Tomcat

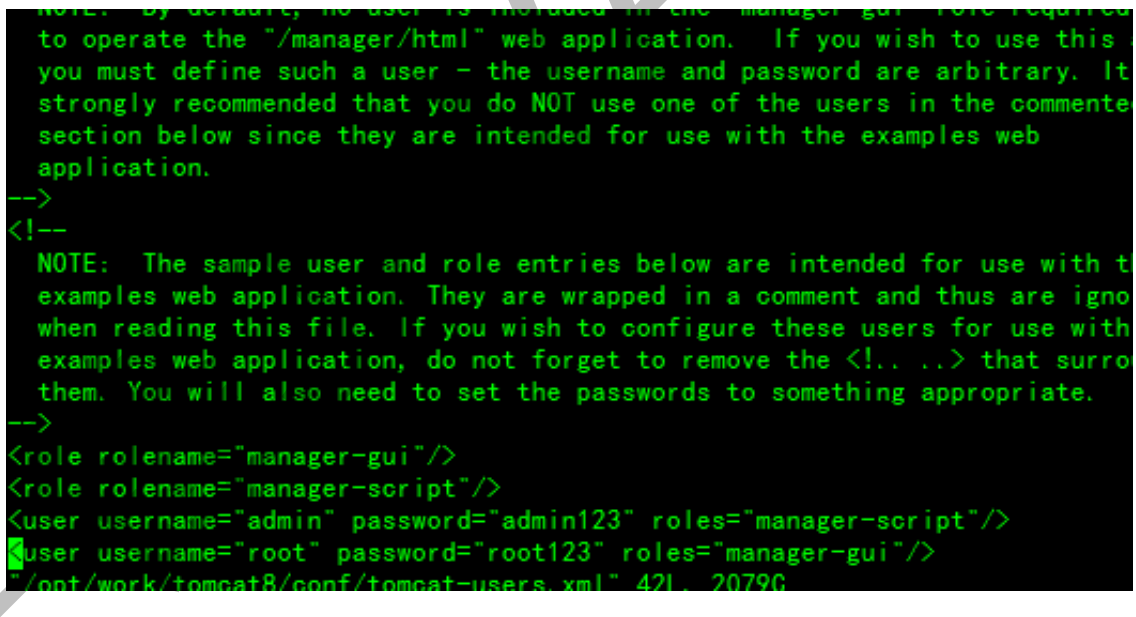
第一节 Maven发布war到远程Tomcat

1.1 配置tomcat的manager

编辑远程tomcat服务器下的conf/tomcat-users.xml,在末尾增加（其实只要拉到文件末尾，去掉注释改一下就可以了）
命令：
vim /opt/work/tomcat8/conf/tomcat-users.xml
添加如下内容：

```
<role rolename="manager-gui"/>
<role rolename="manager-script"/>
<user username="admin" password="admi123" roles="manager-script"/>
<user username="root" password="root123" roles="manager-gui"/>
```

将上面的password改为自己的密码，注意对于tomcat9来说，不能同时赋予用户manager-script和manager-gui角色。
保存tomcat-users.xml。



在tomcat服务器的conf/Catalina/localhost/目录下创建一个manager.xml文件

命令：

```
vim /opt/work/tomcat8/conf/Catalina/localhost/manager.xml
```

写入如下值：

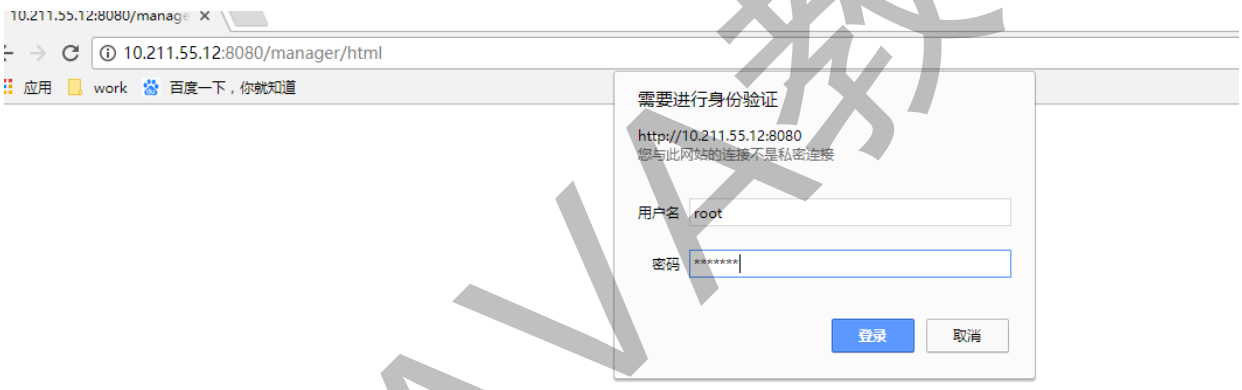
```
<?xml version="1.0" encoding="UTF-8"?>
<Context privileged="true" antiResourceLocking="false"
docBase="${catalina.home}/webapps/manager">
      <Valve className="org.apache.catalina.valves.RemoteAddrValve" allow="^.*$"
/>
</Context>
```

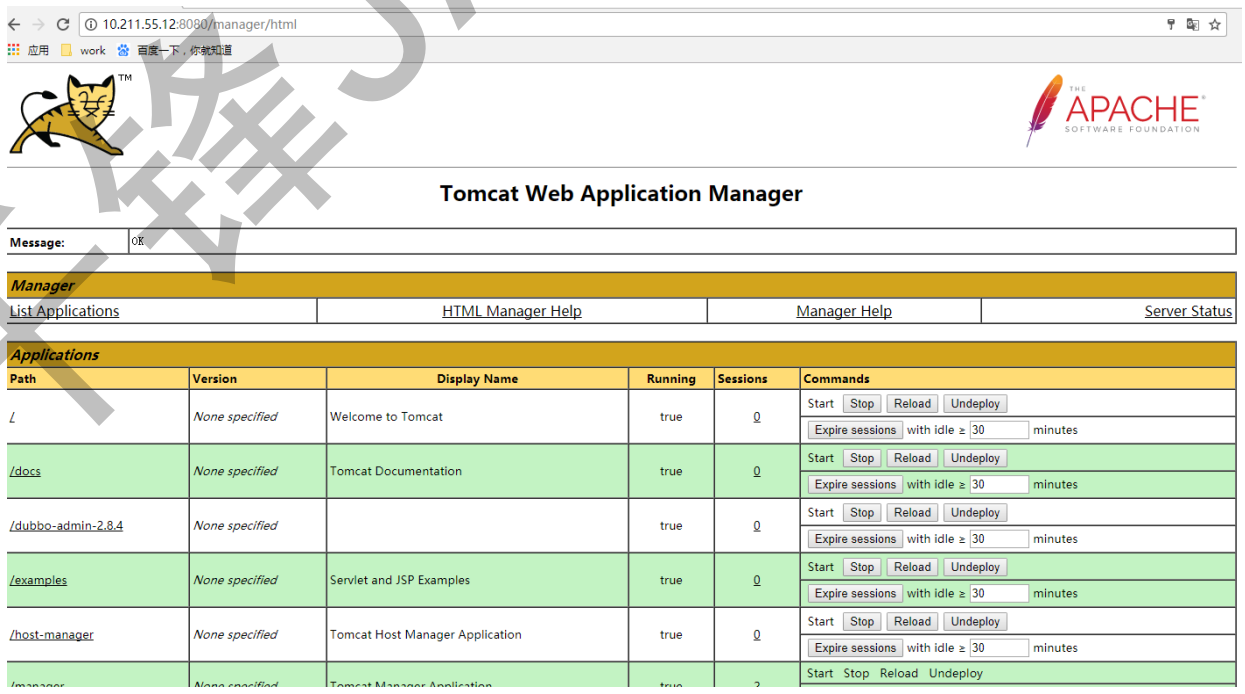
命令：

```
/opt/work/tomcat8/bin/startup.sh    启动
```

浏览器输入地址：

```
http://10.211.55.12:8080/manager/html
```





1.2 创建Maven项目发布

Maven+Idea

pom.xml

```
<?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
http://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>

    <groupId>xph</groupId>
    <artifactId>My_Web</artifactId>
    <version>1.0-SNAPSHOT</version>
    <packaging>war</packaging>

    <build>
        <plugins>
            <plugin>
                <groupId>org.apache.tomcat.maven</groupId>
                <artifactId>tomcat7-maven-plugin</artifactId>
                <version>2.2</version>
                <configuration>
                  <!--远程Tomcat的ip地址和端口号，注意远程Tomcat需要启动-->
                    <url>http://10.211.55.12:8080/manager/text</url>
                    <username>admin</username>
                    <password>admin123</password>
                    <update>true</update>
                    <path>/My_Web</path>
                </configuration>
            </plugin>
        </plugins>
    </build>
</project>
```

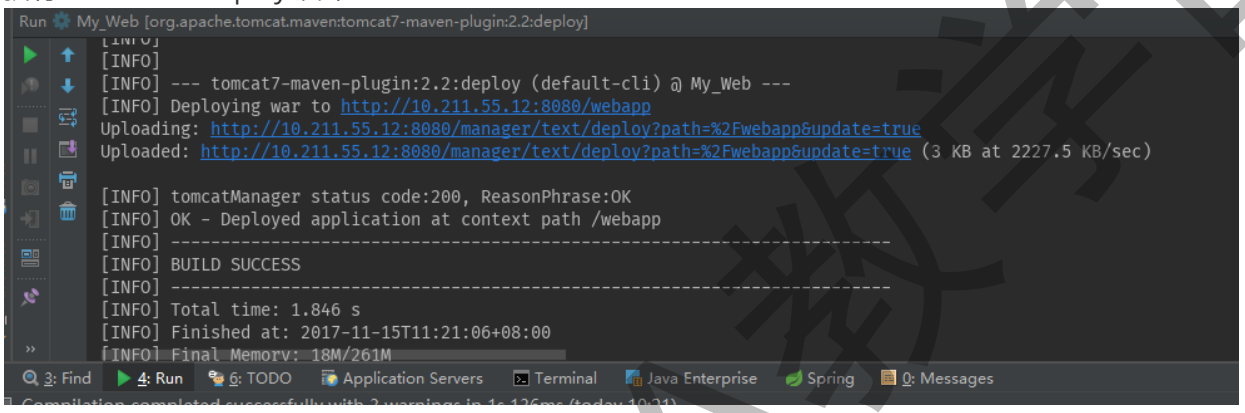
web.xml

```
<welcome-file-list>
    <welcome-file>index.html</welcome-file>
</welcome-file-list>
```

index.html

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Maven发布war到远程Tomcat</title>
</head>
<body>
<h1>Hello,Word!</h1>
</body>
</html>
```

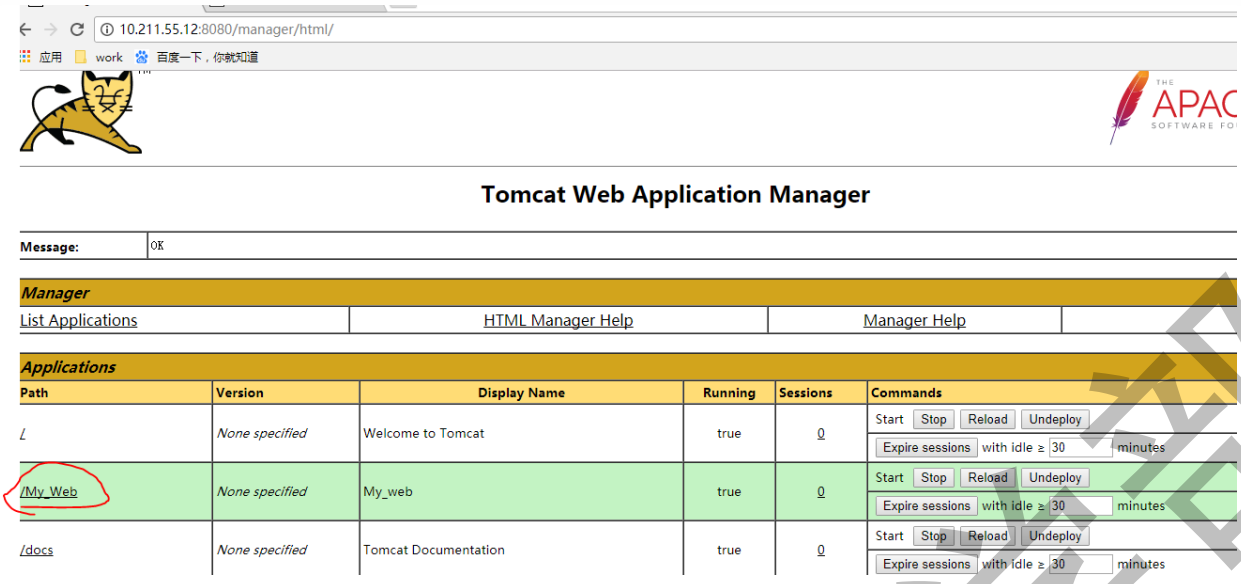
执行：tomcat7:deploy 发布



浏览器访问：

http://10.211.55.12:8080/My_Web/

再看http://10.211.55.12:8080/manager/html，我们可以发现刚刚发布的项目

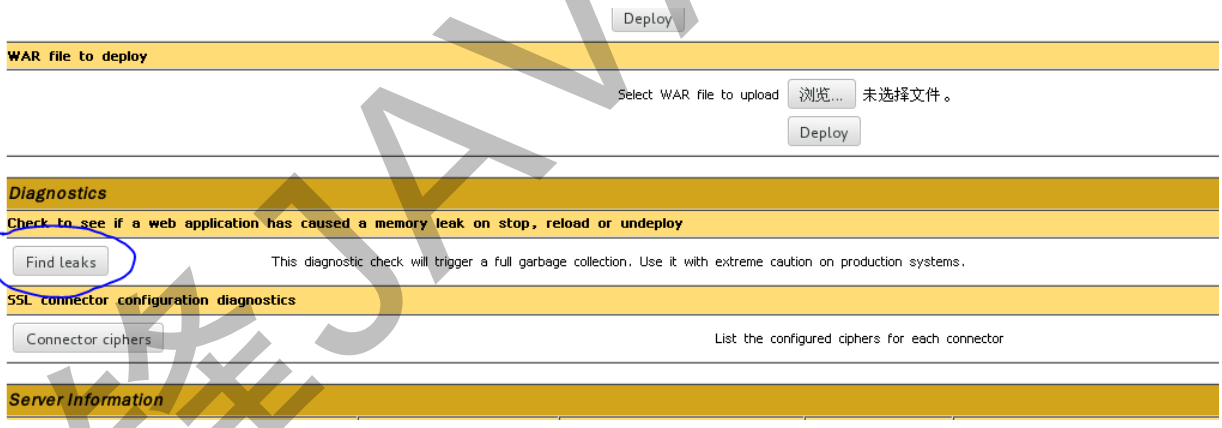


1.3 内存泄漏

不是一定会出现内存泄漏的，如果出现按照下面方案进行解决，如果没有出现就无需解决

1.3.1 查看是否出现内存泄漏

使用上面的方法进行部署后可能会出现严重的内存泄漏现象。tomcat的manager提供了诊断在部署时是否产生内存泄漏的功能，在上面提到的http://serverip:port/manager/html这个页面底部有一个“Find leaks”的按钮，



点击按钮，网页头部出现如下信息说明在部署的时候有内存泄漏

Tom

Message:	The following web applications were stopped (reloaded, undeployed), but their classes from previous runs are still loaded in memory, thus causing a memory leak (use a profiler to confirm): /test

Manager

List Applications | HTML Manage

1.3.2 解决方案

第一步：在项目的pom文件中添加maven依赖

```
<dependency>
          <groupId>se.jiderhamn.classloader-leak-prevention</groupId>
          <artifactId>classloader-leak-prevention-servlet</artifactId>
          <version>2.1.0</version>
</dependency>
```

第二步：web.xml中添加一个Listener

在项目的web.xml中添加一个Listener（必须让此Listener成为web.xml中的第一个Listener，否则不起作用）

```
<listener>
      <listener-
class>se.jiderhamn.classloader.leak.prevention.ClassLoaderLeakPreventorListener</l
istener-class>
</listener>
```

这样部署时的内存泄漏就解决了。
注意：
1、添加这个Listener后，默认在tomcat关闭5s后jvm会进行内存回收的操作，所以在关闭后的5s内，再次启动tomcat，可能会存在问题，导致启动无效（如果出现tomcat重启后日志显示正常但是服务器不工作的话考虑一下是不是这个问题）。
2、这个Listener只解决部署的内存泄漏，其他问题（如jdbc等）产生的内存泄漏还需要自己解决。

第二节 Tomcat支持Https

2.1 什么是HTTPS

在说HTTPS之前先说说什么是HTTP，HTTP就是我们平时浏览网页时候使用的一种协议。HTTP协议传输的数据都是未加密的，也就是明文的，因此使用HTTP协议传输隐私信息非常不安全。

为了保证这些隐私数据能加密传输，于是网景公司设计了SSL（Secure Sockets Layer）协议用于对HTTP协议传输的数据进行加密，从而就诞生了HTTPS。

SSL目前的版本是3.0，被IETF（Internet Engineering Task Force）定义在RFC 6101中，之后IETF对SSL 3.0进行了升级，于是出现了TLS（Transport Layer Security） 1.0，定义在RFC 2246。

实际上我们现在的HTTPS都是用的TLS协议，但是由于SSL出现的时间比较早，并且依旧被现在浏览器所支持，因此SSL依然是HTTPS的代名词，
但无论是TLS还是SSL都是上个世纪的事情，SSL最后一个版本是3.0，今后TLS将会继承SSL优良血统继续为我们进行加密服务。
目前TLS的版本是1.2，定义在RFC 5246中，暂时还没有被广泛的使用。

2.2 HTTPS的工作原理

HTTPS在传输数据之前需要客户端（浏览器）与服务端（网站）之间进行一次握手，在握手过程中将确立双方加密传输数据的密码信息。

TLS/SSL协议不仅仅是一套加密传输的协议，更是一件经过艺术家精心设计的艺术品，TLS/SSL中使用了非对称加密，对称加密以及HASH算法。

握手过程的简单描述如下：

1、浏览器将自己支持的一套加密规则发送给网站

2、网站从中选出一组加密算法与HASH算法，并将自己的身份信息以证书的形式发回给浏览器。证书里面包含了网站地址，加密公钥，以及证书的颁发机构等信息。

3、获得网站证书之后浏览器要做以下工作：
a）验证证书的合法性（颁发证书的机构是否合法，证书中包含的网站地址是否与正在访问的地址一致等），如果证书受信任，则浏览器栏里面会显示一个小锁头，否则会给出证书不受信的提示。
b）如果证书受信任，或者是用户接受了不受信的证书，浏览器会生成一串随机数的密码，并用证书中提供的公钥加密。
c）使用约定好的HASH计算握手消息，并使用生成的随机数对消息进行加密，最后将之前生成的所有信息发送给网站。

4、网站接收浏览器发来的数据之后要做以下的操作：
a）使用自己的私钥将信息解密取出密码，使用密码解密浏览器发来的握手消息，并验证HASH是否与浏览器发来的一致。
b）使用密码加密一段握手消息，发送给浏览器。

5、浏览器解密并计算握手消息的HASH，如果与服务端发来的HASH一致，此时握手过程结束，之后所有的通信数据将由之前浏览器生成的随机密码并利用对称加密算法进行加密

这里浏览器与网站互相发送加密的握手消息并验证，目的是为了保证双方都获得了一致的密码，并且可以正常的加密解密数据，为后续真正数据的传输做一次测试。

2.3 加密

HTTPS一般使用的加密与HASH算法如下：
非对称加密算法：RSA，DSA/DSS
对称加密算法：AES，RC4，3DES
HASH算法：MD5，SHA1，SHA256

其中非对称加密算法用于在握手过程中加密生成的密码，对称加密算法用于对真正传输的数据进行加密，而HASH算法用于验证数据的完整性。

由于浏览器生成的密码是整个数据加密的关键，因此在传输的时候使用了非对称加密算法对其加密。非对称加密算法会生成公钥和私钥，公钥只能用于加密数据，因此可以随意传输，而网站的私钥用于对数据进行解密，所以网站都会非常小心的保管自己的私钥，防止泄漏。

TLS握手过程中如果有任何错误，都会使加密连接断开，从而阻止了隐私信息的传输。

2.4 tomcat服务器配置https双向认证

Tomcat为Linux系统的，而浏览器为Window,当然也可以选择Window上的Tomcat作为本次的服务器配置，都一样。

2.4.1 为服务器生成证书

命令：

```
keytool -genkey -v -alias tomcat -keyalg RSA -keystore /opt/work/tomcat.keystore -validity 36500
```

参数简要说明：
/opt/work/tomcat.keystore含义 是将证书文件的保存路径，证书文件名称是tomcat.keystore
-validity 36500含义 是证书有效期，36500表示100年，默认值是90天 “tomcat”为自定义证书名称

在命令行填写必要参数：
A、 输入keystore密码：此处需要输入大于6个字符的字符串。

B、 “您的名字与姓氏是什么？”这是必填项，并且必须是TOMCAT部署主机的域名或者IP[如：gbcom.com 或者 10.1.25.251]（就是你将来要在浏览器中输入的访问地址），否则浏览器会弹出警告窗口，提示用户证书与所在域不匹配。在本地做开发测试时，应填入“localhost”。

C、 你的组织单位名称是什么？”、“您的组织名称是什么？”、“您所在城市或区域名称是什么？”、“您所在的州或者省份名称是什么？”、“该单位的两字母国家代码是什么？”可以按照需要填写也可以不填写直接回车，在系统询问“正确吗？”时，对照输入信息，如果符合要求则使用键盘输入字母“y”，否则输入“n”重新填写上面的信息。

D、 输入<tomcat>的主密码，这项较为重要，会在tomcat配置文件中使用，建议输入与keystore的密码一致，设置其它密码也可以，完成上述输入后，直接回车则在你在第二步中定义的位置找到生成的文件。

```
[root@CentOS6 work]# keytool -genkey -v -alias tomcat -keyalg RSA -keystore /opt/work/tomcat.keystore -va
y 36500
Enter keystore password:
Re-enter new password:
What is your first and last name?
  [Unknown]:  10.211.55.12
What is the name of your organizational unit?
  [Unknown]:
What is the name of your organization?
  [Unknown]:
What is the name of your City or Locality?
  [Unknown]:
What is the name of your State or Province?
  [Unknown]:
What is the two-letter country code for this unit?
  [Unknown]:
Is CN=10.211.55.12, OU=Unknown, O=Unknown, L=Unknown, ST=Unknown, C=Unknown correct?
  [no]:  y

Generating 2,048 bit RSA key pair and self-signed certificate (SHA256withRSA) with a validity of 36,500 d
        for: CN=10.211.55.12, OU=Unknown, O=Unknown, L=Unknown, ST=Unknown, C=Unknown
Enter key password for <tomcat>
        (RETURN if same as keystore password):
Re-enter new password:
[Storing /opt/work/tomcat.keystore]
[root@CentOS6 work]#
```

2.4.2 为客户端生成证书

为浏览器生成证书，以便让服务器来验证它。为了能将证书顺利导入至IE和Firefox，证书格式应该是PKCS12，在windows系统中，使用如下命令生成：
keytool -genkey -v -alias mykey -keyalg RSA -storetype PKCS12 -keystore F:\Resource\mykey.p12

mykey为自定义,对应的证书库存放在F:\Resource\mykey.p12，客户端的CN可以是任意值。双击mykey.p12文件，即可将证书导入至浏览器（客户端）

```
C:\Users\xing>keytool -genkey -v -alias mykey -keyalg RSA -storetype PKCS12 -keystore F:\Resource\mykey.p12
输入密钥库口令:
再次输入新口令:
您的名字与姓氏是什么?
  [Unknown]:  1x
您的组织单位名称是什么?
  [Unknown]:
您的组织名称是什么?
  [Unknown]:
您所在的城市或区域名称是什么?
  [Unknown]:
您所在的省/市/自治区名称是什么?
  [Unknown]:
该单位的双字母国家/地区代码是什么?
  [Unknown]:
CN=1x, OU=Unknown, O=Unknown, L=Unknown, ST=Unknown, C=Unknown是否正确?
  [否]:  y

正在为以下对象生成 2,048 位RSA密钥对和自签名证书 (SHA256withRSA) (有效期为 90 天):
        CN=1x, OU=Unknown, O=Unknown, L=Unknown, ST=Unknown, C=Unknown
[正在存储F:\Resource\mykey.p12]

C:\Users\xing>
```

2.4.3 服务器配置客户端证书

转换PKCS12为CER

由于是双向SSL认证，服务器必须要信任客户端证书，因此，必须把客户端证书添加为服务器的信任认证。由于不能直接将PKCS12格式的证书库导入，必须先把客户端证书导出为一个单独的CER文件，在windows系统中，使用如下命令：

```
keytool -export -alias mykey -keystore F:\Resource\mykey.p12 -storetype PKCS12 -storepass xph123 -rfc -file F:\Resource\mykey.cer
```

将cer文件上传至服务器，然后执行下面的命令

```
keytool -import -v -file /opt/work/mykey.cer -keystore /opt/work/tomcat.keystore
```

查看

通过list命令查看服务器的证书库，可以看到两个证书，一个是服务器证书，一个是受信任的客户端证书：

```
keytool -list -keystore /opt/work/tomcat.keystore (tomcat为你设置服务器端的证书名)。
```

```
[root@CentOS6 work]# keytool -list -keystore /opt/work/tomcat.keystore
Enter keystore password:

Keystore type: JKS
Keystore provider: SUN

Your keystore contains 2 entries

tomcat, Nov 14, 2017, PrivateKeyEntry,
Certificate fingerprint (SHA1): D4:5F:8B:BF:B5:72:8E:BC:B1:19:DB:DC:E4:49:67:CE:B0:A1:79:76
mykey, Nov 14, 2017, trustedCertEntry,
Certificate fingerprint (SHA1): BC:18:02:C8:49:C6:96:C4:9A:5B:51:6D:9F:77:F8:E9:F9:FC:E5:1F
[root@CentOS6 work]#
```

2.4.4 客户端配置服务器证书

由于是双向SSL认证，客户端也要验证服务器证书，因此，必须把服务器证书添加到浏览的“受信任的根证书颁发机构”。由于不能直接将keystore格式的证书库导入，必须先把服务器证书导出为一个单独的CER文件，使用如下命令：
keytool -keystore /opt/work/tomcat.keystore -export -alias tomcat -file /opt/work/tomcat.cer （tomcat为你设置服务器端的证书名）。

通过以上命令，服务器证书就被我们导出到“/opt/work/tomcat.cer”文件了。
通过远程连接工具将生产的cer文件下载到客户端，双击tomcat.cer文件，按照提示安装证书，将证书填入到“受信任的根证书颁发机构”

2.4.5 配置Tomcat服务器

打开Tomcat根目录下的/conf/server.xml，找到Connector port="8443"配置段，
vim /opt/work/tomcat8/conf/server.xml
修改为如下：

```
<Connector port="8443" protocol="org.apache.coyote.http11.Http11NioProtocol"
SSLEnabled="true" maxThreads="150" scheme="https"
secure="true" clientAuth="true" sslProtocol="TLS"
keystoreFile="/opt/work/tomcat.keystore" keystorePass="xph123"
truststoreFile="/opt/work/tomcat.keystore" truststorePass="xph123" />
```

属性说明：

- clientAuth:设置是否双向验证，默认为false，设置为true代表双向验证
- keystoreFile:服务器证书文件路径
- keystorePass:服务器证书密码
- truststoreFile:用来验证客户端证书的根证书，此例中就是服务器证书
- truststorePass:根证书密码

2.4.6 访问

在浏览器中输入:https://10.211.55.12:8443/，会弹出选择客户端证书界面，点击“确定”，会进入tomcat主页，地址栏后会有“锁”图标，表示本次会话已经通过HTTPS双向验证，接下来的会话过程中所传输的信息都已经过SSL信息加密。