智慧小区平台证书认证

参考手册

2018年01月

目录

[1. 引言 3](#_Toc502829203)

[1.1 编写目的 3](#_Toc502829204)

[1.2 背景 3](#_Toc502829205)

[1.3 文档说明 3](#_Toc502829206)

[1.4 版本修订记录 3](#_Toc502829207)

[1.5 相关读者 4](#_Toc502829208)

[1.6 术语定义 4](#_Toc502829209)

[1.7 参考资料 4](#_Toc502829210)

[2. 单向认证 4](#_Toc502829211)

[2.1 创建根证书私钥 4](#_Toc502829212)

[2.2 创建根证书请求文件 5](#_Toc502829213)

[2.3 自签根证书 5](#_Toc502829214)

[2.4 创建服务端key 6](#_Toc502829215)

[2.5 创建服务端请求文件 6](#_Toc502829216)

[2.6 服务端证书 6](#_Toc502829217)

[2.7 配置nginx单向认证 7](#_Toc502829218)

[3. 双向认证 8](#_Toc502829219)

[3.1 创建根证书私钥 8](#_Toc502829220)

[3.2 创建根证书请求文件 8](#_Toc502829221)

[3.3 自签根证书 9](#_Toc502829222)

[3.4 生成P12证书 9](#_Toc502829223)

[3.5 创建服务端key 10](#_Toc502829224)

[3.6 创建服务端请求文件 10](#_Toc502829225)

[3.7 服务端证书 10](#_Toc502829226)

[3.8 创建客户端key 11](#_Toc502829227)

[3.9 创建客户端请求文件 11](#_Toc502829228)

[3.10 客户端证书 12](#_Toc502829229)

[3.11 客户端P12证书 12](#_Toc502829230)

[3.12 配置nginx双向认证 12](#_Toc502829231)

[3.13 通过浏览器访问 13](#_Toc502829232)

[3.14 通过代码访问 19](#_Toc502829233)

[4. SSH免密码登陆 21](#_Toc502829234)

[4.1 配置步骤 21](#_Toc502829235)

# 

# 引言

## 编写目的

本手册的目的是给开发人员和系统工程师在进行代码开发和安全认证配置。认证包括两种方式：在Nginx服务端进行证书配置，实现浏览器与服务端的单向认证；或者在Nginx服务端进行证书配置并发放客户端证书给客户端，实现双向认证。

预期读者：开发人员、系统部署人员、项目经理。

## 背景

作为恒大集团进军互联网社区服务产业的旗舰公司，恒腾网络受公司指派，开展“智慧小区”的研究和建设工作。恒大希望通过“智慧小区”的建设，一方面通过给客户提供差异化的体验，实现对恒大目前产品的增值，另一方面，实现恒大基于智慧小区的服务能力升级。工程将分阶段进行。未来预期本项目成果将应用到恒大集团800家小区，以及其他房地产合作商。

本项目为“恒大智慧小区”工程的第一阶段，即通过快速而集中的开发建设，预计在2018年3月30日前对整体方案实现现场验证。

## 文档说明

## 版本修订记录

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 日期 | 文档版本 | 修订描述 | 文档作者 | 审批人 |
| 2018/01/02 | 0.1 | 初始版本 | 刘军 |  |
| 2018/01/03 | 0.2 | 修订版本 | 刘军 |  |
| 2018/01/04 | 0.3 | 添加单向认证 | 刘军 |  |
| 2018/01/05 | 0.4 | 添加场景描述 | 刘军 |  |
|  |  |  |  |  |

## 相关读者

|  |  |
| --- | --- |
| 读者分类 | 目的 |
|  |  |
|  |  |
|  |  |
|  |  |

## 术语定义

|  |  |
| --- | --- |
| 术语 | 术语解释 |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |

## 参考资料

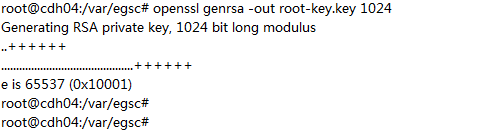
| 名称 | 发表日期 | 出版单位 | 作者 |
| --- | --- | --- | --- |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

# 单向认证

单向认证：只需要验证SSL服务器身份，不需要验证SSL客户端身份。一般Web应用都是采用SSL单向认证的，原因很简单，用户数目广泛，且无需在通讯层对用户身份进行验证，一般都在应用逻辑层来保证用户的合法登入。

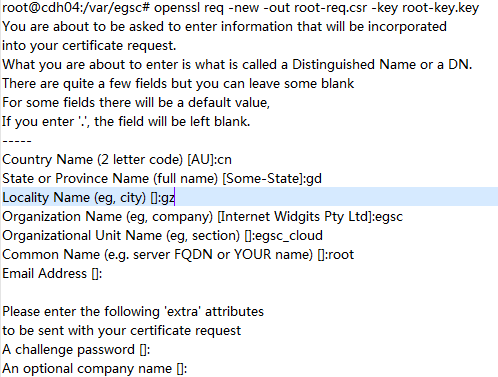
## 创建根证书私钥

命令：openssl genrsa -out root-key.key 2048



## 创建根证书请求文件

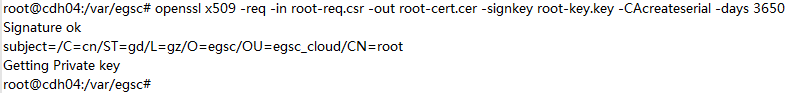
命令：openssl req -new -out root-req.csr -key root-key.key



在创建请求文件的时候会要求输入一些信息，比如国家，省份，城市，组织名等。按照上图所示输入。

## 自签根证书

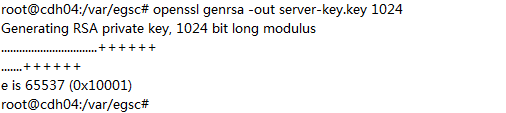
命令：openssl x509 -req -in root-req.csr -out root-cert.cer -signkey root-key.key -CAcreateserial -days 3650



创建一个具有有效期的根证书，时间根据具体情况定。这里作为测试，定为有效期10年。

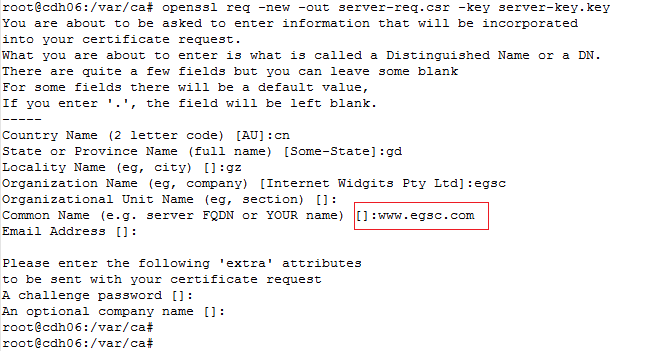
## 创建服务端key

命令：openssl genrsa -out server-key.key 2048



## 创建服务端请求文件

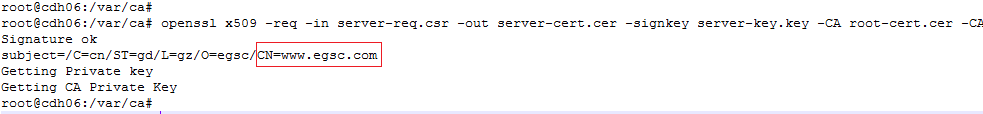
命令：openssl req -new -out server-req.csr -key server-key.key



和创建根证书请求文件的时候一样，这里也会要求输入国家，省份，城市等信息。保持和根证书的一致，特别注意一点，服务器地址填写web server的域名。

## 服务端证书

命令：openssl x509 -req -in server-req.csr -out server-cert.cer -signkey server-key.key -CA root-cert.cer -CAkey root-key.key -CAcreateserial -days 3650



## 配置nginx单向认证

登陆nginx服务器，修改nginx.conf文件：

http {

include mime.types;

default\_type application/octet-stream;

sendfile on;

keepalive\_timeout 65;

server {

listen 443 ssl;

server\_name www.egsc.com;

ssl on;

ssl\_certificate /var/egsc/server-cert.cer; #server证书公钥

ssl\_certificate\_key / var/egsc /server-key.key; #server私钥

location / {

root html;

index index.html index.htm;

}

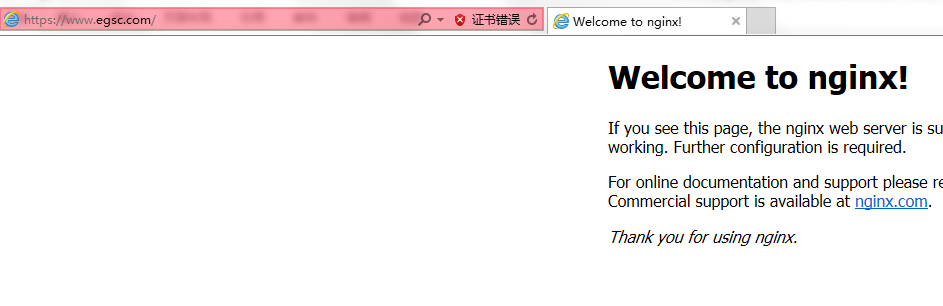
}

}

然后重启nginx。重启完毕后通过浏览器访问nginx，会提示证书存在问题：



点击继续浏览网站：

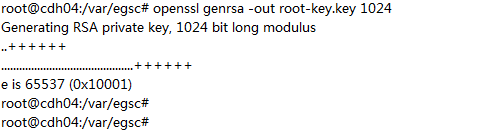


# 双向认证

双向认证：要求服务器和客户端双方都有证书，客户端需要校验服务端，服务端也需要校验客户端。和WEB应用不同，在企业应用对接或者第三方应用对接，可能会要求对客户端(相对而言)做身份验证。这时就需要做SSL双向认证。

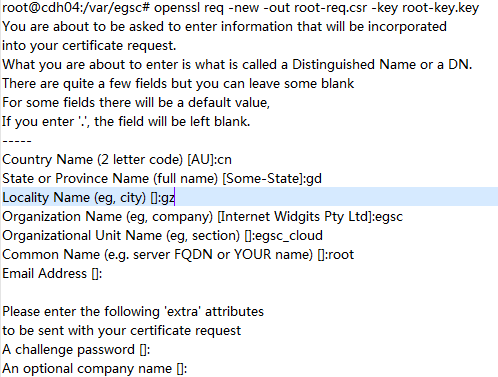
## 创建根证书私钥

命令：openssl genrsa -out root-key.key 2048



## 创建根证书请求文件

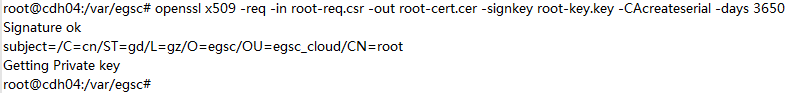
命令：openssl req -new -out root-req.csr -key root-key.key



在创建请求文件的时候会要求输入一些信息，比如国家，省份，城市，组织名等。按照上图所示输入。

## 自签根证书

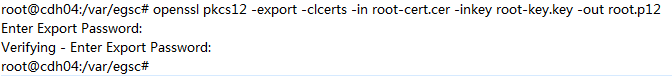
命令：openssl x509 -req -in root-req.csr -out root-cert.cer -signkey root-key.key -CAcreateserial -days 3650



创建一个具有有效期的根证书，时间根据具体情况定。这里作为测试，定为有效期10年。

## 生成P12证书

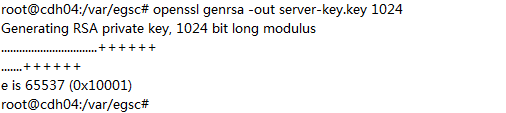
命令：openssl pkcs12 -export -clcerts -in root-cert.cer -inkey root-key.key -out root.p12



这里会要求输入密码，这个密码是用来保护这个p12文件的。

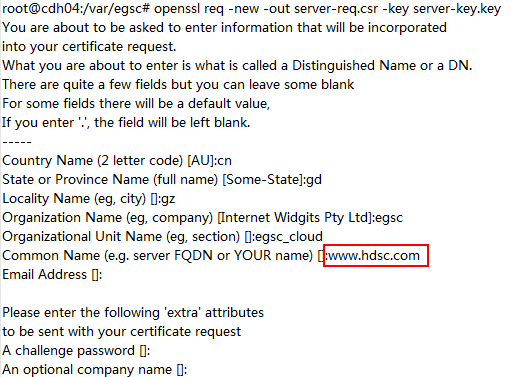
## 创建服务端key

命令：openssl genrsa -out server-key.key 2048



## 创建服务端请求文件

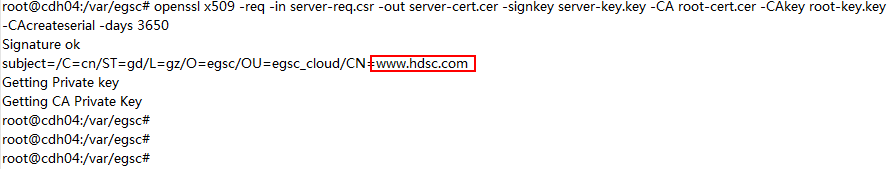
命令：openssl req -new -out server-req.csr -key server-key.key



和创建根证书请求文件的时候一样，这里也会要求输入国家，省份，城市等信息。保持和根证书的一致，特别注意一点，服务器地址填写web server的域名。

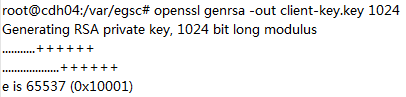
## 服务端证书

命令：openssl x509 -req -in server-req.csr -out server-cert.cer -signkey server-key.key -CA root-cert.cer -CAkey root-key.key -CAcreateserial -days 3650



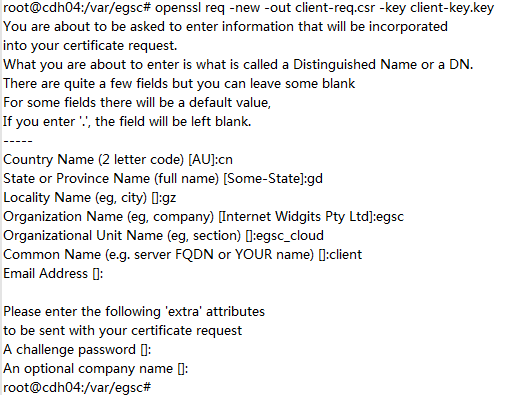
## 创建客户端key

命令：openssl genrsa -out client-key.key 2048



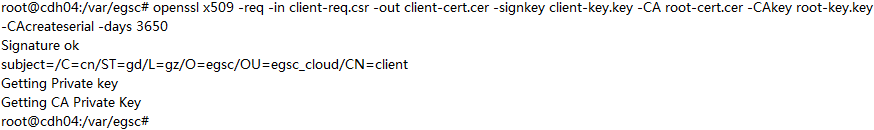
## 创建客户端请求文件

命令：openssl req -new -out client-req.csr -key client-key.key



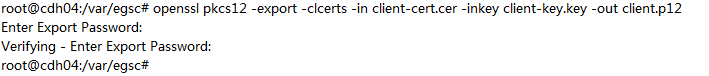
## 客户端证书

命令：openssl x509 -req -in client-req.csr -out client-cert.cer -signkey client-key.key -CA root-cert.cer -CAkey root-key.key -CAcreateserial -days 3650



## 客户端P12证书

命令：openssl pkcs12 -export -clcerts -in client-cert.cer -inkey client-key.key -out client.p12



会要求输入文件保护密码来保护此文件。

## 配置nginx双向认证

登陆nginx服务器，修改nginx.conf文件：

http {

include mime.types;

default\_type application/octet-stream;

sendfile on;

keepalive\_timeout 65;

server {

listen 443 ssl;

server\_name www.hdsc.com;

ssl on;

ssl\_certificate /var/egsc/server-cert.cer; #server证书公钥

ssl\_certificate\_key / var/egsc /server-key.key; #server私钥

ssl\_client\_certificate / var/egsc/root-cert.cer; #根级证书公钥，用于验证各个二级client

ssl\_verify\_client on; #开启客户端证书验证

location / {

root html;

index index.html index.htm;

}

}

}

然后重启nginx。

## 通过浏览器访问

修改客户端hosts文件，添加域名和IP的映射：[www.hdsc.com](http://www.hdsc.com) 映射本机IP地址。

登陆[www.hdsc.com](http://www.hdsc.com)，如下图：

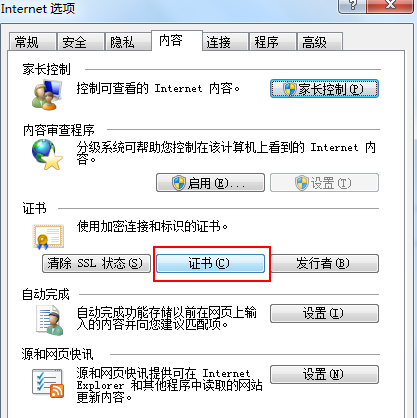


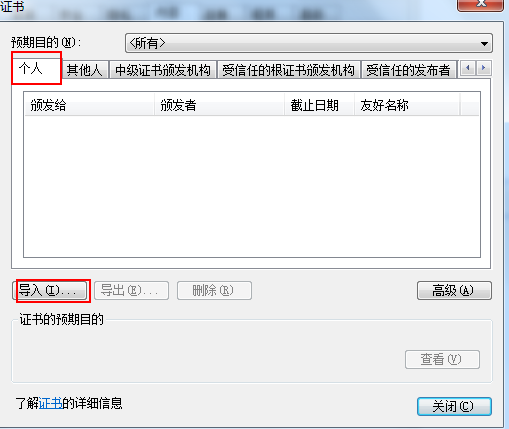
提示证书文件，点击继续浏览网站，会出现错误：



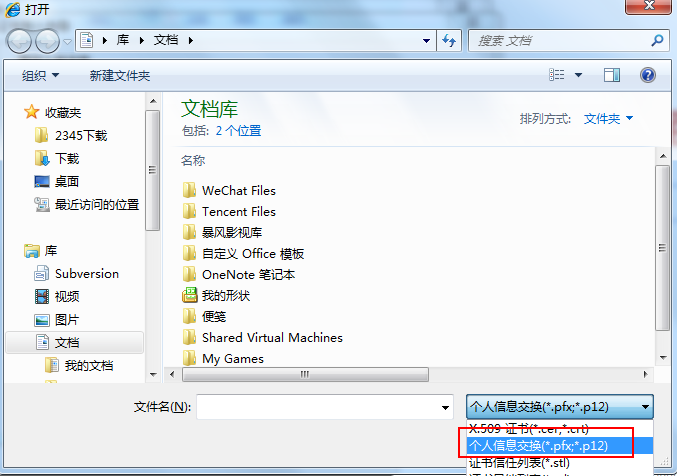
这是因为我们的客户端没有加入客户端证书的原因。

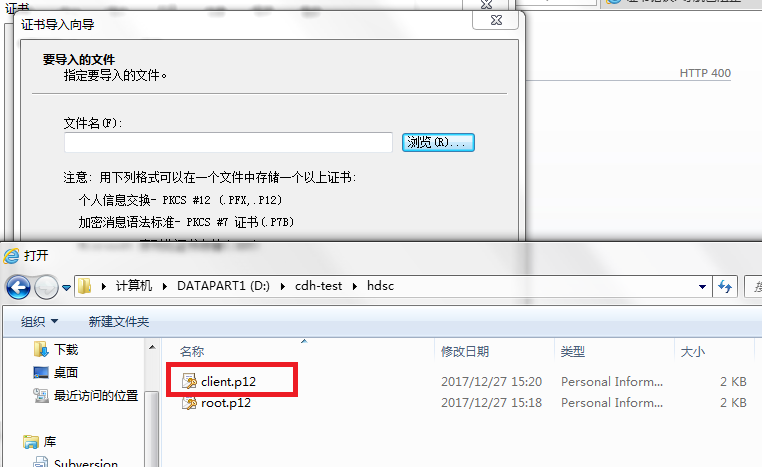
导入client.p12到浏览器：



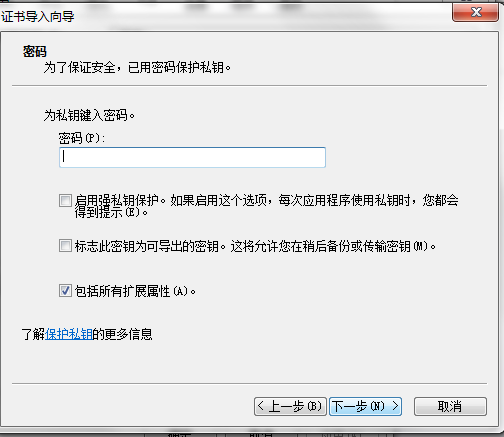


注意选择p12格式的文件

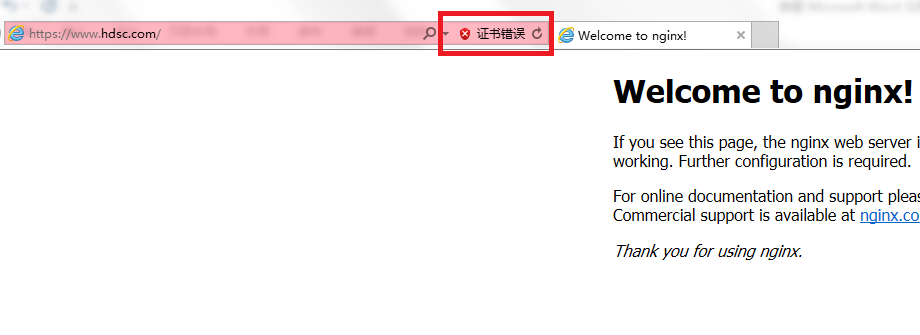




选择文件导入，会要求输入密码，密码是生成client.p12文件的时候输入的密码。



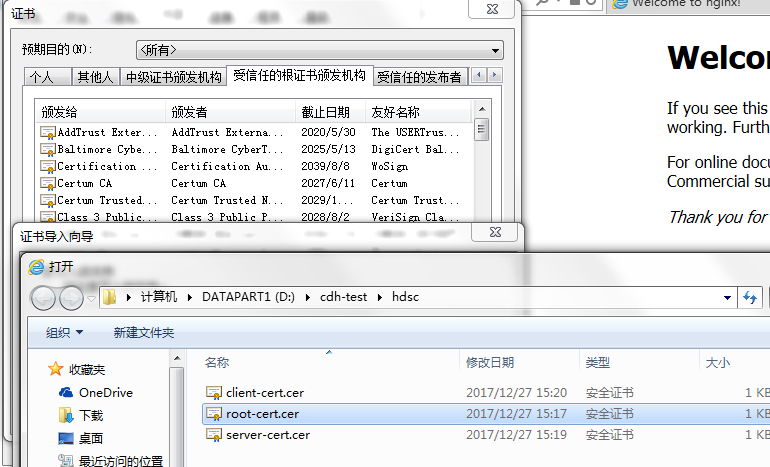
接下来就可以通过浏览器访问了：



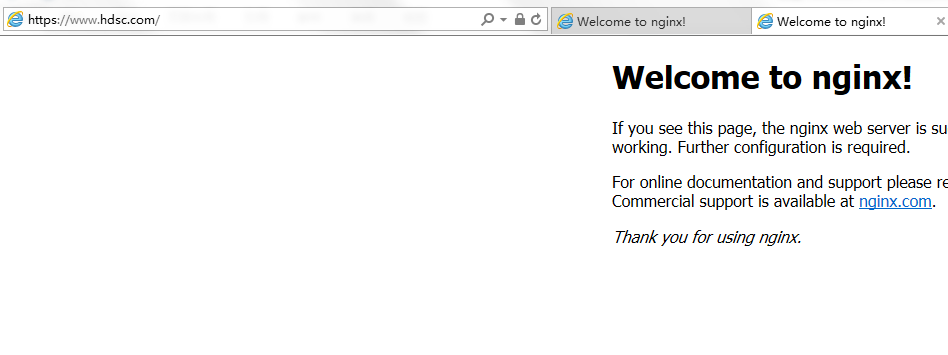
这里会提示证书错误。这是因为这是我们自己签发的证书，是不受浏览器信任的。

再导入根证书到浏览器。 这次需要选择受信任的根证书颁发机构：





导入完毕后，重启浏览器，再打开页面：

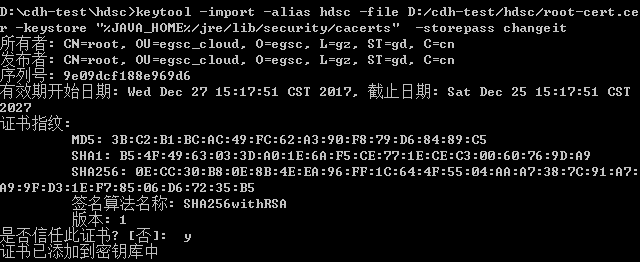


## 通过代码访问

使用代码访问时，要先将root根证书导入到本地的java受信证书库中。

使用命令：

keytool -import -alias hdsc -file D:/test/hdsc/root-cert.cer -keystore "%JAVA\_HOME%/jre/lib/security/cacerts" -storepass changeit



然后可以使用代码访问。

测试代码如下：

**public** **class** HttpsDemo {

**private** **final** **static** String ***PFX\_PATH*** = "d:/egsc/client.p12"; //客户端证书路径

**private** **final** **static** String ***PFX\_PWD*** = "egsc2017"; //客户端证书密码

**public** **static** String sslRequestGet(String url) **throws** Exception {

KeyStore keyStore = KeyStore.*getInstance*("PKCS12");

InputStream instream = **new** FileInputStream(**new** File(***PFX\_PATH***));

**try** {

keyStore.load(instream, ***PFX\_PWD***.toCharArray());

} **finally** {

instream.close();

}

SSLContext sslcontext = SSLContexts.*custom*().loadKeyMaterial(keyStore, ***PFX\_PWD***.toCharArray()).build();

SSLConnectionSocketFactory sslsf = **new** SSLConnectionSocketFactory(sslcontext

, **new** String[] { "TLSv1" } // supportedProtocols ,这里可以按需要设置

, **null** // supportedCipherSuites

, SSLConnectionSocketFactory.*getDefaultHostnameVerifier*());

CloseableHttpClient httpclient = HttpClients.*custom*().setSSLSocketFactory(sslsf).build();

**try** {

HttpGet httpget = **new** HttpGet(url);

httpget.addHeader("egid", "12345678");

httpget.addHeader("token", "ESFGTR64562DFG5663GDH54DFGDS");

CloseableHttpResponse response = httpclient.execute(httpget);

**try** {

HttpEntity entity = response.getEntity();

String jsonStr = EntityUtils.*toString*(response.getEntity(), "UTF-8");//返回结果

EntityUtils.*consume*(entity);

**return** jsonStr;

} **finally** {

response.close();

}

} **finally** {

httpclient.close();

}

}

**public** **static** **void** main(String[] args) **throws** Exception {

System.***out***.println(System.*getProperty*("java.home"));

System.***out***.println(*sslRequestGet*("https://www.hdsc.com/"));

}

}



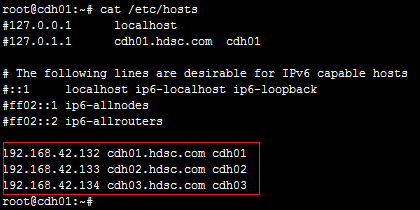
# SSH免密码登陆

配置SSH免密码登陆实现了在不同的服务器之间不输入密码直接ssh登陆。这对于部署在不同服务器之间的同类型服务互相访问是非常方便和必要的。需要SSH免密码登录的场景一般是同一个局域网内的几台服务器上的相同账号有互相访问和通信的需求，这个时候需要设置SSH免密码。最典型的场景是hadoop集群，postgreSQL集群等。

## 配置步骤

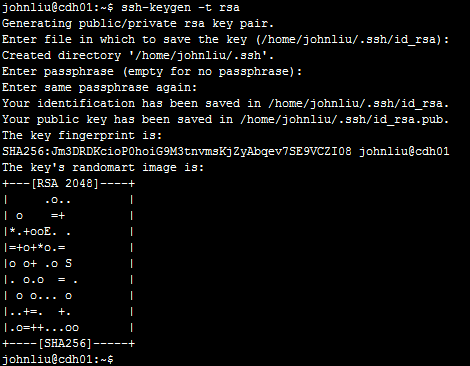
* 修改每个服务器的/etc/hosts文件，配置域名和ip映射。

本次示例共有3台server：cdh01, cdh02, cdh03



* 在要免密码通信的每台server上生成秘钥

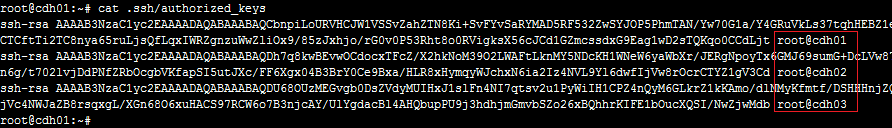
ssh-keygen –t rsa



* 将公钥加入到认证文件

cat id\_rsa.pub > authorized\_keys

* 复制每台server的公钥到一个认证文件中



* 分发认证文件

假如有3台server，cdh01，cdh02，cdh03，将cdh01的认证文件分发给另外两天server。

scp authorized\_keys @cdh02:/root/

scp authorized\_keys @cdh03:/root/

* 免密码登录

现在可以实现在这三台server之间直接免密码ssh登录。

