什么是https

要说https我们得先说SSL(Secure Sockets Layer，安全套接层)，这是一种为网络通信提供安全及数据完整性的一种安全协议，SSL在网络传输层对网络连接进行加密。SSL协议可以分为两层：SSL记录协议（SSL Record Protocol），它建立在可靠的传输协议如TCP之上，为高层协议提供数据封装、压缩、加密等基本功能支持；SSL握手协议（SSL Handshake Protocol），它建立在SSL记录协议之上，用于在实际数据传输开始之前，通信双方进行身份认证、协商加密算法、交换加密密钥等。在Web开发中，我们是通过HTTPS来实现SSL的。HTTPS是以安全为目标的HTTP通道，简单来说就是HTTP的安全版，即在HTTP下加入SSL层，所以说HTTPS的安全基础是SSL，不过这里有一个地方需要小伙伴们注意，就是我们现在市场上使用的都是TLS协议(Transport Layer Security，它来源于SSL)，而不是SSL，只不过由于SSL出现较早并且被各大浏览器支持因此成为了HTTPS的代名词，。你可以把HTTPS和SSL的关系理解成iPhone和富土康的关系，大概就是这样哈。

证书生成

使用SSL需要我们先生成一个证书，这个证书我们可以自己生成，也可以从SSL证书授权中心获得，自己生成的不被客户端认可，从授权中心获得的可以被客户端认可，提供SSL授权证书的服务商有很多，小伙伴们有兴趣可以自行查找，我这里以自己生成的证书为例。

生成方式也很简单，直接使用java自带的命令keytool来生成，生成命令如下：

keytool -genkey -alias tomcat -storetype PKCS12 -keyalg RSA -keysize 2048 -keystore keystore.p12 -validity 3650

这里涉及到几个参数的含义我简单说一下：

1.-storetype 指定密钥仓库类型

2.-keyalg 生证书的算法名称，RSA是一种非对称加密算法

3.-keysize 证书大小

4.-keystore 生成的证书文件的存储路径

5.-validity 证书的有效期

OK，执行完上面一行命令后，在你的系统的当前用户目录下会生成一个keystore.p12文件（如果你修改了证书文件的名称那就是你修改的名字），将这个文件拷贝到我们项目的根目录下，然后修改application.properties文件，添加HTTPS支持。在application.properties中添加如下代码：

server.ssl.key-store=keystore.p12

server.ssl.key-store-password=111111

server.ssl.keyStoreType=PKCS12

server.ssl.keyAlias:tomcat

第一行指定签名文件，第二行指定签名密码，第三行指定密钥仓库类型，第四个是别名。OK，这样配置完成之后我们就可以通过HTTPS来访问我们的Web了

当然，这里我在CustomServletContainer类中将默认端口修改为8443了(这个类如果尚不清楚可以参考[使用Spring Boot开发Web项目](http://blog.csdn.net/u012702547/article/details/53784992)这篇文章)，当然你用其他的端口也行。

HTTP自动转向HTTPS（可选）

光有HTTPS肯定还不够，很多用户可能并不知道，用户有可能继续使用HTTP来访问你的网站，这个时候我们需要添加HTTP自动转向HTTPS的功能，当用户使用HTTP来进行访问的时候自动转为HTTPS的方式。这个配置很简单，在入口类中添加相应的转向Bean就行了，如下：

@Bean

public EmbeddedServletContainerFactory servletContainer() {

TomcatEmbeddedServletContainerFactory tomcat = new TomcatEmbeddedServletContainerFactory() {

@Override

protected void postProcessContext(Context context) {

SecurityConstraint constraint = new SecurityConstraint();

constraint.setUserConstraint("CONFIDENTIAL");

SecurityCollection collection = new SecurityCollection();

collection.addPattern("/\*");

constraint.addCollection(collection);

context.addConstraint(constraint);

}

};

tomcat.addAdditionalTomcatConnectors(httpConnector());

return tomcat;

}

@Bean

public Connector httpConnector() {

Connector connector = new Connector("org.apache.coyote.http11.Http11NioProtocol");

connector.setScheme("http");

//Connector监听的http的端口号

connector.setPort(8080);

connector.setSecure(false);

//监听到http的端口号后转向到的https的端口号

connector.setRedirectPort(8443);

return connector;

}

这个时候当我们访问http://localhost:8080的时候系统会自动重定向到https://localhost:8443这个地址上。这里的Connector实际就是我们刚刚接触jsp时在xml中配置的Tomcat的Connector节点