容器  docker.removeContainer( id );   // Close the Docker client 关闭这个容器客户机  docker.close();   } catch (Exception e) {  e.printStackTrace(); //printStackTrace()方法的意思是：在命令行打印异常信息在程序中出错的位置及原因  }    return null; } |

其中：

// Start container  
docker.startContainer( id );

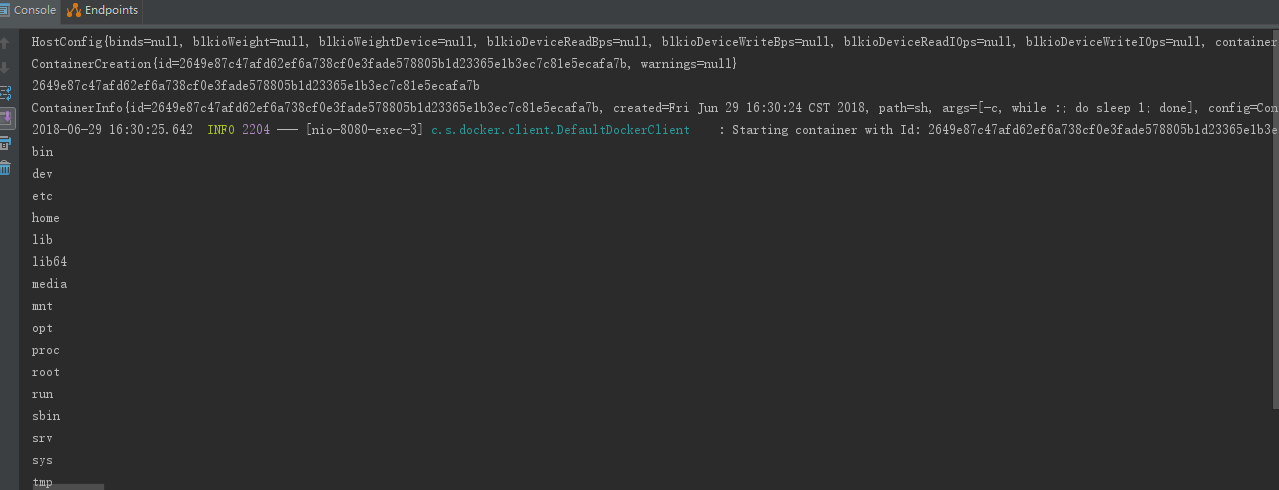
这边出现bug

解决：

Docker: Error response from daemon: driver failed programming external connectivity on endpoint - CSDN博客

<https://blog.csdn.net/shida_csdn/article/details/79376761>

成功



## （2）java实现 dokcer 建立容器（如tomcat等）（不需指定环境变量）

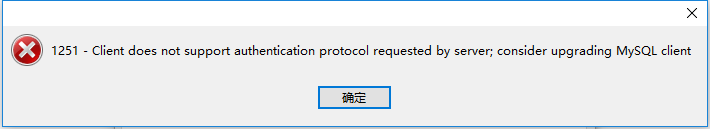
|  |
| --- |
| */\*\*  \** ***@Auther:*** *zj  \** ***@Date:*** *2018/6/28 22:49  \** ***@Description:create*** *a container,and start it(前提条件是镜像默认端口是一个，如果多个可能出错)  \** ***@return*** *\*/* @GetMapping("/createContainer")  public String createContainer(@RequestParam("imageName") String imageName,  @RequestParam("port") String modifyPort) {  try {  // Create a client by using the builder 通过builder连接一个客户机  final DockerClient docker = DefaultDockerClient.*builder*()  .uri( URI.*create*( "http://192.168.23.128:2375" ) ) //2375端口，是centos7打开的远程访问端口，自己可自行设计  // Set various options  .build();  final List<Image> quxImages = docker.listImages( DockerClient.ListImagesParam.*allImages*());  Integer status = 0;  //System.out.println(imageName);  for(Image quxImage:quxImages) {  if (quxImage.repoTags().toString().equals( String.*format*( "[" +imageName.toString()+"]") )){  //System.out.println( "已存在" );  status= status +1;  }  else {  status = status;  }  }  if(status !=0){  System.*out*.println( "该镜像已存在" );  }  else {  System.*out*.println( "该镜像不存在" );  docker.pull( imageName );//拉取镜像  }  //docker.pull( imageName );  System.*out*.println( "++++" );   // *TODO 宿主机端口 要随机分配* String exposedPort = "" + modifyPort;//宿主机端口   //获取镜像默认端口  final ImageInfo info = docker.inspectImage(imageName);  Set containerPortSet = info.containerConfig().exposedPorts();//镜像默认端口  String str = StringUtils.*join*(containerPortSet.toArray(), ";");//set to string  System.*out*.println( str );  // Iterator<Set> iterator = containerPortSet.iterator(); // while (iterator.hasNext()) { // System.out.println(iterator.next()); // String str2 = iterator.next(); // System.out.println( str2 ); // }  //String containerPort = "8080" + "/tcp";//镜像默认端口  String containerPort = str;//容器暴露端口  String containerPortName = str.split( "/" )[0];//容器的预暴露端口名  //final String[] ports = {"8080" +""};   //将宿主机的端口 与 容器暴露端口 绑定  final Map<String, List<PortBinding>> portBindings = new HashMap<String, List<PortBinding>>();  List<PortBinding> hostPorts = new ArrayList<PortBinding>();  hostPorts.add(PortBinding.*of*("0.0.0.0", exposedPort));  portBindings.put(containerPort + "", hostPorts);   final HostConfig = HostConfig.*builder*().portBindings( portBindings ).build();   System.*out*.println( hostConfig );     // Create container with exposed ports 创建带有暴露端口的容器  List<String> list = new ArrayList<>();  final ContainerConfig = ContainerConfig.*builder*()  .hostConfig( hostConfig )  .image( imageName )  //.exposedPorts( "8080" ) //------------  .exposedPorts( containerPortName )//暴露指定端口  //.cmd( "sh", "-c", "while :; do sleep 1; done" )//不加有默认cmd  .build();  final ContainerCreation creation = docker.createContainer( containerConfig );   final String id = creation.id();//获得容器的id   System.*out*.println( creation );  System.*out*.println( id );   // Inspect container 获得容器的详细信息  final ContainerInfo infoC = docker.inspectContainer( id );  System.*out*.println( ">>>>>>>>>>>>>>>>>>>" );  System.*out*.println( infoC );   // Start container  docker.startContainer( id );   // Exec command inside running container with attached STDOUT and STDERR 执行命令在正运行容器带有附加的STDUT和STDER  final String[] command = {"ls"};  final ExecCreation = docker.execCreate(  id, command, DockerClient.ExecCreateParam.*attachStdout*() );  final LogStream output = docker.execStart( execCreation.id() );  final String execOutput = output.readFully();   System.*out*.println( execOutput );   // Kill container 杀死这个进程  //docker.killContainer( id );   // Remove container 删除容器  //docker.removeContainer( id );   // Close the docker client 关闭这个容器客户机  docker.close();   return infoC.toString();   } catch (Exception e) {  e.printStackTrace(); //printStackTrace()方法的意思是：在命令行打印异常信息在程序中出错的位置及原因  }  return null;  } |

## （3）java实现docker 创建容器（如msyql等）（需指定环境环境变量）

将（2）相关部分，修改为下面的代码

|  |
| --- |
| // Create container with exposed ports 创建带有暴露端口的容器 List<String> list = new ArrayList<>(); final ContainerConfig = ContainerConfig.*builder*()  .hostConfig( hostConfig )  .image( imageName )  //.exposedPorts( "8080" ) //------------  .exposedPorts( containerPortName )//暴露指定端口  .env( "MYSQL\_ROOT\_PASSWORD=123456" ) //设置mysql初始密码  .build(); |

用navicat连接服务器，出现一下错误



此时我mysql版本是8

解决方案：

<https://zhuanlan.zhihu.com/p/36087723>

但我没有解决。。。。

但我mysql版本是5.7.1时，一切正常

mysql的latest版本bug比较多，建议用5.xxx版

参考：

MySQL 官方Docker镜像的使用 - IT笔录

<https://itbilu.com/linux/docker/EyP7QP86M.html>

Docker for Java Developers: Docker over HTTP/REST | Java Code Geeks - 2018

<https://www.javacodegeeks.com/2017/10/docker-java-developers-docker-http-rest.html>

docker常用命令详解 - CSDN博客

<https://blog.csdn.net/permike/article/details/51879578>

Docker 安装 MySQL | 菜鸟教程

<http://www.runoob.com/docker/docker-install-mysql.html>

## （4）java实现docker 创建容器（如nginx等）（需配置容器）

### 1、先用命令行测试

1、准备条件：先清理无用的数据卷

查看所有的 数据卷

$ docker volume ls

删除数据卷

$ docker volume rm Id/Name

2、在当前目录创建一个目录：

docker run -d -p 6800:80 -v /root/html:/usr/share/nginx/html nginx

出现bug:

docker 403 Forbidden nginx/1.15.0

解决方案：

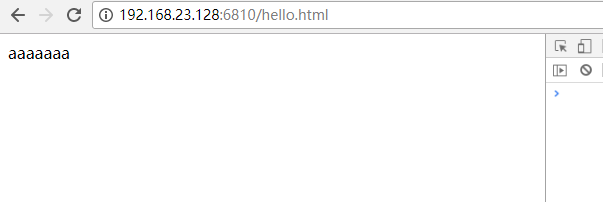
Docker 容器中配置nginx后报403 Forbidden 解决办法 - hct118 - 博客园

<https://www.cnblogs.com/HCT118/p/6801292.html>

3、

往/root/html中，添加一个文件hello.html

结果：



表明成功！！！

docker部署nginx - wzyy - 博客园

<https://www.cnblogs.com/wwzyy/p/8337965.html>

Docker 安装 Nginx | 菜鸟教程

http://www.runoob.com/docker/docker-install-nginx.html

<https://blog.docker.com/2015/04/tips-for-deploying-nginx-official-image-with-docker/>

数据卷 · Docker —— 从入门到实践

<https://yeasy.gitbooks.io/docker_practice/data_management/volume.html>

### 2、java代码实现

|  |
| --- |
| HostConfig.Bind bind = new HostConfig.Bind() {  @Override  public String to() {  return "/usr/share/nginx/html";  }   @Override  public String from() {  return "/home/zj/ppp";  }   @Override  public Boolean readOnly() {  return true;  }   @Override  public Boolean noCopy() {  return false;  }   @Override  public Boolean selinuxLabeling() {  return false;  } };  final HostConfig hostConfig = HostConfig.*builder*()  .binds(bind)  .portBindings( portBindings )  .build(); System.*out*.println( hostConfig ); |

## （5）java实现docker 创建容器（如nginx，mysql,tomcat）源码最终版！！！

|  |
| --- |
| */\*\*  \** ***@Auther:*** *zj  \** ***@Date:*** *2018/7/4 10:28  \** ***@Description:*** *create a container (Latest version)  \** ***@return*** *\* 注意env的格式，es:镜像是 mysql 时,env=MYSQL\_ROOT\_PASSWORD=123456  \* nginxUpDir的格式，es:/root/dockerDir/aaa ,/root/dockerDir/目录下一定要先建立aaa目录  \*/* @GetMapping("/createContainerLatestVersion")  public String createContainerLatestVersion(@RequestParam("imageName") String imageName,  @RequestParam("port") Integer modifyPort,  @RequestParam(value = "env",required = false,defaultValue = "no env") String env,  @RequestParam(value = "nginxUpDir",required = false,defaultValue = "no dir") String nginxupDir) throws DockerException, InterruptedException {   // Create a client by using the builder 通过builder连接一个客户机  final DockerClient docker = DefaultDockerClient.*builder*()  // *TODO ip地址自己修改* .uri( URI.*create*( "http://192.168.23.128:2375" ) ) //2375端口，是centos7打开的远程访问端口，自己可自行设计  // Set various options  .build();  final List<Image> quxImages = docker.listImages( DockerClient.ListImagesParam.*allImages*());  Integer status = 0;  //System.out.println(imageName);  for(Image quxImage:quxImages) {  if (quxImage.repoTags().toString().equals( String.*format*( "[" +imageName.toString()+"]") )){  //System.out.println( "已存在" );  status= status +1;  }  else {  status = status;  }  }  if(status !=0){  System.*out*.println( "该镜像已存在" );  }  else {  System.*out*.println( "该镜像不存在" );  docker.pull( imageName );//拉取镜像  }  //docker.pull( imageName );  System.*out*.println( "++++" );    // 宿主机端口与暴露端口绑定  Map<String, List<PortBinding>> portBindings = new HashMap<>(100);   final ImageInfo info = docker.inspectImage(imageName);  ImmutableSet<String> containerPortSet = info.containerConfig().exposedPorts();//获取镜像默认端口   // 去除/tcp后的端口集合  Set<String> realExportPorts = new HashSet<>();   UnmodifiableIterator<String> iterator = containerPortSet.iterator();  //镜像默认端口可能有多个，要循环绑定  while(iterator.hasNext()) {  // 取出暴露端口号,形如：80/tcp  String exportPort = iterator.next();//容器暴露预端口  // 取出去除/tcp后的端口号  String tmp = exportPort.trim().split("/")[0];  realExportPorts.add(tmp);   // 捆绑端口  List<PortBinding> hostPorts = new ArrayList<>();//  // *TODO 要随机分配 宿主机端口* Integer hostPort = modifyPort;  hostPorts.add(PortBinding.*of*("0.0.0.0", hostPort));  portBindings.put(exportPort, hostPorts);  }  // //String containerPort = "8080" + "/tcp";//镜像默认端口 // String containerPort = str;//容器暴露端口 // String containerPortName = str.split( "/" )[0];//容器的预暴露端口名 // //final String[] ports = {"8080" +""}; // // //将宿主机的端口 与 容器暴露端口 绑定 // final Map<String, List<PortBinding>> portBindings = new HashMap<String, List<PortBinding>>(); // List<PortBinding> hostPorts = new ArrayList<PortBinding>(); // hostPorts.add(PortBinding.of("0.0.0.0", exposedPort)); // portBindings.put(containerPort + "", hostPorts);   if (nginxupDir.equals( "no dir" )) { //没有指定nginx上传目录  final HostConfig hostConfig = HostConfig.*builder*()  .portBindings( portBindings )  .build();   System.*out*.println( hostConfig );   if (env.equals( "no env" )) {//不设置环境变量  // Create container with exposed ports 创建带有暴露端口的容器  ContainerConfig containerConfig = ContainerConfig.*builder*()  .hostConfig( hostConfig )  .image( imageName )  .exposedPorts( realExportPorts )//暴露指定端口  .build();  final ContainerCreation creation = docker.createContainer( containerConfig );   final String id = creation.id();//获得容器的id   // Inspect container 获得容器的详细信息  final ContainerInfo infoC = docker.inspectContainer( id );  System.*out*.println( ">>>>>>>>>>>>>>>>>>>" );  System.*out*.println( infoC );   // Start container  docker.startContainer( id );   // Close the docker client 关闭这个容器客户机  docker.close();   return infoC.toString();  } else { //设置环境变量  // Create container with exposed ports 创建带有暴露端口的容器  ContainerConfig containerConfig = ContainerConfig.*builder*()  .hostConfig( hostConfig )  .image( imageName )  .exposedPorts( realExportPorts )//暴露指定端口  .env(env) //设置环境变量  .build();  final ContainerCreation creation = docker.createContainer( containerConfig );   final String id = creation.id();//获得容器的id   // Inspect container 获得容器的详细信息  final ContainerInfo infoC = docker.inspectContainer( id );  System.*out*.println( ">>>>>>>>>>>>>>>>>>>" );  System.*out*.println( infoC );   // Start container  docker.startContainer( id );   // Close the docker client 关闭这个容器客户机  docker.close();   return infoC.toString();  }   } else { //指定nginx上传目录  HostConfig.Bind bind = new HostConfig.Bind() {  @Override  public String to() {  return "/usr/share/nginx/html";  }   @Override  public String from() {  //*TODO nginxupDir 应该事先根据用户id,创建用户自己的上传目录* return nginxupDir;  }   @Override  public Boolean readOnly() {  return true;  }   @Override  public Boolean noCopy() {  return false;  }   @Override  public Boolean selinuxLabeling() {  return false;  }  };   final HostConfig hostConfig = HostConfig.*builder*()  .binds( bind )  .portBindings( portBindings )  .build();   System.*out*.println( hostConfig );   if (env.equals( "no env" )) {//不设置环境变量  // Create container with exposed ports 创建带有暴露端口的容器  ContainerConfig containerConfig = ContainerConfig.*builder*()  .hostConfig( hostConfig )  .image( imageName )  .exposedPorts( realExportPorts )//暴露指定端口  .build();  final ContainerCreation creation = docker.createContainer( containerConfig );   final String id = creation.id();//获得容器的id   // Inspect container 获得容器的详细信息  final ContainerInfo infoC = docker.inspectContainer( id );  System.*out*.println( ">>>>>>>>>>>>>>>>>>《" );  System.*out*.println( infoC );   // Start container  docker.startContainer( id );   // Close the docker client 关闭这个容器客户机  docker.close();   return infoC.toString();  } else { //设置环境变量  // Create container with exposed ports 创建带有暴露端口的容器  ContainerConfig containerConfig = ContainerConfig.*builder*()  .hostConfig( hostConfig )  .image( imageName )  .exposedPorts( realExportPorts )//暴露指定端口  .env(env) //设置环境变量  .build();  final ContainerCreation creation = docker.createContainer( containerConfig );   final String id = creation.id();//获得容器的id   // Inspect container 获得容器的详细信息  final ContainerInfo infoC = docker.inspectContainer( id );  System.*out*.println( "《《《《《《《《《《" );  System.*out*.println( infoC );   // Start container  docker.startContainer( id );   // Close the docker client 关闭这个容器客户机  docker.close();    return infoC.toString();  }   } // // Exec command inside running container with attached STDOUT and STDERR 执行命令在正运行容器带有附加的STDUT和STDER // final String[] command = {"ls"}; // final ExecCreation execCreation = docker.execCreate( // id, command, DockerClient.ExecCreateParam.attachStdout() ); // final LogStream output = docker.execStart( execCreation.id() ); // final String execOutput = output.readFully(); // System.out.println( execOutput  } |

# 三、docker实际应用，在容器中运行java Web项目

## 1、在java container(单个容器)运行 简单的java web项目

（1）先准备一个简单的java web项目（spring boot）

下面是该项目的核心：

|  |
| --- |
| C:\Users\zj\Desktop\2345截图20180705102732.png |

把它打成jar包

并将整个项目上传到宿主机的一个目录下（该目录自己创），我这里是/root/java-test

（2）创建centos7的容器

源码：

|  |
| --- |
| */\*\*  \** ***@Auther:*** *zj  \** ***@Date:*** *2018/7/5 09:49  \** ***@Description:*** *create a docker (适用于java容器)  \** ***@return*** *\* 注意：java镜像是没有默认端口的  \*/* @GetMapping("/createContainerJava")  public String createContainerJava(@RequestParam("imageName") String imageName,  @RequestParam("port") String modifyPort) {   try {  // Create a client by using the builder 通过builder连接一个客户机  final DockerClient docker = DefaultDockerClient.*builder*()  .uri( URI.*create*( "http://192.168.23.128:2375" ) ) //2375端口，是centos7打开的远程访问端口，自己可自行设计  // Set various options  .build();  final List<Image> quxImages = docker.listImages( DockerClient.ListImagesParam.*allImages*() );  Integer status = 0;  System.*out*.println(imageName);  //*TODO 可能id一样，tag不一样。es:[java:8,java:8u111].这边要修改* for (Image quxImage : quxImages) {  if (quxImage.repoTags().toString().equals( String.*format*( "[" + imageName.toString() + "]" ) )) {  System.*out*.println( quxImage.repoTags().toString() );  status = status + 1;  } else {  status = status;  }  }  if (status != 0) {  System.*out*.println( "该镜像已存在" );  } else {  System.*out*.println( "该镜像不存在" );  docker.pull( imageName );//拉取镜像  }  //docker.pull( imageName );  System.*out*.println( "+++++" );   // *TODO 宿主机端口 要随机分配* String exposedPort = "" + modifyPort;//宿主机端口  // //获取镜像默认端口 // final ImageInfo info = docker.inspectImage(imageName); // Set containerPortSet = info.containerConfig().exposedPorts();//镜像默认端口 // String str = StringUtils.join(containerPortSet.toArray(), ";");//set to string // System.out.println( str ); // Iterator<Set> iterator = containerPortSet.iterator(); // while (iterator.hasNext()) { // System.out.println(iterator.next()); // String str2 = iterator.next(); // System.out.println( str2 ); // }  //String containerPort = "8080" + "/tcp";//镜像默认端口  //*TODO 该端口是用户上传项目的端口（server.port）* String containerPort = "8080";//容器暴露端口   //将宿主机的端口 与 容器暴露端口 绑定  final Map<String, List<PortBinding>> portBindings = new HashMap<String, List<PortBinding>>();  List<PortBinding> hostPorts = new ArrayList<PortBinding>();  hostPorts.add(PortBinding.*of*("0.0.0.0", exposedPort));  // *todo 很重要这里"8080/tcp"表示8080必须是容器的默认端口，才能绑定。"8080"表明指定容器的暴露端口，即可绑定* portBindings.put(containerPort,hostPorts);   //实现文件挂载（挂载成功目录下的文件实现共享）  HostConfig.Bind bind = new HostConfig.Bind() {  @Override  public String to() {  return "/www";  }//容器内的目录。如果不存在，会自动新建   @Override  public String from() {  return "/root/java-test";  }//宿主机的目录（即存放项目的目录），一定要事先建立好，不然会报错   @Override  public Boolean readOnly() {  return true;  }   @Override  public Boolean noCopy() {  return false;  }   @Override  public Boolean selinuxLabeling() {  return false;  }//  };   final HostConfig hostConfig = HostConfig.*builder*()  .binds(bind)//实现文件挂载  .portBindings( portBindings )//实现端口绑定  .build();  System.*out*.println( hostConfig );   // Create container with exposed ports 创建带有暴露端口的容器  final ContainerConfig containerConfig = ContainerConfig.*builder*()  .hostConfig( hostConfig )  .image( imageName )  .exposedPorts( containerPort )//暴露指定端口  .cmd( "sh", "-c", "while :; do sleep 1; done" )//让容器创建后不默认关闭  .build();  final ContainerCreation creation = docker.createContainer( containerConfig );   final String id = creation.id();//获得容器的id   System.*out*.println( creation );  System.*out*.println( id );   // Inspect container 获得容器的详细信息  final ContainerInfo infoC = docker.inspectContainer( id );  System.*out*.println( ">>>>>>>>>>>>>>>>>>>" );  System.*out*.println( infoC );   // Start container  docker.startContainer( id );   // Exec command inside running container with attached STDOUT and STDERR 执行命令在正运行容器带有附加的STDUT和STDER  final String[] command = {"ls"};  final ExecCreation execCreation = docker.execCreate(  id, command, DockerClient.ExecCreateParam.*attachStdout*() );  final LogStream output = docker.execStart( execCreation.id() );  final String execOutput = output.readFully();   System.*out*.println( execOutput );   // Close the docker client 关闭这个容器客户机  docker.close();   return infoC.toString();   }catch (Exception e) {  e.printStackTrace(); //printStackTrace()方法的意思是：在命令行打印异常信息在程序中出错的位置及原因  }   return null;  } |

注意：要保证java容器不关闭，cmd命令："sh", "-c", "while :; do sleep 1; done"

或者"/bin/bash"

docker ps –a

C:\Users\zj\Desktop\2345截图20180706100925.png

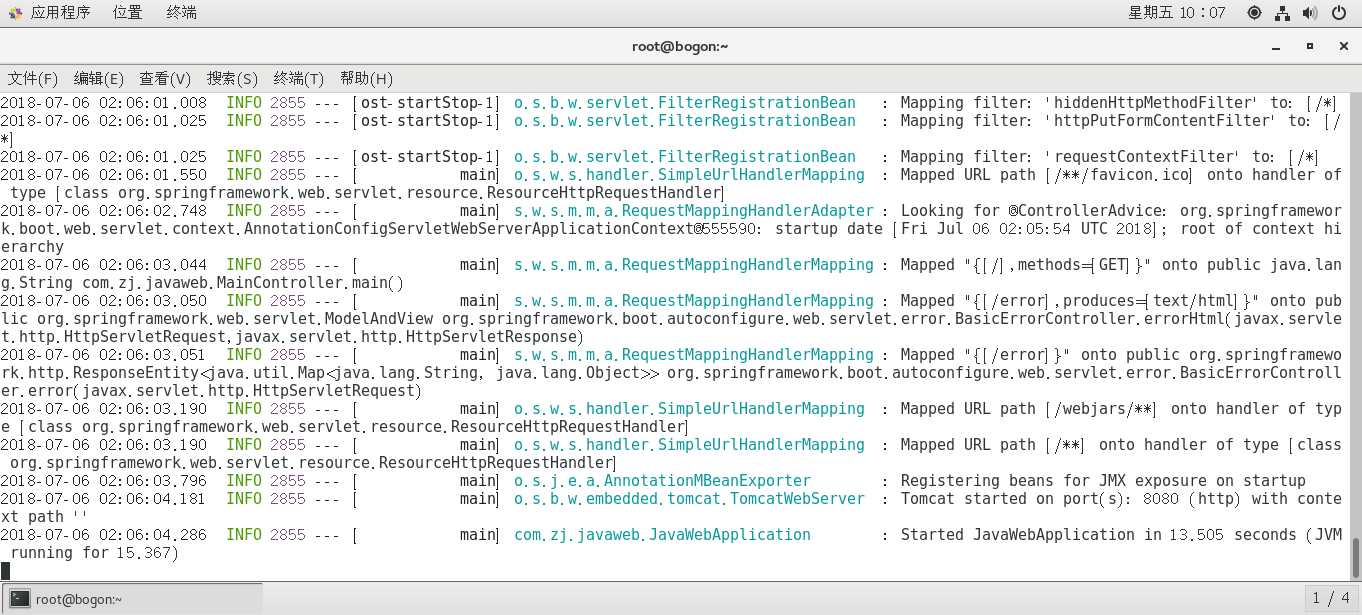
3、

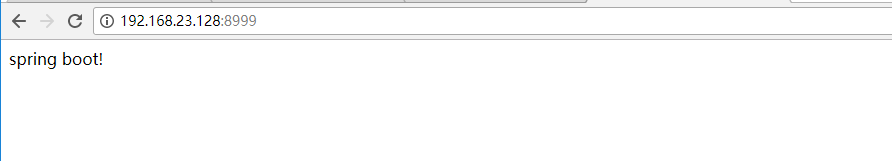
进入容器中

docker exec –it 容器的id bash

然后

java –jar /www/java-web/target/ java-web-0.0.1-SNAPSHOT.jar





表明运行成功

参考资料：

docker运行jar文件 - 幽冥领域的个人空间 - 开源中国

<https://my.oschina.net/aidelingyu/blog/877970>

java - 如何在docker容器中运行一个jar而不会被退出？ - SegmentFault 思否

<https://segmentfault.com/q/1010000006964527>

docker运行jar文件\_Java\_第七城市

<http://www.th7.cn/Program/java/201704/1150054.shtml>

Using Docker containers for your Spring boot applications - g00glen00b

<https://g00glen00b.be/docker-spring-boot/>

使用 Docker 搭建 Java Web 运行环境 - 黄勇 - 开源中国

<https://my.oschina.net/huangyong/blog/372491>

Spring Boot 2.0(四)：使用 Docker 部署 Spring Boot - 纯洁的微笑博客

<http://www.ityouknow.com/springboot/2018/03/19/spring-boot-docker.html>

Running Spring Boot in a Docker container #16 - 后端 - 掘金

<https://juejin.im/entry/59ee11836fb9a044fd10ef0f>

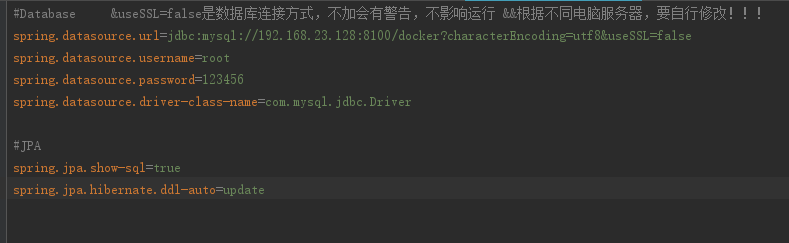
## 2、在java +mysql container(多个容器)运行 java web项目

(1)本次我使用的是 –link 来连接多个容器

首先准备一个简单的spring boot 项目，带有mysql

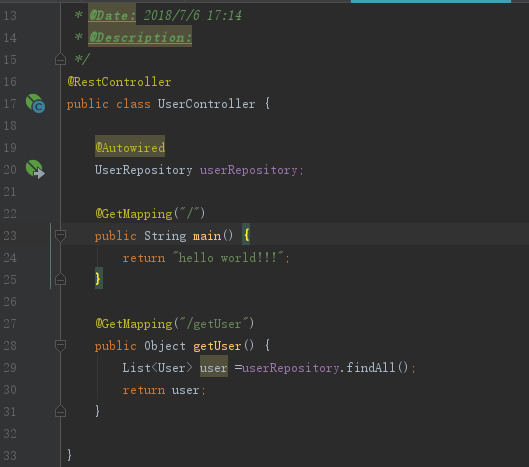
依赖就不放了。

配置：



192.168.23.128：8100是宿主机与mysql容器绑定的ip+端口

controller:



domain,repository层就省略了

把项目打成jar包

把整个项目放到宿主机的java-web目录下（接下来会与容器绑定）

(2)

接下来介绍docker相关接口

|  |
| --- |
| */\*\*  \** ***@Auther:*** *zj  \** ***@Date:*** *2018/7/5 09:49  \** ***@Description:*** *create a docker (比如创建java容器，同时连接mysql容器)  \** ***@return*** *\* 注意：java镜像是没有默认端口的  \*/* @GetMapping("/createContainerJavaWithMysql")  public String createContainerJavaWithMysql(@RequestParam("imageName") String imageName,  @RequestParam("port") String modifyPort) {   try {  // Create a client by using the builder 通过builder连接一个客户机  final DockerClient docker = DefaultDockerClient.*builder*()  .uri( URI.*create*( "http://192.168.23.128:2375" ) ) //2375端口，是centos7打开的远程访问端口，自己可自行设计  // Set various options  .build();  final List<Image> quxImages = docker.listImages( DockerClient.ListImagesParam.*allImages*() );  Integer status = 0;  System.*out*.println(imageName);  //*TODO 可能id一样，tag不一样。es:[java:8,java:8u111].这边要修改* for (Image quxImage : quxImages) {  if (quxImage.repoTags().toString().equals( String.*format*( "[" + imageName.toString() + "]" ) )) {  System.*out*.println( quxImage.repoTags().toString() );  status = status + 1;  } else {  status = status;  }  }  if (status != 0) {  System.*out*.println( "该镜像已存在" );  } else {  System.*out*.println( "该镜像不存在" );  docker.pull( imageName );//拉取镜像  }  //docker.pull( imageName );  System.*out*.println( "+++++" );   // *TODO 宿主机端口 要随机分配* String exposedPort = "" + modifyPort;//宿主机端口   //String containerPort = "8080" + "/tcp";//镜像默认端口  //*TODO 该端口是用户上传项目的端口（server.port）* String containerPort = "8080";//容器暴露端口   //将宿主机的端口 与 容器暴露端口 绑定  final Map<String, List<PortBinding>> portBindings = new HashMap<String, List<PortBinding>>();  List<PortBinding> hostPorts = new ArrayList<PortBinding>();  hostPorts.add(PortBinding.*of*("0.0.0.0", exposedPort));  // *todo 很重要这里"8080/tcp"表示8080必须是容器的默认端口，才能绑定。"8080"表明指定容器的暴露端口，即可绑定* portBindings.put(containerPort,hostPorts);   //实现文件挂载（挂载成功目录下的文件实现共享）  HostConfig.Bind bind = new HostConfig.Bind() {  @Override  public String to() {  return "/qqq";  }//容器内的目录。如果不存在，会自动新。   @Override  public String from() {  return "/root/java-test";  }//宿主机的目录（即存放项目的目录），一定要事先建立好，不然会报错   @Override  public Boolean readOnly() {  return true;  }   @Override  public Boolean noCopy() {  return false;  }   @Override  public Boolean selinuxLabeling() {  return false;  }//  };   String link\_mysql = String.*format*( "08449aa44992097676f3e8dc9190c13ceee59b56b789863188882b968709f5f6" //连接容器的id或名称// *todo*该容器id应该是前端传入  +":"  +"zjMysql" //别称  );   final HostConfig hostConfig = HostConfig.*builder*()  .links( link\_mysql )//连接容器  .binds(bind)//实现文件挂载  .portBindings( portBindings )//实现端口绑定  .build();  System.*out*.println( hostConfig );   // Create container with exposed ports 创建带有暴露端口的容器  final ContainerConfig containerConfig = ContainerConfig.*builder*()  .hostConfig( hostConfig )  .image( imageName )  .exposedPorts( containerPort )//暴露指定端口  .cmd( "sh", "-c", "while :; do sleep 1; done" )//让容器创建后不默认关闭  .build();  final ContainerCreation creation = docker.createContainer( containerConfig );   final String id = creation.id();//获得容器的id   System.*out*.println( creation );  System.*out*.println( id );   // Inspect container 获得容器的详细信息  final ContainerInfo infoC = docker.inspectContainer( id );  System.*out*.println( ">>>>>>>>>>>>>>>>>>>" );  System.*out*.println( infoC );   // Start container  docker.startContainer( id );   // Exec command inside running container with attached STDOUT and STDERR 执行命令在正运行容器带有附加的STDUT和STDER  final String[] command = {"ls"};  final ExecCreation execCreation = docker.execCreate(  id, command, DockerClient.ExecCreateParam.*attachStdout*() );  final LogStream output = docker.execStart( execCreation.id() );  final String execOutput = output.readFully();   System.*out*.println( execOutput );   // Close the docker client 关闭这个容器客户机  docker.close();   return infoC.toString();   }catch (Exception e) {  e.printStackTrace(); //printStackTrace()方法的意思是：在命令行打印异常信息在程序中出错的位置及原因  }   return null;  } |

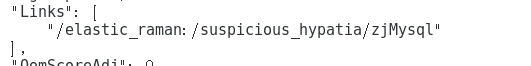
调用此接口

http://127.0.0.1:8080/createContainerJavaWithMysql?imageName=java:8&&port=8888

（3）docker ps -a

C:\Users\zj\Desktop\2345截图20180706210700.png

docker inspect 3138a0937c15

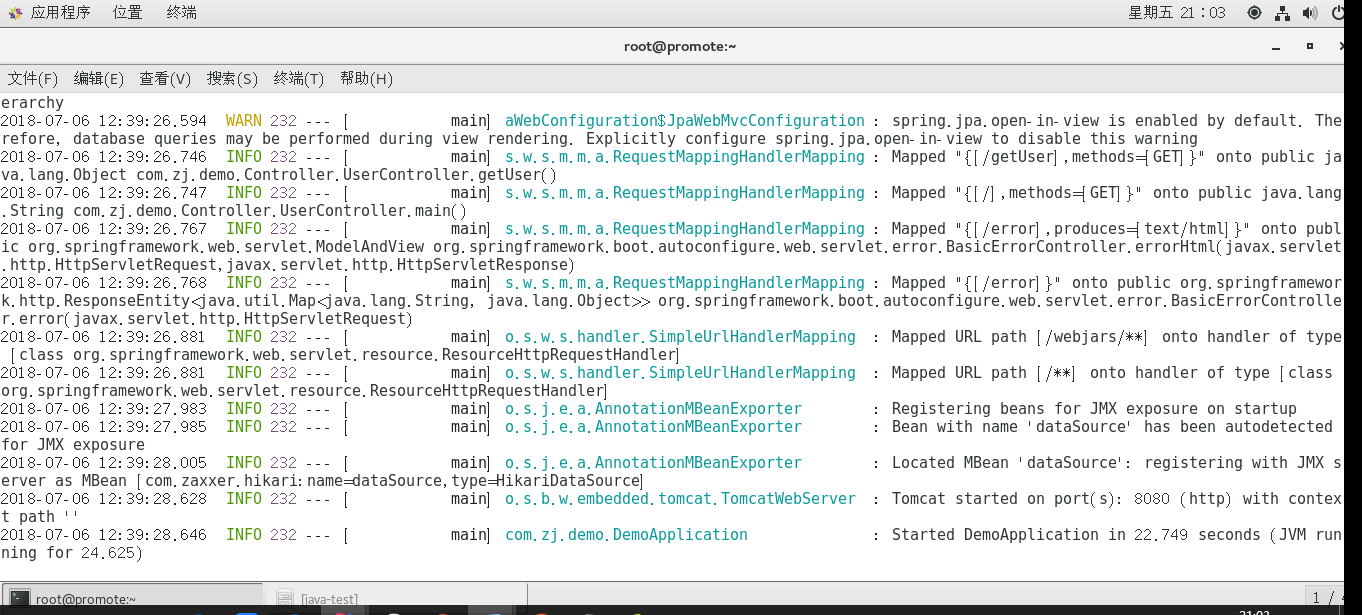


我们可以看到已经连接了

docker exec –it 3138a0937c15 bash

进入该容器

java –jar xx.jar



通过navicat查看数据库



表明本次接口测试成功！

参考资料;

Docker Container Operations with Docker-Client API for Java

<https://qiita.com/bmj0114/items/c4f3c4f4cb8912b0d4a9>

Legacy container links | Docker Documentation

<https://docs.docker.com/network/links/>

DefaultDockerClient (docker-client 8.9.1 API)

<https://static.javadoc.io/com.spotify/docker-client/8.9.1/com/spotify/docker/client/DefaultDockerClient.html#createNetwork-com.spotify.docker.client.messages.NetworkConfig->

Legacy container links | Docker Documentation

<https://docs.docker.com/network/links/#connect-using-network-port-mapping>

# 四、报错信息及参考文档

如果报错：WARNING: IPv4 forwarding is disabled. Networking will not work.

解决方案1：重启centos7

解决方案2：<https://www.cnblogs.com/daxiansmd/p/6413063.html>

参考：

Docker Container Operations with Docker-Client API for Java

<https://qiita.com/bmj0114/items/c4f3c4f4cb8912b0d4a9>

Java Examples for com.spotify.docker.client.messages.ContainerConfig（宿主机端口与指定的容器端口绑定等一系列例子）

<https://www.javatips.net/api/com.spotify.docker.client.messages.containerconfig>

Docker Container Operations with Docker-Client API for Java

<https://qiita.com/bmj0114/items/c4f3c4f4cb8912b0d4a9>

Java 集合转换(数组、List、Set、Map相互转换) - CSDN博客

<https://blog.csdn.net/top_code/article/details/10552827>

mysql | Docker Documentation

<https://docs.docker.com/samples/library/mysql/>

Docker for Java Developers: Docker over HTTP/REST | Java Code Geeks - 2018

<https://www.javacodegeeks.com/2017/10/docker-java-developers-docker-http-rest.html>

# 五、源码：docker-demo