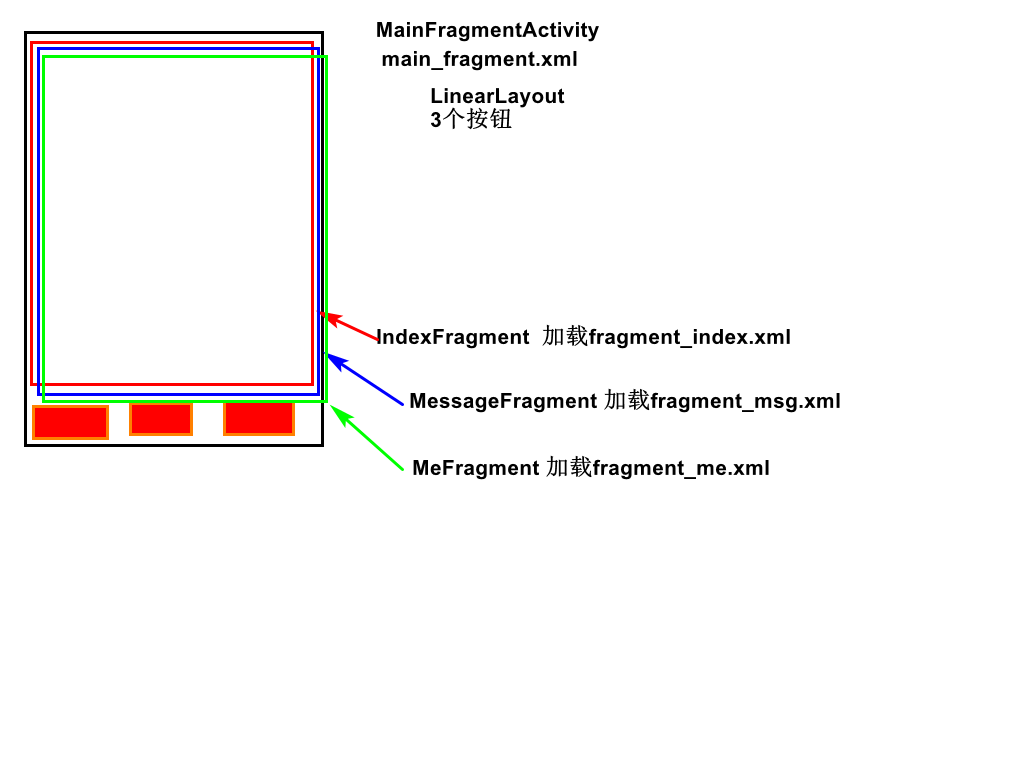
# 复习



MessageFragment

MeFragment

MainFragmentActivity

Fragments[]=indexFragment,MessgeFrgment,MeFragment

Buttons[]=indexBtn,messageBtn,MeBtn

onCreate()

{

创建msgFragment,MeFragmnet

找到三个按钮

Btn.setOnClickListener(BtnListener)

}

Class BtnLisener implements OnClickListener

{

onClick{

int selectBtnIndex=1

fragments[1]=messageFragment

fragmentManager

fragmentTransaction

trans.add()

trans.show()

trans.hide()

}

}

# 拼多商城

## 显示 消息，我

商城首页，消息，我之间的关系，如图-99所示。

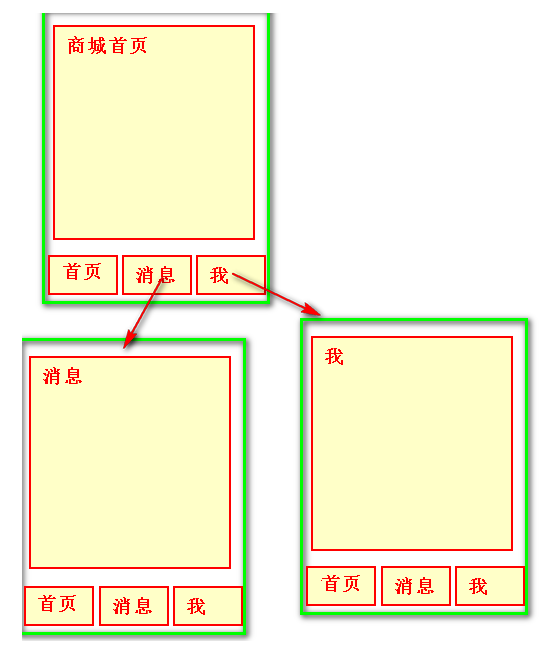


图- 99

创建messageFragment

**public class** MessageFragment **extends** Fragment {  
 @Nullable  
 @Override  
 **public** View onCreateView(LayoutInflater inflater, @Nullable ViewGroup container, @Nullable Bundle savedInstanceState) {  
 View view=View.*inflate* (getActivity(),  
 R.layout.***fragment\_message***,  
 **null**);  
 **return** view;  
 }  
}

创建meFragment

**public class** MeFragment **extends** Fragment {  
 @Nullable  
 @Override  
 **public** View onCreateView(LayoutInflater inflater, @Nullable ViewGroup container, @Nullable Bundle savedInstanceState) {  
 View view=View.*inflate* (getActivity(),  
 R.layout.***fragment\_me***,  
 **null**);  
 **return** view;  
 }  
}

修改MainFragmentActivity

**public class** MainFragmentActivity **extends** FragmentActivity {  
 IndexFragment **indexFragment**;  
 *//indexfragment,messgeFragment,mefragment* Fragment[] **fragments**=**new** Fragment[3];  
  
 *//indexBtn,messageBtn,MeBtn* Button[] **buttons**=**new** Button[3];  
  
 *//现在显示的是第几个fragment* **int currentFragmentIndex**=0;  
  
 *//单击的按钮是第几个* **int clickBtnIndex**=0;  
 *//参数只有一个* @Override  
 **protected void** onCreate  
 (@Nullable Bundle savedInstanceState) {  
 **super**.onCreate(savedInstanceState);  
 setContentView(R.layout.***main\_fragment***);  
 *//在fragment\_container中显示indexFragment  
  
 //开始事务* FragmentManager manager=getSupportFragmentManager();  
 FragmentTransaction transaction=manager.beginTransaction();  
 *//动作1：加载indexFragment到fragment\_container中* **indexFragment**=**new** IndexFragment();  
 transaction.add(R.id.***fragment\_container***,**indexFragment**);  
 *//动作2：显示indexfragment* transaction.show(**indexFragment**);  
 *//add()和show()有一个失败了，出异常，不执行commit()  
 //提交事务* transaction.commit();  
  
 *//创建两个fragment* MessageFragment messageFragment=**new** MessageFragment();  
MeFragment meFragment=**new** MeFragment();  
 *//把三个fragment放到fragments* **fragments**[0]=**indexFragment**;  
 **fragments**[1]=messageFragment;  
 **fragments**[2]=meFragment;  
  
 *//找到三个按钮* **buttons**[0]= (Button) findViewById(R.id.***btn\_main\_fragment\_store***);  
 **buttons**[1]= (Button) findViewById(R.id.***btn\_main\_fragment\_message***);  
 **buttons**[2]= (Button) findViewById(R.id.***btn\_main\_fragment\_me***);  
  
 *//设置事件监听* MyBtnListener myBtnListener=**new** MyBtnListener();  
 **for**(Button button:**buttons**)  
 {  
 button.setOnClickListener(myBtnListener);  
 }  
  
 *//设置第一个按钮的状态为选中* **buttons**[**currentFragmentIndex**].setSelected(**true**);  
  
 }  
 *//按钮事件监听的类* **class** MyBtnListener **implements** View.OnClickListener  
 {  
  
 @Override  
 **public void** onClick(View v) {  
 *//判断单击是第几个按钮 clickBtnIndex=1 fragments[1]* **switch** (v.getId())  
 {  
 **case** R.id.***btn\_main\_fragment\_store***:  
 **clickBtnIndex**=0;  
 **break**;  
 **case** R.id.***btn\_main\_fragment\_message***:  
 **clickBtnIndex**=1;  
 **break**;  
 **case** R.id.***btn\_main\_fragment\_me***:  
 **clickBtnIndex**=2;  
 **break**;  
 }  
 *//判断单击的是不是当前* **if** (**currentFragmentIndex**!=**clickBtnIndex**) {  
 *//开始事务* FragmentManager manager=getSupportFragmentManager();  
 FragmentTransaction transaction=manager.beginTransaction();  
 *//动作1：隐藏以前的fragment* Fragment hideFragment=**fragments**[**currentFragmentIndex**];  
 transaction.hide(hideFragment);  
 *//动作2:添加新的fragment* Fragment showFragment=**fragments**[**clickBtnIndex**];  
 **if** (showFragment.isAdded()==**false**)  
 {  
 transaction.add(R.id.***fragment\_container***,showFragment);  
 }  
 *//动作3:显示新的fragment* transaction.show(showFragment);  
 *//提交* transaction.commit();  
 **buttons**[**currentFragmentIndex**].setSelected(**false**);  
 **buttons**[**clickBtnIndex**].setSelected(**true**);  
 **currentFragmentIndex**=**clickBtnIndex**;  
 }  
 }  
 }  
}

在androidManifest.xml配置屏幕切换时不执行onCreate()

<activity android:configChanges="orientation|screenSize" android:name=".MainFragmentActivity"/>

# 总结



做的功能

1. 做界面用scroll-view linearLayout,EditText,button
2. 跳转用intent
3. 联网用volley,fastjson
4. 把数据保存到sqlite
5. c/s缺点是布局改了要升级，用户不想升级
6. 用了webView去加载网页，网页自适应用的了bootstrap,数据绑定用的是vue,联网用axios

# Docker容器技术

## 需求场景

Docker 提供轻量的虚拟化，通过docker镜像能创建出多个容器，一个容器相当于一个虚拟机。可以把镜像快速部署到无数台服务器上。

滴滴等公司都用了docker.





## Docker与虚拟机对比

每个虚拟器需要安装一个操作系统

虚拟机：资源全部隔离，cpu、内存、磁盘相互独立，一台宿主机中可以安装的虚拟机数量是一定的，有限的，移植性不强（文件很大，拷贝传输困难）。

Docker：部分共享部分隔离，共享cpu，内存，磁盘甚至操作系统，极为轻量,如图-126 所示。

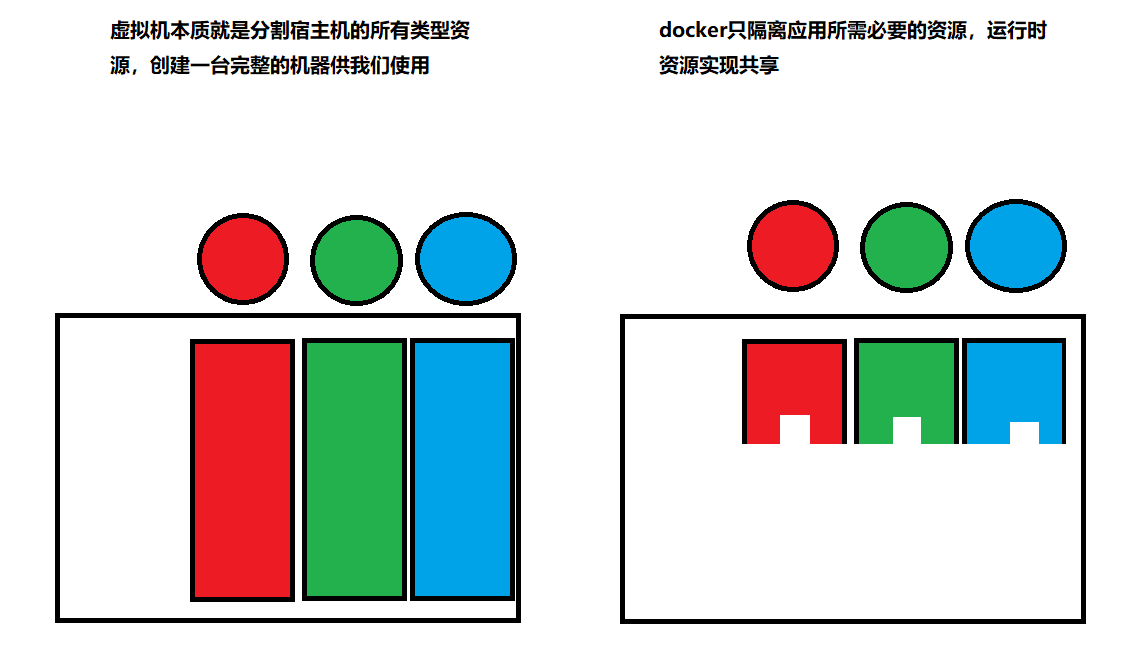
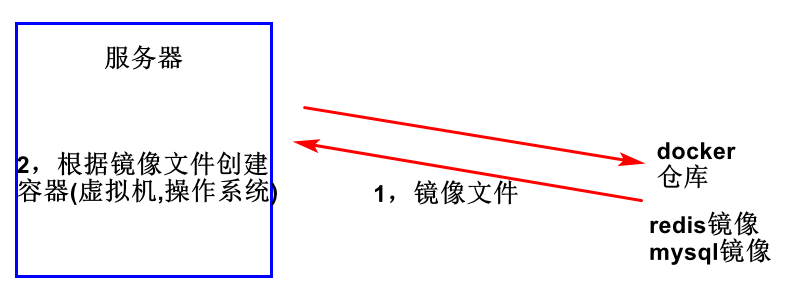


图- 126

## Docker五大要素

1. 沙箱：隔离，将应用之间的必要资源隔离开，防止互相影响，应用之间互相独立。
2. 镜像：模板，使用dockerfile制作镜像，通过镜像创建容器。
3. 容器：镜像的实例，镜像只读，容器可写，容器中可以保存应用产生的零时文件
4. 数据卷：挂载到容器上，用于保存必要数据。比如容器中数据库的数据，但是不建议用，因为重启容器需要挂载同步数据，效率较慢，解决方式：数据库拆分
5. 仓库：与maven仓库概念一致，应用都会保存在仓库中，创建容器是根据镜像的规定进行拉取，可以共享。



https://www.cnblogs.com/wushuaishuai/p/9984228.html

## Docker的版本及要求

Docker分为docker-ce（社区版-开源）和docker-ee（企业级-收费）

Docker基于linux3.8及以上版本64bit内核开发，所以在使用docker前需要确认linux内核版本，一般centos7以上。

查看linux版本命令：

uname -a

执行结果如图-127 所示.

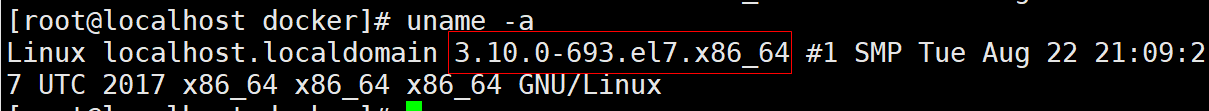


图- 127

## 虚拟机的安装

### 安装虚拟机

1.解压“\亿发,移动端课前资料\docker”文件夹中的 centos7-empty.zip，得到centos7虚拟机。

2.双击CentOS7-empty.vmx文件启动centos7，单击“获取所有权”，如图- 128所示

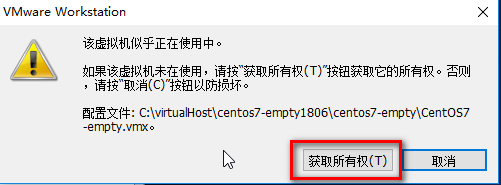


图- 128

3.在虚拟机窗口中单击输入用户名

用户名和密码都是root，如图-129 所示

Pwd:显示工作目录 print working directory

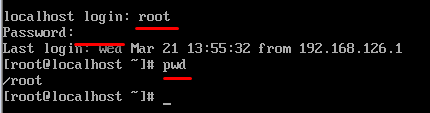
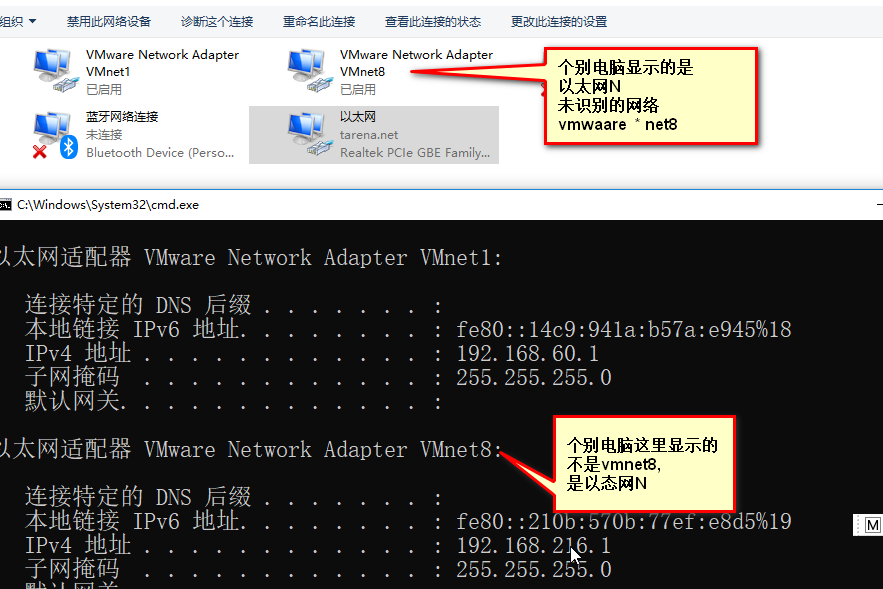


图- 129

### 修改静态IP

1. 在windows上执行ipconfig查看vmnet8的ip.



2.在虚拟机中执行vim命令修改ip,不是在xshell中执行

修改ip地址与vmnet8在同一个网络中，命令如下，先拷贝下面的命令，

vim /etc/sysconfig/network-scripts/ifcfg-ens33

然后单击vmware的编辑菜单🡪粘贴 拷贝命令到虚拟机中执行，如图-130 所示。

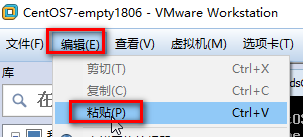


图- 130

修改ipAddr,gateway,dns, 如图-131 所示。

gateway，dns的前三段与ip的前三段一样，最后一段是2。

IPADDR，GATEWAY,DNS1是关键字，不要误操作删除了字母或添加了字母。

按i进入编辑模式，编辑ipAddr,GATEWAy,DNS1

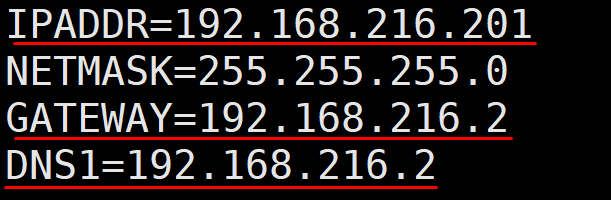


图- 131

按esc进入命令模式，

输入:wq,注意:wq显示在屏幕最下面。

3.执行下面的命令重启网络服务，让ip地址生效

service network restart

使用xshell连接centos7虚拟机

xshell中执行 ping [www.tmooc.cn](http://www.tmooc.cn)

Ctrl+c退出

### 关闭防火墙

systemctl stop firewalld.service #关闭防火墙服务

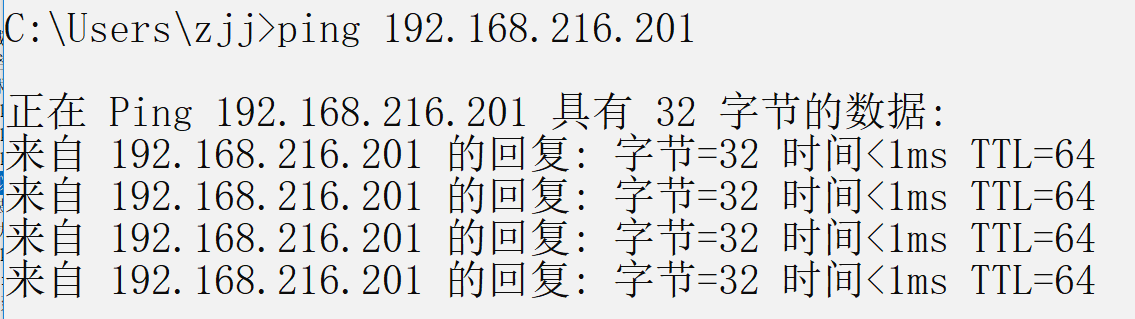
systemctl disable firewalld.service #禁止防火墙开启启动

firewall-cmd --state #检查防火墙状态

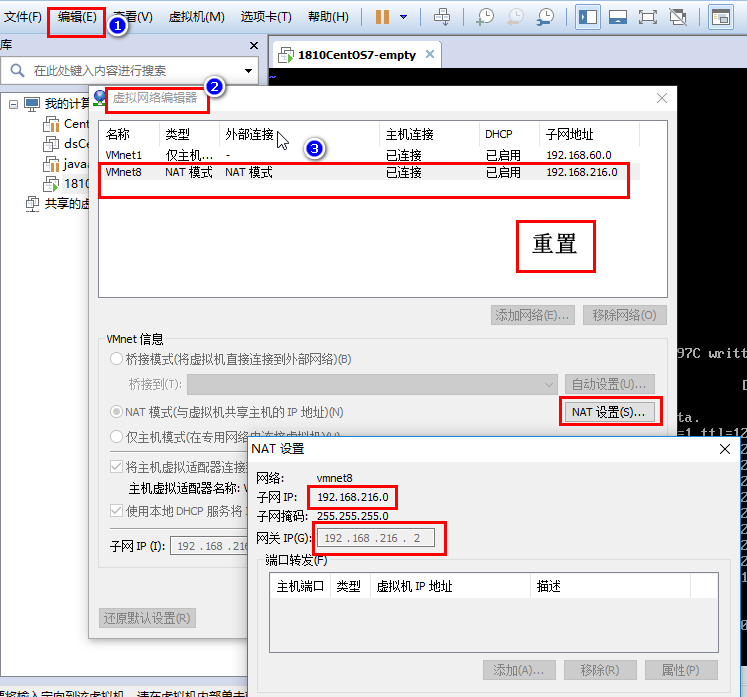
not running

### 测试防火墙是否关闭

在windows中ping虚拟机



## 如果网络出问题了，要重置vmnet8



## 安装docker

### 验证docker是否安装好

虚拟机中的docker已经装好了, 执行docker version 命令查看docker的版本。

[root@localhost ~]# docker version

Client:

Version: 17.12.1-ce

API version: 1.35

Go version: go1.9.4

Git commit: 7390fc6

Built: Tue Feb 27 22:15:20 2018

OS/Arch: linux/amd64

Cannot connect to the Docker daemon at unix:///var/run/docker.sock. Is the docker daemon running?

## 启动 docker

//查看docker状态

systemctl status docker

Active: inactive (dead)

//启动docker

systemctl start docker

systemctl status docker

//停止docker

service docker stop

//重启docker

service docker restart

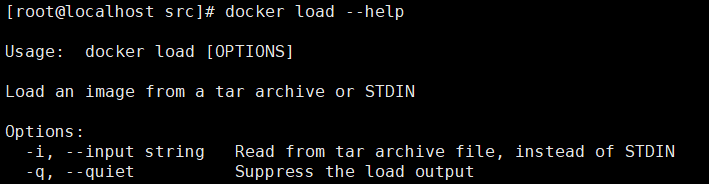
## docker命令

|  |  |
| --- | --- |
| 命令 | 用法 |
| yum -y install docker-ce | 下载最新版的docker |
| service docker start | 启动Docker服务 |
| service docker stop | 停止Docker服务 |
| service docker restart | 重新启动Docker服务 |
| docker version | 查看Docker的版本号 |
| docker pull 镜像地址:版本 | 从镜像仓库中下载 |
| docker save a2a69ca5184a > jt-centOS6.tar | 根据镜像id导出镜像 |
| docker save -o redis-3.2.8.tar redis:3.2.8 | 根据镜像名称导出镜像 |
| docker load -i docker-centos-6.5.tar | 指定jar包导入镜像文件 |
| docker rmi a2a69ca5184a | 根据Id号删除镜像文件 |
| docker rmi -f a2a69ca5184a | 强制删除镜像文件 删除镜像前需要先关闭容器 |
| docker images | 查询所有镜像文件 |
| docker inspect index.alauda.cn/tutum/centos:6.5 | 查看镜像文件细节信息 |
| docker tag 旧镜像名称和端口 redis-ali:0.0.1 | 修改镜像的名称 |
| docker build -t 镜像名称:版本号 | 根据dockerfile来创建镜像文件 |
| docker run -d --name 容器名 镜像名:版本号 | 根据镜像名称启动容器 |
| docker run -d --name 容器名(自定) 镜像id号 | 根据镜像id启动容器 |
| docker run -d -p 虚拟机端口:镜像端口 --name 容器名 镜像名:版本号 | 启动容器,并指定暴露端口 |
| docker ps | 查看活动的docker容器进程 |
| Docker ps -a/-all | 查看全部的容器 |
| docker exec -it 容器id bash | 进入指定的容器 |
| docker stop 容器Id号 | 停止指定容器 |
| docker start 容器Id号 | 启动创建好的容器 |
| docker stop $(docker ps -q) & docker rm $(docker ps -aq) | 关闭和删除所有的容器 |
| docker rm 容器Id | 删除指定的容器 |

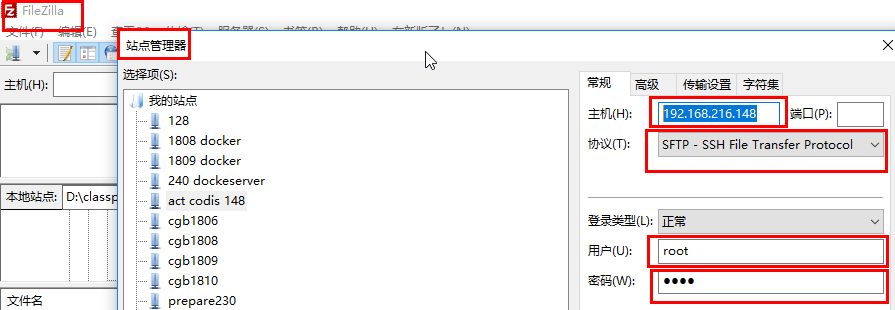
## Redis案例

### 加载镜像

#### 查看帮助文档



#### 上传文件



使用fz进入/usr/local/src创建redis文件夹

上传\亿发,移动端课前资料\docker\ dockerImage\redis-3.2.8.tar到/usr/local/src/redis

#### 加载镜像

cd /usr/local/src/redis

docker load -i redis-3.2.8.tar

#### 查看镜像

docker images

执行结果如图-134 所示

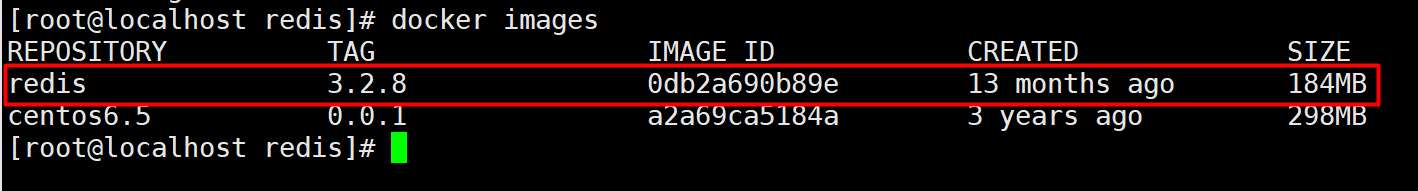


图- 134

### 创建容器

docker run -d --name redis7000 -p 7000:6379 redis:3.2.8

参数说明：

-d，则containter将会运行在后台模式(Detached mode)

--name 实例名称

-p 对外程序访问端口7000，宿主机映射的redis端口6379

最后的redis为镜像的名称

访问过程如图-135 所示.

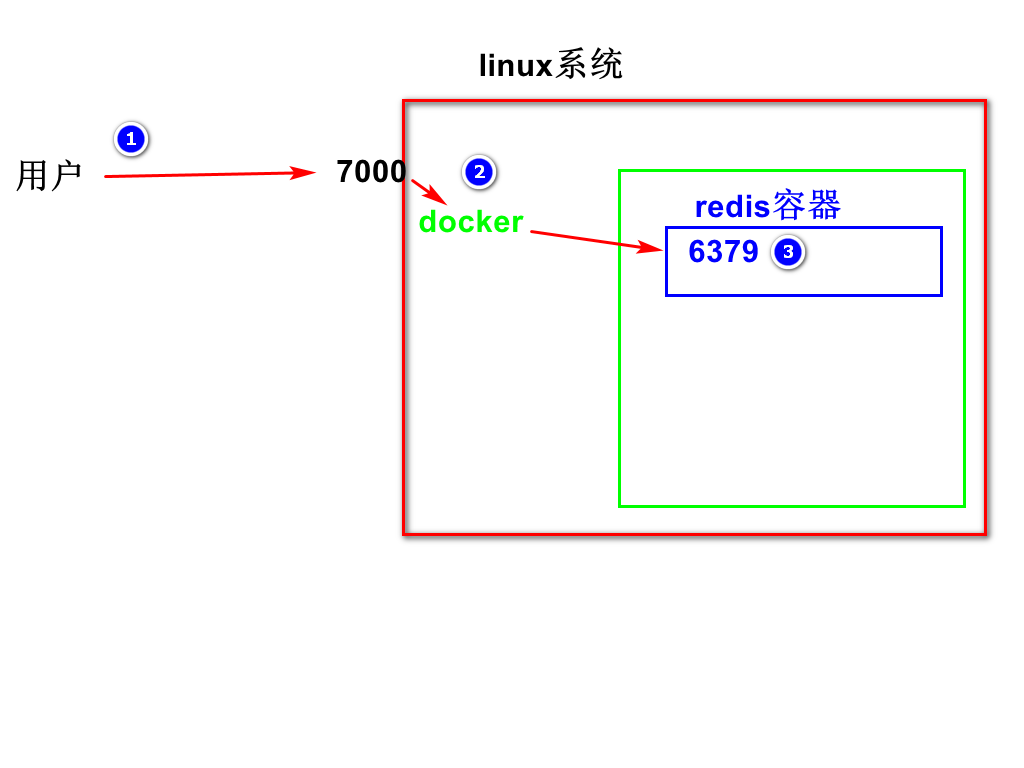


图- 135

查看容器

docker ps -a

显示结果如图-136 所示

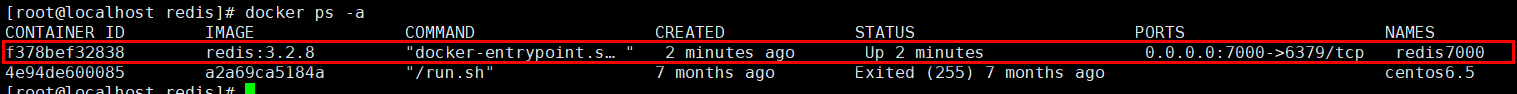


图- 136

创建容器出的错误

IPv4 forwarding is disabled. Networking will not work.

# vim /usr/lib/sysctl.d/00-system.conf

* 1添加如下代码：

net.ipv4.ip\_forward=1

* 1

重启network服务

# systemctl restart network

### 删除容器

docker stop redis7000

//容器状态是exit

docker ps -a

docker rm redis7000

docker ps -a

docker run -d --name redis7000 -p 7000:6379 redis:3.2.8

docker stop redis7000

docker start redis7000

### 测试单个节点

创建maven quick start项目，项目名pd-redis。

pom.xml中把junit版本改成4.12,添加jedis依赖

<!-- jedis -->

<dependency>

<groupId>redis.clients</groupId>

<artifactId>jedis</artifactId>

<version>2.6.2</version>

</dependency>

创建测试类 JedisTest，拷贝下面的方法

@Test //完成单实例链接,修改代码中ip地址

public void jedis(){

Jedis jedis = new Jedis("192.168.163.30", 7000);

//jedis.auth("123456");

jedis.set("name", "tony"); //调用redis命令set

String s = jedis.get("name");

System.out.println(s);

jedis.close();

}

### 查看服务器上的数据

进入到redis7000容器中，进入到容器中执行命令

[root@localhost redis]# docker exec -it redis7000 bash

root@9cc7930f4c69:/data# redis-cli

127.0.0.1:6379> keys \*

1) "name"

127.0.0.1:6379> get name

"tony"

//退出redis客户端

127.0.0.1:6379> exit

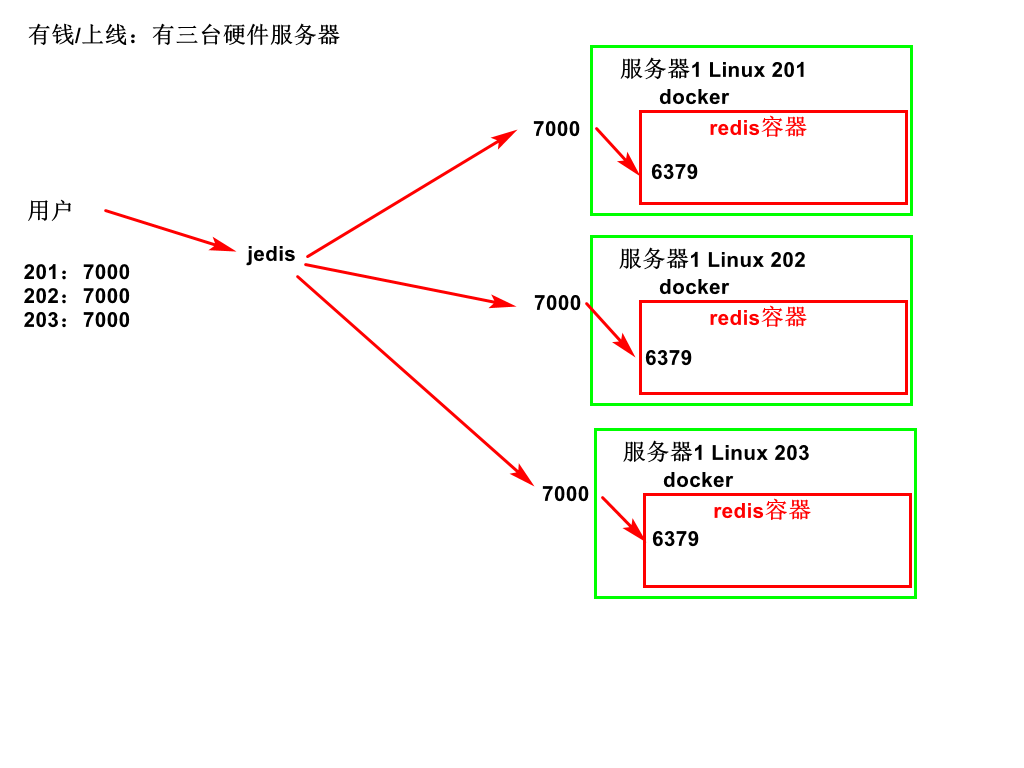
//退出容器

root@9cc7930f4c69:/data# exit

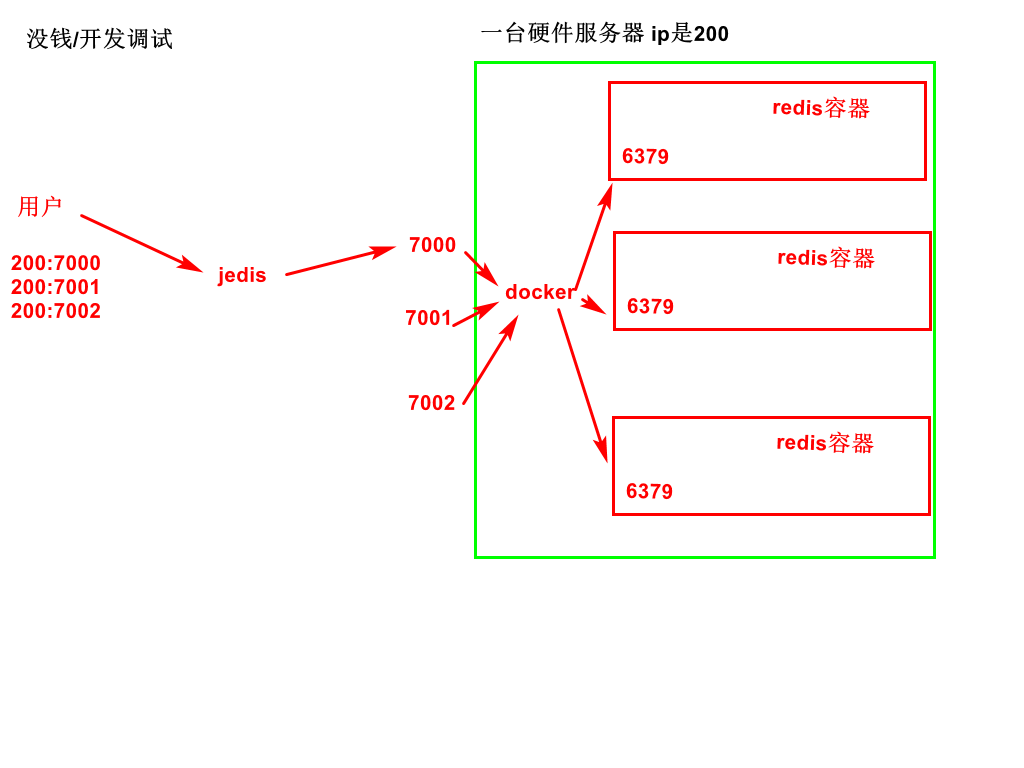
exit

[root@localhost redis]#

### 创建多个redis容器



访问过程如图- 137所示



使用镜像创建容器保证开发调试阶段和上线后的环境一致

图- 137

docker run -d --name redis7001 -p 7001:6379 redis:3.2.8

docker run -d --name redis7002 -p 7002:6379 redis:3.2.8

### 查看容器

docker ps -a

显示的结果如图-138 所示

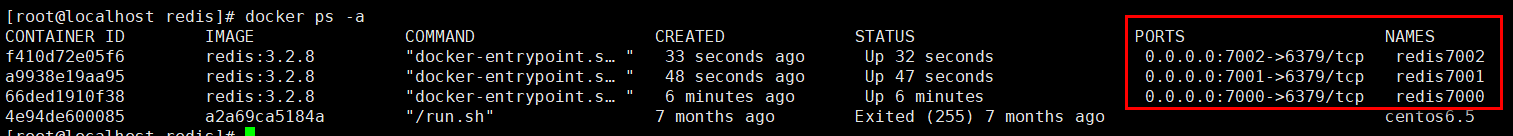


图- 138

### 测试分片

拷贝代码

//redis测试分片

@Test

public void test02(){

//2.创建分片的连接池

JedisPoolConfig poolConfig = new JedisPoolConfig();

poolConfig.setMaxTotal(500);

poolConfig.setMaxIdle(20);

//3.准备redis的分片

List<JedisShardInfo> shards = new ArrayList<JedisShardInfo>();

shards.add(new JedisShardInfo("192.168.65.110", 7000));

shards.add(new JedisShardInfo("192.168.65.110", 7001));

shards.add(new JedisShardInfo("192.168.65.110", 7002));

//1.创建分片的对象

ShardedJedisPool jedisPool =

new ShardedJedisPool(poolConfig, shards);

//获取jedis对象

ShardedJedis shardedJedis = jedisPool.getResource();

//5.redis的存取值操作

for (int i = 0; i < 9; i++) {

shardedJedis.set("n"+i,"我是分片操作"+i);

}

}

### 进入3个容器内部查看数据

//进入到第一个redis容器，查看数据

[root@localhost redis]# docker exec -it redis7000 bash

root@4fb42d95757f:/data# redis-cli

127.0.0.1:6379> keys \*

1) "n8"

2) "n0"

3) "n6"

4) "n3"

127.0.0.1:6379> exit

root@4fb42d95757f:/data# exit

exit

//进入到第二个redis容器，查看数据

[root@localhost redis]# docker exec -it redis7001 bash

root@1418ed384483:/data# redis-cli

127.0.0.1:6379> keys \*

1) "n2"

127.0.0.1:6379> exit

root@1418ed384483:/data# exit

exit

//进入到第三个redis容器，查看数据

[root@localhost redis]# docker exec -it redis7002 bash

root@58de528a1c05:/data# redis-cli

127.0.0.1:6379> keys \*

1) "n5"

2) "n1"

3) "n7"

4) "n4"

127.0.0.1:6379> exit

root@58de528a1c05:/data# exit

### 删除3个容器重新创建

#查看所有容器

docker ps -a

#停止实例

docker stop redis7000

#开启实例

docker start redis7000

#删除容器,先要停止容器

docker stop redis7000

docker rm redis7000

docker ps -a

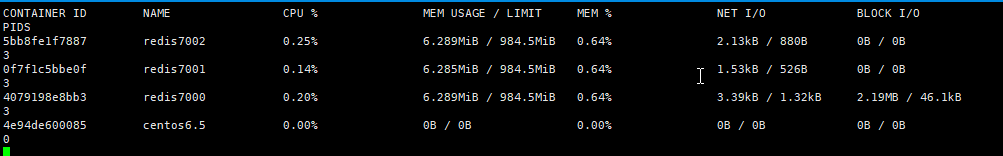
docker run -d --name redis7000 -p 7000:6379 redis:3.2.8

docker run -d --name redis7001 -p 7001:6379 redis:3.2.8

docker run -d --name redis7002 -p 7002:6379 redis:3.2.8

查看容器使用的cpu,内存

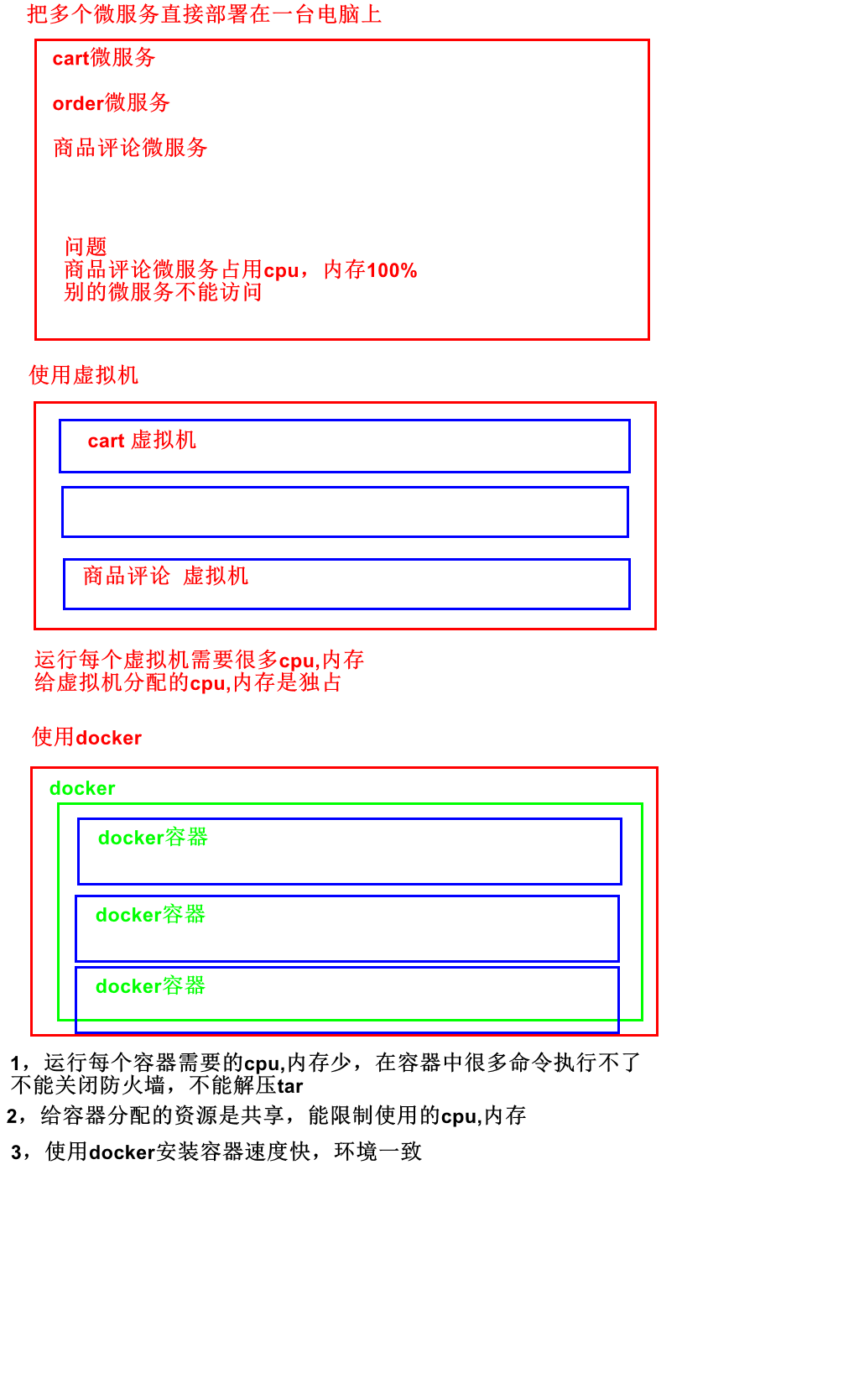


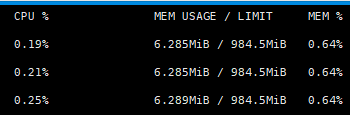


退出 ctrl+c

### Docker小结

在有多台服务器时使用docker的优点。



docker stats 查看每个容器使用的cpu,内存

ctrl+c退出

## 创建网络

不指定ip地址tomcat无法访问mysql，如图-139 所示

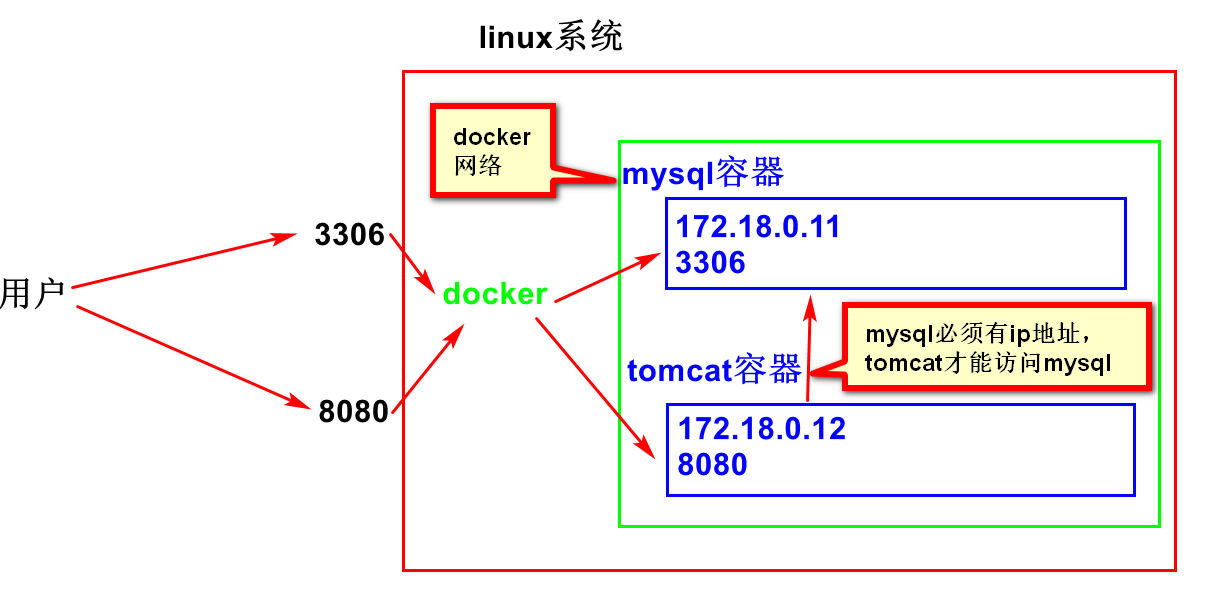
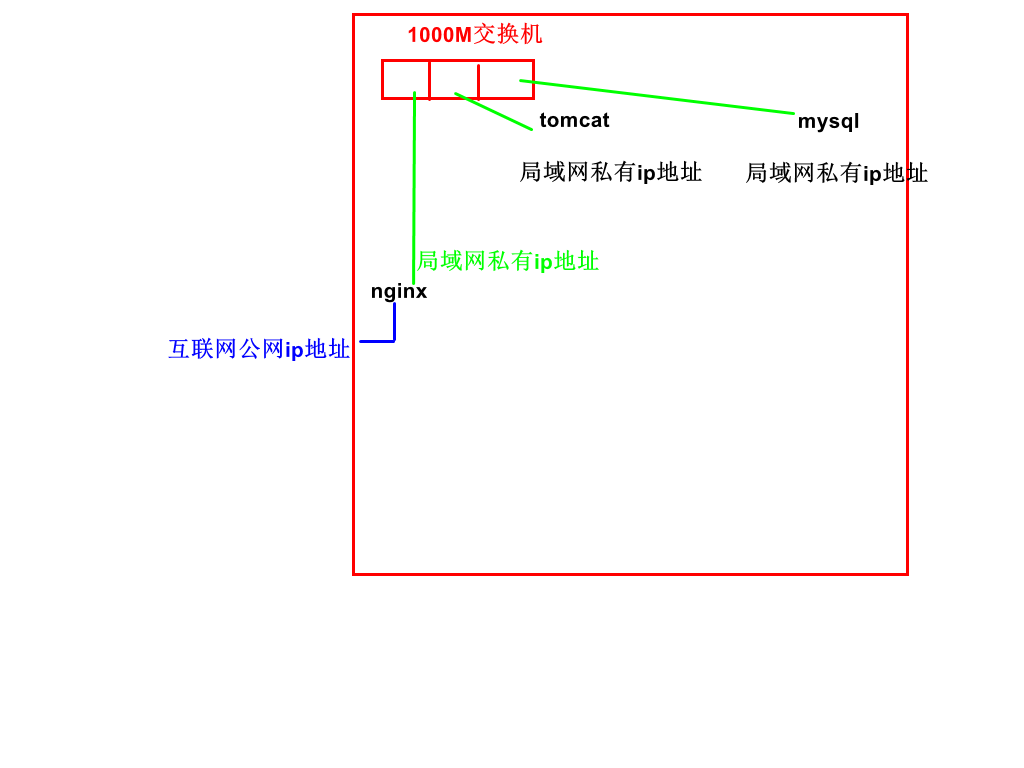


图- 139

Tomcat,Mysql配置成独立服务器



### 案例

### 查看ip地址

[root@localhost ~]# ifconfig

docker0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500

inet 172.17.0.1 netmask 255.255.0.0 broadcast 172.17.255.255

ens33: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500

inet 192.168.216.201 netmask 255.255.255.0 broadcast 192.168.216.255

### 创建网络

执行docker network create --subnet=172.18.0.0/24 dockernet创建一个网络

docker network create --subnet=172.18.0.0/24 dockernet

a5cd262aeff63cdeefbab3f768ccc542a635c658b51e4e95307b03adb05a69d0

//执行ifconfig，查看ip信息，发现多了一个172.18.0.1 IP

[root@localhost ~]# ifconfig

br-a5cd262aeff6: flags=4099<UP,BROADCAST,MULTICAST> mtu 1500

inet 172.18.0.1 netmask 255.255.255.0 broadcast 172.18.0.255

docker0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500

inet 172.17.0.1 netmask 255.255.0.0 broadcast 172.17.255.255

ens33: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500

inet 192.168.216.201 netmask 255.255.255.0 broadcast 192.168.216.255

## 安装mysql容器

### 上传文件

上传“\培优课前资料\亿发,移动端课前资料\docker\dockerImage\”中的mysql-image.tar到/usr/local/src文件夹中。

确认上传完成后，再执行下面的命令，否则加载mysql镜像失败

### 加载mysql镜像

[root@localhost ~]# cd /usr/local/src

[root@localhost src]# ls

mysql-image.tar redis

//加载镜像

[root@localhost src]# docker load -i mysql-image.tar

//查看镜像，刚加载的镜像的repository,tag为none

[root@localhost src]# docker images

REPOSITORY TAG IMAGE ID CREATED SIZE

<none> <none> 44a8e1a5c0b2 16 months ago 407MB

//设置镜像的名称

[root@localhost src]# docker tag 44a8e1a5c0b2 docker.io/mysql:5.7

//再次查看镜像

[root@localhost src]# docker images

REPOSITORY TAG IMAGE ID CREATED SIZE

mysql 5.7 44a8e1a5c0b2 16 months ago 407MB

### 创建mysql容器

//启动mysql docker容器，设置root密码，设置默认编码，指定端口映射。

[root@localhost src]# docker run --net dockernet --ip 172.18.0.11 --name=mysql-container -it -d -p 3306:3306 -e MYSQL\_ROOT\_PASSWORD=root mysql:5.7 --character-set-server=utf8

[root@localhost src]# docker ps -a

//查看mysql的ip地址

[root@localhost src]# docker inspect mysql-container

[

"Networks": {

"dockernet": {

"IPAMConfig": {

"IPv4Address": "172.18.0.11"

},

]

### 导入数据库

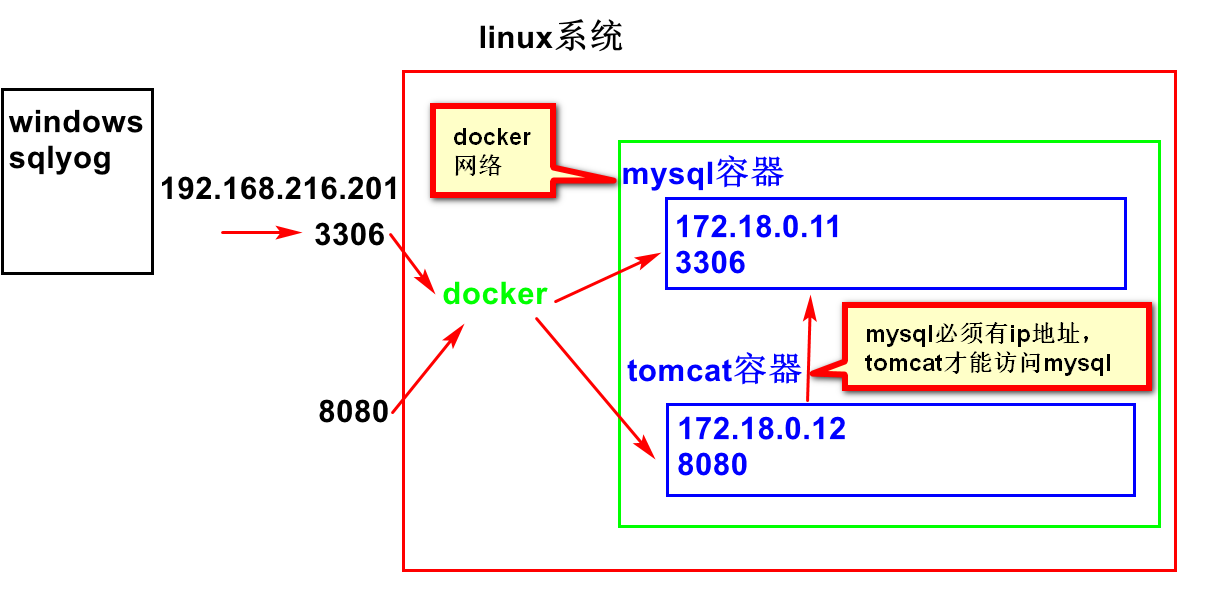
执行过程如图-140 所示

图- 140

在windows启动sqlyog客户端，导入数据,Ip地址是虚拟机的ip

关闭以前的连接，创建新的连接，配置信息如图-141 所示

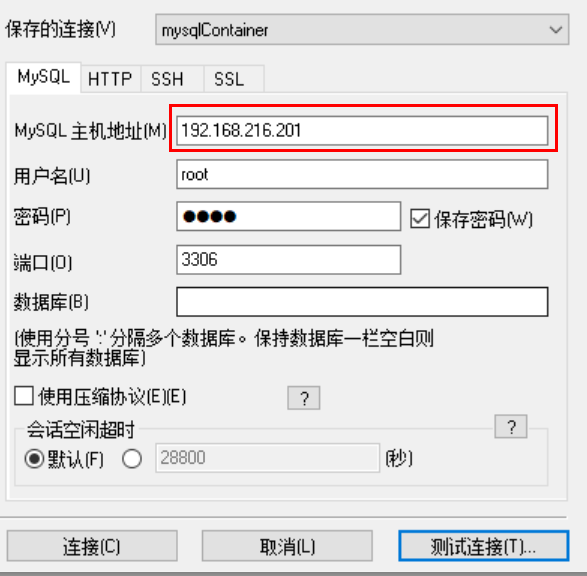


图- 141

连接成功后，只显示4个数据库，如图- 142所示

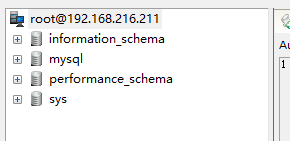


图- 142

导入数据库，如图-143 所示。

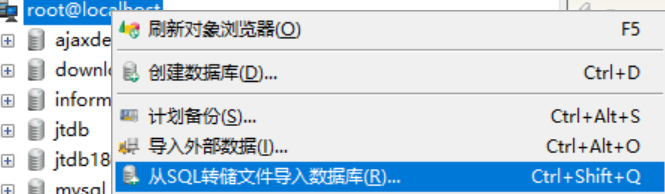


图- 143

执行“\培优课前资料\亿发,移动端课前资料\docker\EasyMall”文件夹下的easymall.sql

刷新数据库，如图-144 所示。



图- 144

刷新后能看到easymall数据库，如图-145 所示。

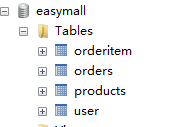


图- 145

### 如查连不上mysql,报10060或10061,解决办法如下

先把虚拟机关机poweroff,再开机，xshell重新打开连接

systemctl stop firewalld.service

systemctl disable firewalld.service

systemctl start docker

docker start mysql-container

## 安装tomcat容器

### 服务器结构图，如图-146 所示

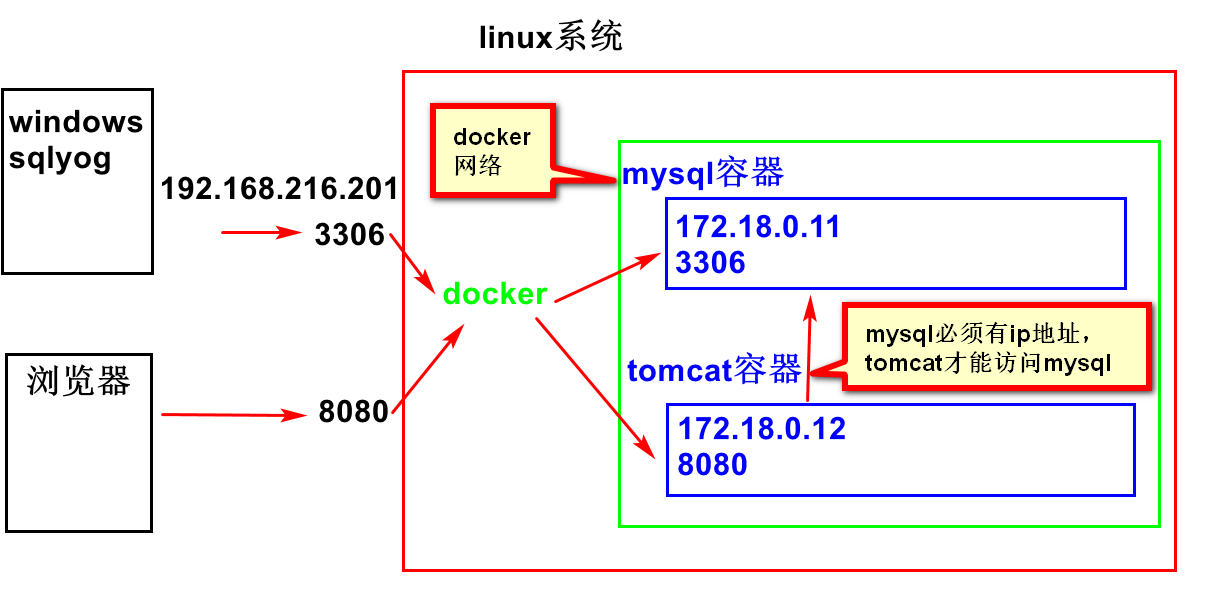


图- 146

### 上传镜像

上传“\培优课前资料\亿发,移动端课前资料\docker\dockerImage\”中的tomcat7-jre8-image.tar到/usr/local/src文件夹中。

### 安装镜像

[root@localhost ~]# cd /usr/local/src

[root@localhost src]# ls

mysql-image.tar redis tomcat7-jre8-image.tar

[root@localhost src]# docker load -i tomcat7-jre8-image.tar

[root@localhost src]# docker images

[root@localhost src]# docker tag 510e45f8f9f7 docker.io/tomcat:7.0

[root@localhost src]# docker images

### 创建容器

docker run -d -p 8080:8080 --net dockernet --ip 172.18.0.12 --name easyMall-tomcat tomcat:7.0

docker ps -a

windows浏览器中访问<http://虚拟机ip:8080>，能看到tomcat首页。

## 部署easymall

### 修改数据库链接地址

* 1. 在资源管理器中设置打开xml的软件

创建1.xml,双击1.xml,测试 一下是不是用你喜欢的编辑软件打开的。如果不是，按照下图设置打开xml的编辑软件，如图- 147所示。



图- 147

* 1. 将“\培优课前资料\亿发,移动端课前资料\docker\EasyMall”文件夹中的ROOT.war改成ROOT.rar,用winrar直接打开，如图-148 所示。

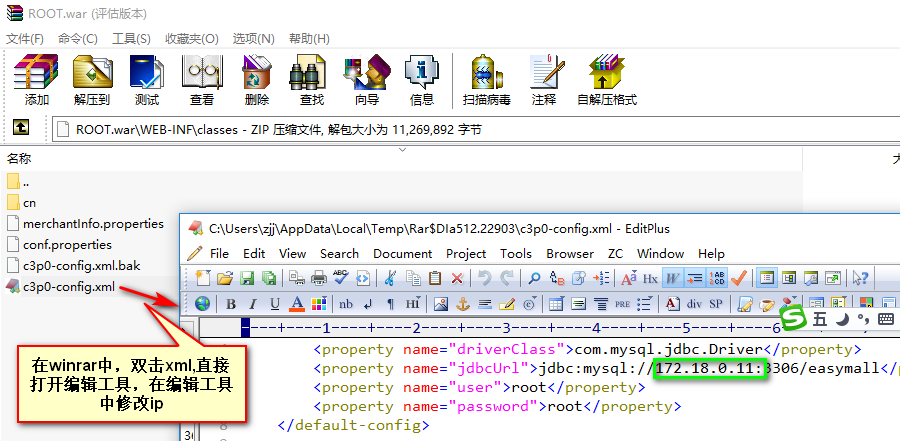


图- 148

将ROOT文件中的\WEB-INF\classes\c3p0-config.xml文件修改,修改数据库ip地址为mysql容器的ip地址172.18.0.11。保存修改，

<property name="jdbcUrl">

jdbc:mysql://172.18.0.11:3306/easymall

</property>

3.使用winrar再次打开ROOT.rar,查看ip地址是否修改过来了。

4.将ROOT.rar改成ROOT.war

5.上传ROOT.war到/usr/local/src文件夹中。

### 查看tomcat容器文件夹目录

docker ps -a

[root@localhost src]# docker exec -it easyMall-tomcat bash

root@c620df99d5e5:/usr/local/tomcat# cd /usr/local/tomcat

root@c620df99d5e5:/usr/local/tomcat# ls

LICENSE RELEASE-NOTES bin include logs temp work

NOTICE RUNNING.txt conf lib native-jni-lib webapps

root@c620df99d5e5:/usr/local/tomcat# cd webapps

root@c620df99d5e5:/usr/local/tomcat/webapps# ls

ROOT docs examples host-manager manager

root@c620df99d5e5:/usr/local/tomcat/webapps# exit

exit

[root@localhost src]#

### 上传root.war到tomcat容器中

[root@localhost src]# cd /usr/local/src

[root@localhost src]# ls

[root@localhost src]# docker cp ROOT.war easyMall-tomcat:/usr/local/tomcat/webapps/

浏览器中访问<http://虚拟机ip:8080/>，如图-149 所示



图- 149

如果root.war没有自动解压，重启容器

# docker stop easyMall-tomcat

# docker start easyMall-tomcat

### 为什么用容器

