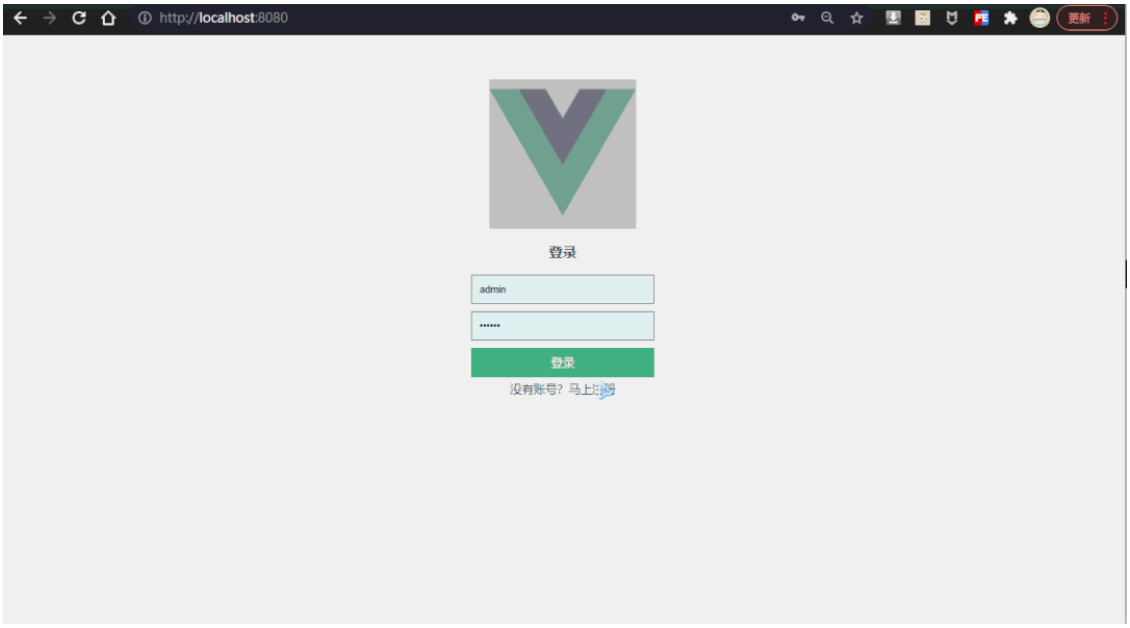
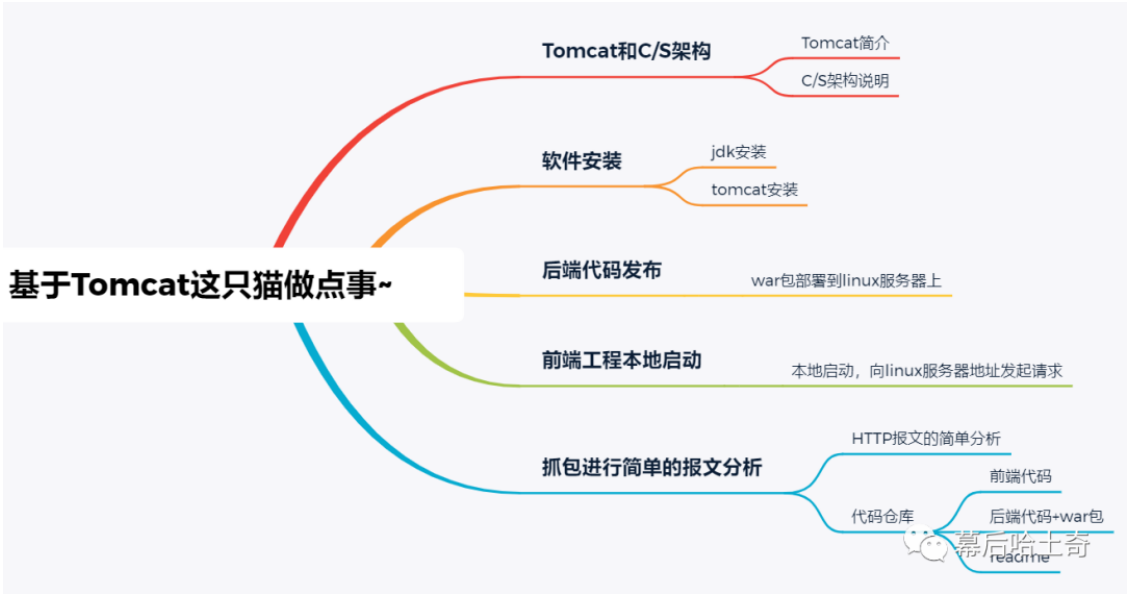
经过前面的折腾，虚拟机已经准备好了，并且网络也是互通的，下面我们就要干点事情了，本篇文章我将简单介绍下web服务器的代表人物：tomcat，我们实现一个小目标：

- 实现一个用户登录和注册的demo
- 使用比较主流的前后端分离架构

完成效果：



或许有读者朋友以为走错片场了，本篇文章主要是写给不清楚后端服务器是怎么玩的同学，直观给出一个展示，便于后续对HTTP协议的理解，比如请求发给服务端，神秘的后端长什么样子？前端又是什么样子？这一切本篇通过这个demo呈现出来，如果你感兴趣，不用关心代码是如何实现的，只需要知道如何部署和验证自己的想法。本文大纲：

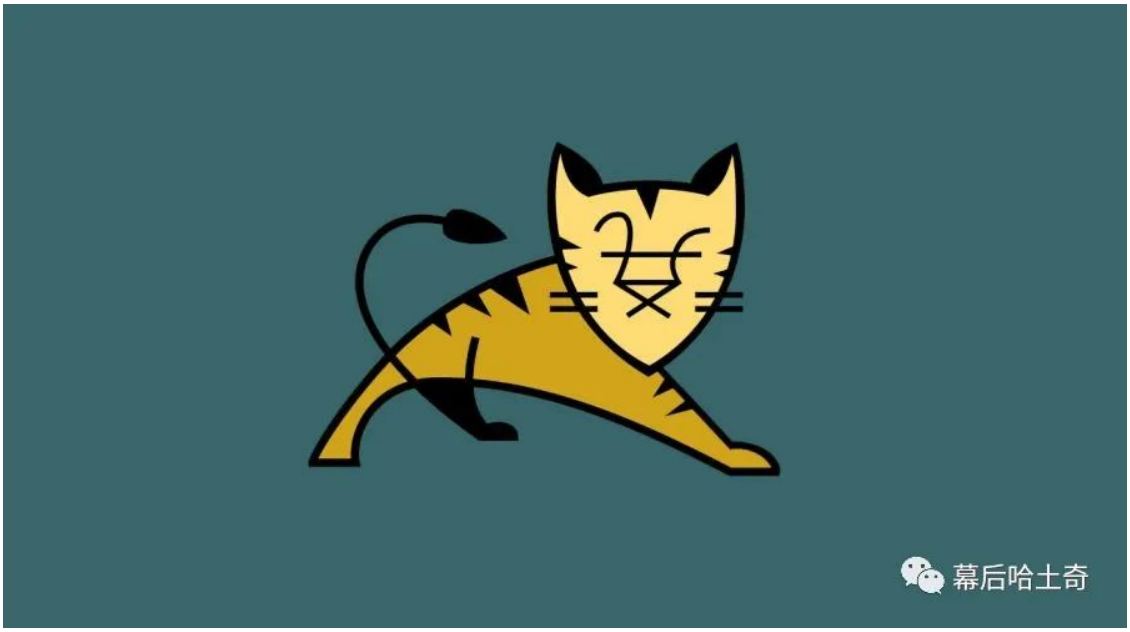


下面我们开始实现下小demo，直观体验HTTP协议！

如果你觉得本篇文章无用，直接跳过即可，不影响后续的学习。

一、Tomcat和C/S架构

Tomcat是Apache软件基金会组织下的开源项目，在中小型系统和并发量小的场合下被普遍使用，是开发和调试Servlet、JSP 程序的首选。



> logo是一只猫，汤姆猫是为人熟知的、叫的上名字的公猫，并且汤姆猫与tomcat发音相似，也同样让开发者们喜爱，叫起来有种熟悉自然的感觉。

在说tomcat作用前，我们先来说下C/S架构，即client /server（客户端 / 服务器）架构。

服务器的英语单词是 server，server 的原意是 “服务生，侍者”，生活中的例子就是餐厅里为顾客服务的人。客户端的英语单词是 client，client 的原意是 “顾客”。

因此，服务器的特性，顾名思义就是为 client（客户端）提供服务（service）。例如，如果我们以 Web 服务器（web 表示 “网络”）为例，其作用就是为网民们提供网页。电子邮件服务器的作用是提供电子邮件地址，以及提供发送和接收电子邮件的服务。

同理，我们的tomcat就可以作为一个服务生，7*24小时不间断为顾客提供服务，就是所谓的“服务器一直监听”，随时准备接收和响应客户端的请求。

这个很容易理解，毕竟如果王者荣耀的服务端说我上班时间为朝9晚6，其他时间不接待，用户不得天天骂街。

在这种架构下，99%的场景请求都是由客户端主动发起，就像客户自己饿了才会走进餐厅一样，不过，也会有服务端主动往客户端推送的场景，最经典的就是聊天室，A发给B的消息，要经过A-服务器-B，因此，第一步是客户端请求服务端，第二步是服务端主动推动给客户端，我们称之为服务端主动推送，一般可以通过websocket这样的技术来实现。

二、Tomcat安装

在介绍完tomcat的角色定位后，我们完成了服务端选角，由于tomcat是一个软件，跟其他软件没有啥区别，需要将它安装到我们上节说的centos中，然后启动一直让其监听即可，只要服务端不断电，tomcat就要一直工作。好了，废话不多说，开始安装。

随便挑一台装好的服务器，在确认能访问互联网后，我们开始动手，在安装tomcat之前，需要先安装JDK，我们选择的版本是JDK8版本。

①上传jdk和tomcat安装包

所以我们先需要去下载软件，并且上传到服务器上，这一步我就不过多赘述了。

```
[root@localhost ~]# ll
total 190388
-rw-------. 1 root root      1452 May  5 04:31 anaconda-ks.cfg
-rw-r--r--. 1 root root   9433364 Sep  4  2017 apache-tomcat-8.5.20.tar.gz
-rw-r--r--. 1 root root 185515842 Sep  3  2017 jdk-8u144-linux-x64.tar.gz
[root@localhost ~]#
```


②检查服务器是否已安装了默认的OPENJDK，有则需要移除

有的系统可能会默认帮助你安装好OpenJDK，如下：

```
[root@centos7-basic ~]# java -version
openjdk version "1.8.0_181"
OpenJDK Runtime Environment (build 1.8.0_181-b13)
OpenJDK 64-Bit Server VM (build 25.181-b13, mixed mode)
```


如果如上图一样有，则需要卸载。

第一步：检查系统安装的OpenJDK：`rpm -qa|grep openjdk -i`

```
[root@centos7-basic software]# rpm -qa|grep openjdk -i
java-1.7.0-openjdk-1.7.0.191-2.6.15.5.el7.x86_64
java-1.7.0-openjdk-headless-1.7.0.191-2.6.15.5.el7.x86_64
java-1.8.0-openjdk-headless-1.8.0.181-7.b13.el7.x86_64
java-1.8.0-openjdk-1.8.0.181-7.b13.el7.x86_64
```


第二步：删除以上几项，再用上面的命令检查还是否存在：`rpm -e --nodeps 需要删除的软件`

```
[root@centos7-basic software]# rpm -e --nodeps java-1.7.0-openjdk-1.7.0.191-2.6.15.5.el7.x86_64
[root@centos7-basic software]# rpm -e --nodeps java-1.7.0-openjdk-headless-1.7.0.191-2.6.15.5.el
[root@centos7-basic software]# rpm -e --nodeps java-1.8.0-openjdk-headless-1.8.0.181-7.b13.el7.x
[root@centos7-basic software]# rpm -e --nodeps java-1.8.0-openjdk-1.8.0.181-7.b13.el7.x86_64
[root@centos7-basic software]# rpm -qa|grep openjdk -i
[root@centos7-basic software]#
```

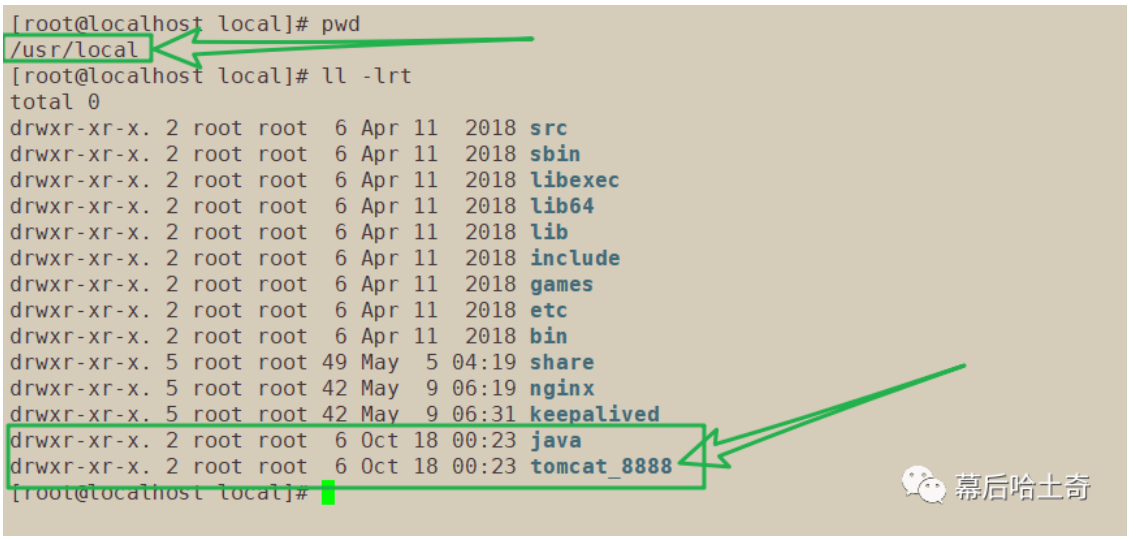

下面就开始安装我们的JDK。

③安装JDK

第一步是解压压缩包。

```
tar -zxvf jdk-8u144-linux-x64.tar.gz
```

习惯将软件安装在 `usr/local` 下。因此我在此目录下又新建了两个目录备用：



当然了，我们先关注JDK，我把刚才解压的文件转移到这个 `java` 目录下即可：

```
mv jdk1.8.0_144/ /usr/local/java/
```

即当前的JAVA安装路径为： `/usr/local/java/jdk1.8.0_144`

下面就是配置环境变量：

```
vim /etc/profile
```

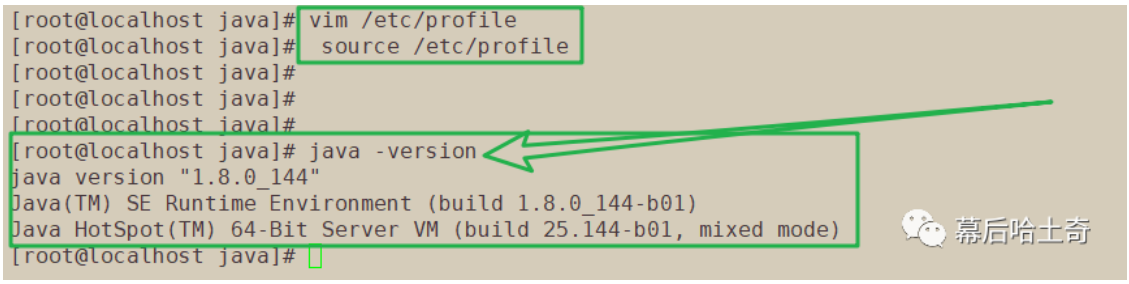
在最后追加三行：

```
export JAVA_HOME=/usr/local/java/jdk1.8.0_144
export CLASSPATH=.:%JAVA_HOME%/lib/dt.jar:%JAVA_HOME%/lib/tools.jar
export PATH=$PATH:$JAVA_HOME/bin
```

保存后执行：

```
source /etc/profile
```

最后查看安装效果：



大功告成！下面继续来安装 `tomcat` 。

④解压安装tomcat

解压安装包：

```
tar -zxvf apache-tomcat-8.5.20.tar.gz
```

挪到 `usr/local/tomcat_8888/` 下：

```
mv apache-tomcat-8.5.20 /usr/local/tomcat_8888
```

OK了，我们我们来启动tomcat看能不能访问到默认的欢迎页面。

首先来到tomcat的目录：

```
cd /usr/local/tomcat_8888/apache-tomcat-8.5.20/bin
```

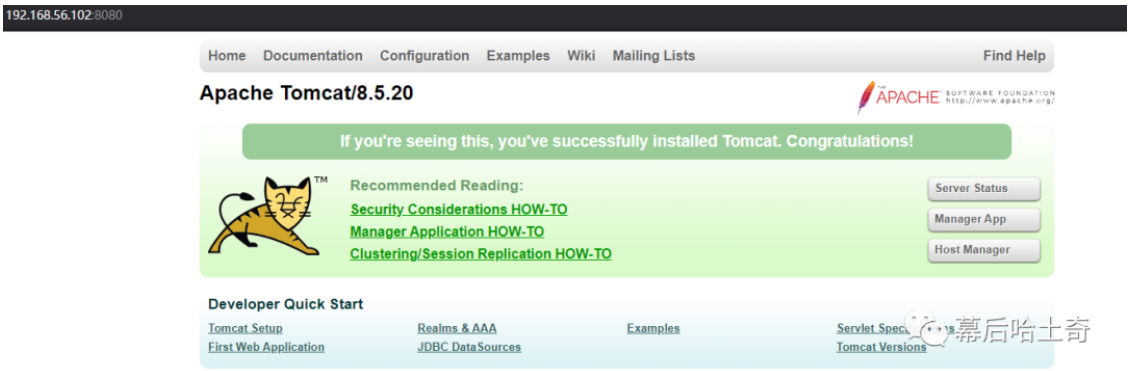
执行：

```
./startup.sh
```

在浏览器输入：

```
http://192.168.56.102:8080
```

成功页面如下：



这里可能存在一个问题：输入后无法访问，看下防火墙是否已关闭，未关闭则无脑关闭即可：

```
[root@localhost bin]# service iptables status
Redirecting to /bin/systemctl status iptables.service
Unit iptables.service could not be found.
[root@localhost bin]# systemctl status firewall
Unit firewall.service could not be found.
[root@localhost bin]# systemctl status firewalld
● firewalld.service - firewall - dynamic firewall daemon
   Loaded: loaded (/usr/lib/systemd/system/firewalld.service; enabled; vendor preset:
 enabled)
   Active: active (running) since Sun 2021-10-17 09:53:27 EDT; 14h ago
     Docs: man:firewalld(1)
 Main PID: 681 (firewalld)
   CGroup: /system.slice/firewalld.service
           └─681 /usr/bin/python2 -Es /usr/sbin/firewalld --nofork --nopid

Oct 17 09:53:25 localhost.localdomain systemd[1]: Starting firewalld - dynamic fi....
Oct 17 09:53:27 localhost.localdomain systemd[1]: Started firewalld - dynamic fir....
Oct 17 09:53:28 localhost.localdomain firewalld[681]: WARNING: AllowZoneDrifting ....
Hint: Some lines were ellipsized, use -l to show in full.
[root@localhost bin]# systemctl stop firewalld
[root@localhost bin]# systemctl status firewalld
● firewalld.service - firewall - dynamic firewall daemon
   Loaded: loaded (/usr/lib/systemd/system/firewalld.service; enabled; vendor preset:
 enabled)
   Active: inactive (dead) since Mon 2021-10-18 00:35:25 EDT; 2s ago
     Docs: man:firewalld(1)
  Process: 681 ExecStart=/usr/sbin/firewalld --nofork --nopid $FIREWALLD_ARGS (code=e
xited, status=0/SUCCESS)
 Main PID: 681 (code=exited, status=0/SUCCESS)

Oct 17 09:53:25 localhost.localdomain systemd[1]: Starting firewalld - dynamic fi....
Oct 17 09:53:27 localhost.localdomain systemd[1]: Started firewalld - dynamic fir....
Oct 17 09:53:28 localhost.localdomain firewalld[681]: WARNING: AllowZoneDrifting ....
Oct 18 00:35:24 localhost.localdomain systemd[1]: Stopping firewalld - dynamic fi....
Oct 18 00:35:25 localhost.localdomain systemd[1]: Stopped firewalld - dynamic fir....
Hint: Some lines were ellipsized, use -l to show in full.
[root@localhost bin]#
```


⑤更改监听端口号

从tomcat命名可以看出，我们希望服务器监听的端口是8888，但是tomcat默认使用的是8080端口，我们怎么改呢?

修改文件：

```
/usr/local/tomcat_8888/apache-tomcat-8.5.20/conf/server.xml
```

需要修改三个地方，分别如下：

> 一个小技巧：vim可以用:set number显示行号

修改前：

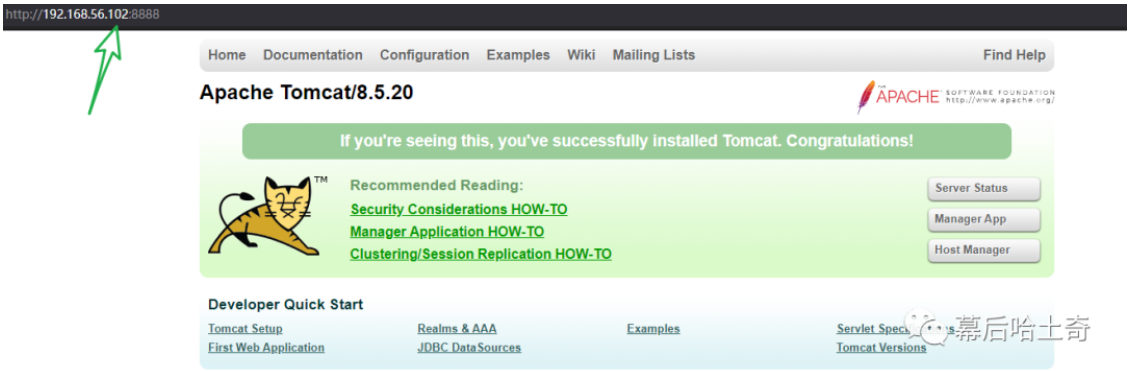
```
18 <!-- Note:  A "Server" is not itself a "Container", so you may not
19      define subcomponents such as "Valves" at this level.
20      Documentation at /docs/config/server.html
21  -->
22 <Server port="8005" shutdown="SHUTDOWN">
23   <Listener className="org.apache.catalina.startup.VersionLoggerListener" />
24   <!-- Security listener. Documentation at /docs/config/listeners.html
25   <Listener className="org.apache.catalina.security.SecurityListener" />
26   -->
27   <!--APR library loader. Documentation at /docs/apr.html -->
28   <Listener className="org.apache.catalina.core.AprLifecycleListener" SSLEngine="on" />
29   <!-- Prevent memory leaks due to use of particular java/javax APIs-->
30   <Listener className="org.apache.catalina.core.JreMemoryLeakPreventionListener" />
31   <Listener className="org.apache.catalina.mbeans.GlobalResourcesLifecycleListener" />
32   <Listener className="org.apache.catalina.core.ThreadLocalLeakPreventionListener" />
62     <!-- A "Connector" represents an endpoint by which requests are received
63          and responses are returned. Documentation at :
64          Java HTTP Connector: /docs/config/http.html
65          Java AJP  Connector: /docs/config/ajp.html
66          APR (HTTP/AJP) Connector: /docs/apr.html
67          Define a non-SSL/TLS HTTP/1.1 Connector on port 8080
68     -->
69     <Connector port="8080" protocol="HTTP/1.1"
70                connectionTimeout="20000"
71                redirectPort="8443" />
111         </SSLHostConfig>
112     </Connector>
113     -->
114
115     <!-- Define an AJP 1.3 Connector on port 8009 -->
116     <Connector port="8009" protocol="AJP/1.3" redirectPort="8443" />
117
118
```


修改后：

```
18 <!-- Note:  A "Server" is not itself a "Container", so you may not
19      define subcomponents such as "Valves" at this level.
20      Documentation at /docs/config/server.html
21  -->
22 <Server port="8015" shutdown="SHUTDOWN">
23   <Listener className="org.apache.catalina.startup.VersionLoggerListener" />
24   <!-- Security listener. Documentation at /docs/config/listeners.html
25   <Listener className="org.apache.catalina.security.SecurityListener" />
26   -->
27   <!--APR library loader. Documentation at /docs/apr.html -->
28   <Listener className="org.apache.catalina.core.AprLifecycleListener" SSLEngine="on" />
29   <!-- Prevent memory leaks due to use of particular java/javax APIs-->
30   <Listener className="org.apache.catalina.core.JreMemoryLeakPreventionListener" />
31   <Listener className="org.apache.catalina.mbeans.GlobalResourcesLifecycleListener" />
32   <Listener className="org.apache.catalina.core.ThreadLocalLeakPreventionListener" />
62     <!-- A "Connector" represents an endpoint by which requests are received
63          and responses are returned. Documentation at :
64          Java HTTP Connector: /docs/config/http.html
65          Java AJP  Connector: /docs/config/ajp.html
66          APR (HTTP/AJP) Connector: /docs/apr.html
67          Define a non-SSL/TLS HTTP/1.1 Connector on port 8080
68     -->
69     <Connector port="8888" protocol="HTTP/1.1"
70                connectionTimeout="20000"
71                redirectPort="8443" />
72     <!-- A "Connector" using the shared thread pool-->
73     <!--
74     <Connector executor="tomcatThreadPool"
75                port="8080" protocol="HTTP/1.1"
76                connectionTimeout="20000"
77                redirectPort="8443" />
78     -->
114
115     <!-- Define an AJP 1.3 Connector on port 8009 -->
116     <Connector port="8019" protocol="AJP/1.3" redirectPort="8443" />
117
```

修改保存后，执行`./shutdown.sh`，再重新执行`./startup.sh`启动服务，再次打开页面，需要输入：

```
http://192.168.56.102:8888
```



不大放心，在服务器上执行如下命令查看端口占用情况：

```
netstat -ntlp
```

```
[root@localhost bin]# netstat -ntlp
Active Internet connections (only servers)
Proto Recv-Q Send-Q Local Address           Foreign Address         State       PID/Program na
me
tcp        0      0 0.0.0.0:22              0.0.0.0:*               LISTEN      1063/sshd

tcp        0      0 127.0.0.1:25            0.0.0.0:*               LISTEN      1276/master

tcp6       0      0 :::8019                 :::*                    LISTEN      6863/java

tcp6       0      0 :::22                   :::*                    LISTEN      1063/sshd

tcp6       0      0 :::8888                 :::*                    LISTEN      6863/java

tcp6       0      0 ::1:25                  :::*                    LISTEN      1276/master

[root@localhost bin]#
```


完全正常，大功告成。

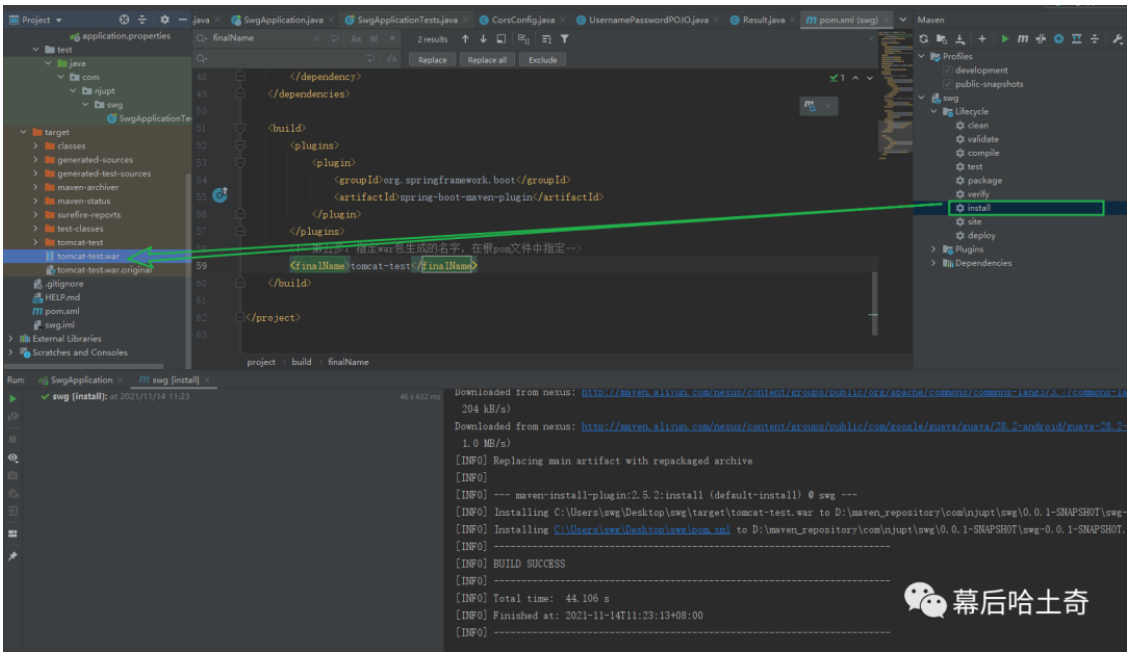
为了简单且能达到目的，本demo后端未使用数据库，因此软件安装部分已经全部完成，剩下的就是编码过程和代码发布过程了。

三、后端代码发布

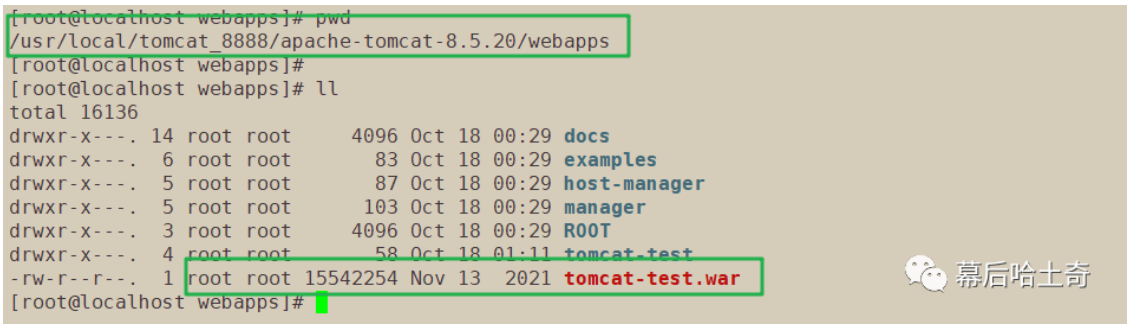
本demo虽小，五脏俱全，使用的是前后端分离项目，不过前端项目直接在本电脑上运行，远程连接服务器，因此我们只需要将服务端代码发布至tomcat下即可。

代码不是我们这个系列关心的问题，文末我会贴出代码仓库，只需要知道如何启动和发布的即可。

首先说下后端代码，基于springboot快速搭建一个服务端，并且将其打为war包（纯粹是为了不让上面tomcat安装白费，实际上默认的jar包即可直接发布启动）



生成的war包直接上传到tomcat的`/usr/local/tomcat_8888/apache-tomcat-8.5.20/webapps`目录下，重新启动tomcat即可，tomcat会自动解压war包并提供web服务。



如何测试接口是否正常呢？我预留了一个测试接口，只需要在浏览器中访问：

```
http://192.168.56.102:8888/tomcat-test/demo/test
```

返回测试成功字样，则说明后端工程已发布完成。

四、前端工程本地启动

主要包含一个登录页面、注册页面以及登录成功后的跳转页面。

来到前端代码目录，打开命令行，首先进行依赖安装：

```
npm install --save vue-resource
```

如果还没有npm，则需要安装下，方法是可以直接安装nodejs，即可安装好npm。

然后编译：

```
npm run build
```

最后运行：

```
npm run dev
```

代码会自动打开浏览器页面 `http://localhost:8080`，显示出来登录页面。

那么工程启动就没问题了，已经完成了95%，剩下的就是修改下前端请求的代码路径，与我们实际部署的代码相匹配即可。

我们在 `前台vue代码\src\views\login` 目录下打开文件 `login.vue`，修改其中请求登录和注册路径：

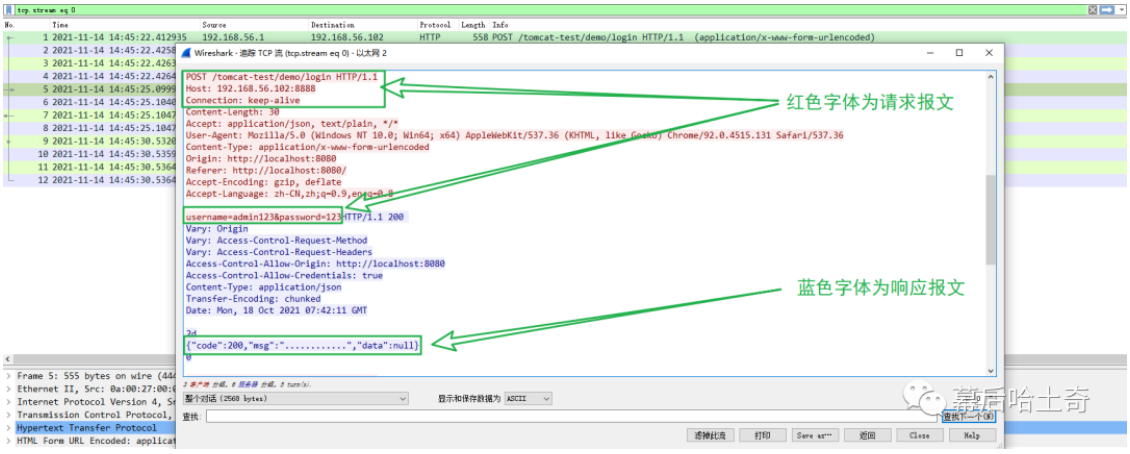
```
        }
    },
    methods: {
        login(){
            if(this.username == "" || this.password == ""){
                alert("请输入用户名或密码")
            }else{
                let data = {'username':this.username,'password':this.password}

                this.$http.post('http://192.168.56.102:8888/tomcat-test/demo/login',data).then((res)=>{
                    console.log(res.data)
                    if(res.data.code == 200){
                        this.tishi = "登录成功"
                        this.showTishi = true
                        setCookie('username',this.username,1000*60)
                        setTimeout(function(){
                            this.$router.push({path:'home',query:{id:1}})
                        }.bind(this),1000)
                    }else{
                        this.tishi = res.data.msg
                        this.showTishi = true
                    }
                })
            }
        },
        ToRegister(){
            this.showRegister = true
            this.showLogin = false
        },
        ToLogin(){
            this.showRegister = false
            this.showLogin = true
        },
        register(){
            if(this.newUsername == "" || this.newPassword == ""){
                alert("请输入用户名或密码")
            }else{
                let data = {'username':this.newUsername,'password':this.newPassword}
                this.$http.post('http://192.168.56.102:8888/tomcat-test/demo/register',data).then((res)=>{
                    console.log(res)
                    if(res.data.code == 200){
                        this.tishi = "恭喜你，注册成功"
                        this.showTishi = true
                        this.username = ''
                        this.password = ''
                        setTimeout(function(){
                            this.showRegister = false
                            this.showLogin = true
                            this.showTishi = false
```


启动前端工程，进行验证，就会得到文章一开始动图的效果。

五、抓包进行简单的报文分析

启动wireshark对登录流程抓包，可以得到如下：



红色部分为HTTP请求报文，蓝色部分为HTTP响应报文，即通过HTTP协议实现的报文请求和响应，HTTP也是我们平时请求任何网站的基本协议，HTTPS也是基于HTTP进行了加密，比如我们这种用户名密码信息不能在网络中明文传输。

具体的，我们后面还会细说，后面一些分析都可以基于我们自己搭建的一套环境进行，并且也更加能直观感受我们平时访问的一些web服务背后的实现机制。

代码仓库地址： **https://github.com/sunweiguo/tomcat-test**

PS：代码仅为简单演示使用，请忽略写法细节。