## TCP/IP协议组

## HTTP

### 后退按钮访问浏览器缓存

https://blog.csdn.net/yangshijin1988/article/details/44418587

当点击后退按钮时，默认情况下浏览器不是从Web服务器上重新获取页面，而是从浏览器缓存中载入页面。

HTTP头信息“**Expires**”和“**Cache-Control**”为应用程序服务器提供了一个控制浏览器和代理服务器上缓存的机制。

HTTP头信息**Expires**告诉代理服务器它的缓存页面何时将过期。   
HTTP1.1规范中新定义的头信息**Cache-Contro**l可以通知浏览器不缓存任何页面。

如下是使用Cache-Control的基本方法：   
 1) no-cache:强制缓存从服务器上获取新的页面   
 2) no-store: 在任何环境下缓存不保存任何页面

例：

对于HTML网页，加入：

<meta HTTP-EQUIV="pragma" CONTENT="no-cache">   
<meta HTTP-EQUIV="Cache-Control" CONTENT="no-cache, must-revalidate">   
<meta HTTP-EQUIV="expires" CONTENT="0">

对于JSP页面，加入：  
<%   
response.setHeader("Cache-Control","no-store");   
response.setHeader("Pragrma","no-cache");   
response.setDateHeader("Expires",0);   
%>

## HTTPS

SSL协议（Secure Sockets Layer [安全套接层](https://baike.baidu.com/item/%E5%AE%89%E5%85%A8%E5%A5%97%E6%8E%A5%E5%B1%82" \t "https://baike.baidu.com/item/ssl/_blank)）

TLS协议（Transport Layer Security 安全传输层协议）

### 证书

尽管有 非对称算法 与 对称算法 对数据进行加密与解密。但为了防止公钥被人恶意劫持、替换，以取代通信双方进行交流，于是需要公证机构（**CA**）颁布一个 **证书** 证明某个 公钥 属于 某个机构。

### 数字签名

同样为防止 证书 被劫持，故需要通过 Hash 算法，将公钥与网站信息通过 Hash 算法单向加密，生成 消息摘要。

由于 Hash 算法只要输入数据有一点变化，那么生成的消息摘要就会发生巨变，防止他人修改原始数据。

公钥及网站信息

CA

消息摘要

Hash 加密

数字签名

CA密钥 加密

公钥及网站信息

数字签名

数字证书

当 浏览器 得到数字证书时，将用 Hash 算法对 公钥及网站信息 进行加密生成**消息摘要1**，再用CA的公钥对数字前面进行解密得到**消息摘要2**，将两者对比以知是否被修改。

注意：CA有也可能被人劫持

在操作系统或浏览器中预置顶层的CA证书链，以互相验证CA。

## FTP（文件传输协议）

FTP与HTTP均为文件传输协议，具有很多共同点。

均运行在TCP上。

FTP协议使用两个并行的TCP连接来传输文件：控制连接 与 数据连接

### 控制连接

用于 客户端 与 服务端 之间传输控制信息，比如用户标识、口令、改变远程目录以及“存放（put）”和“获取（get）”文件的命令。

### 数据连接

TCP控制连接端口21

TCP数据连接端口20

服务端

客户端

传输文件

### 带外传送 与 带内传送

由于FTP使用独立的 控制连接 来传输控制信息，所以我们称FTP是 带外（out-of-band）传送；

HTTP则通过 带内（in-band） 发送控制信息

### 流程

用户主机 与 远程主机 开始一个FTP会话时。

客户端首先在向服务器21端口与服务器发起一个用于控制的 TCP 连接。FTP客户端也通过该控制连接发送用户的标识和口令，发送改变远程目录的命令。

FTP服务器端从该连接上收到一个文件传输的命令后，就发起一个到客户端的TCP数据连接。

FTP在该数据连接上准确地传送一个文件，然后关闭该连接。

故每次传输文件，TCP都是一个新的 数据连接。

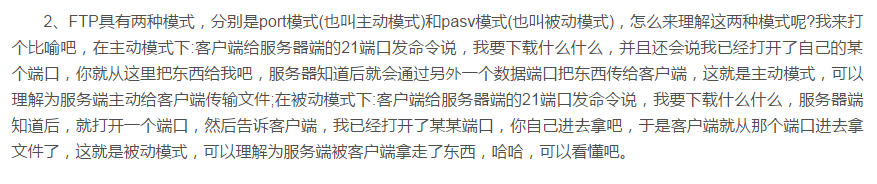
注意，FTP服务器必须在整个会话过程中保持用户的状态。特别是服务器必须把用户账户与控制连接联系起来，随着用户在远程目录树上徘徊，用户必须追踪用户在远程目录上的当前位置。

对每个进行中的用户会话状态信息进行追踪，大大限制了FTP同时维持的会话总数。

20180905

<https://blog.csdn.net/hbhgyu/article/details/80463499>

关于主动模式与被动模式的区别，笔者反而觉得此人讲法最容易让人产生印象。





#### 主动模式（Actitve Mode）

客户端从一个任意的非特权端口N（N > 1024 ）连接到FTP服务器的控制端口，即21端口。然后客户端开始监听端口N + 1到FTP服务器，并发送命令 “PORT N+1”到FTP服务器。接着服务器会从它自己的数据端口（20）连接到客户端指定的数据端口（ N + 1 ）

针对防火墙来说，必须允许以下通讯要求才能主动连接FTP：

1、任何大于1024的端口N 到FTP服务器的21端口。（客户端初始化的连接）

2、FTP服务器的21端口 到大于1024的端口N。 （服务器响应客户端的控制端口）

3、FTP服务器的20端口 到大于1024的端口。（服务器端初始化数据连接到客户端的数据端口）

4、大于1024端口 到FTP服务器的20端口（客户端发送ACK响应到服务器的数据端口）

#### 被动模式（Passive Mode）

 为了解决服务器发起到客户的连接的问题，人们开发了一种不同的FTP连接方式。这就是所谓的被动方式，或者叫做PASV，当客户端通知服务器它处于被动模式时才启用。

   在被动方式FTP中，命令连接和数据连接都由客户端发起，这样就可以解决从服务器到客户端的数据端口的入方向连接被防火墙过滤掉的问题。

   当开启一个 FTP连接时，客户端打开两个任意的非特权本地端口（N 大于 1024和N+1）。第一个端口连接服务器的21端口，但与主动方式的FTP不同，**客户端不会提交PORT命令并允许服务器来回连它的数据端口**，而是提交 PASV命令。这样做的结果是服务器会开启一个任意的非特权端口（P大于 1024），并发送PORT P命令给客户端。然后客户端发起从本地端口N+1到服务器的端口P的连接用来传送数据。    
       对于服务器端的防火墙来说，必须允许下面的通讯才能支持被动方式的FTP:

1. 从任何大于1024的端口N 到服务器的21端口（客户端初始化的连接）    
   2、服务器的21端口 到任何大于1024的端口N（服务器响应到客户端的控制端口的连接）  
   3、从任何大于1024端口N+1 到服务器的大于1024端口X（客户端初始化数据连接到服务器指定的任意端口）

4、服务器的大于1024端口X 到远程的大于1024的端口N+1（服务器发送ACK响应和数据到客户端的数据端口）（X这个端口是个1025--5000的端口）

### Python 服务器代码

<https://www.cnblogs.com/huangxm/p/6274645.html>

from pyftpdlib.authorizers import DummyAuthorizer #傻瓜式授权

from pyftpdlib.handlers import FTPHandler #FTP句柄

from pyftpdlib.servers import FTPServer #FTP服务

print("Hello world")

authorizer = DummyAuthorizer() #新建一个用户组

authorizer.add\_user("fan", "root", "D:/", perm="elr")#将用户名，密码，指定目录，权限 添加到里面

#这个是添加匿名用户,任何人都可以访问，如果去掉的话，需要输入用户名和密码，可以自己尝试

#authorizer.add\_anonymous("D:/")

#初始化FTP句柄

handler = FTPHandler

handler.authorizer = authorizer

#开启服务器

server = FTPServer(("127.0.0.1", 21), handler)

server.serve\_forever()

### Python客户端代码

'''

Created on 2018年9月1日

https://blog.csdn.net/qq\_39091609/article/details/79338906

@author: Administrator

'''

import ftplib

import sys

print("Hello FTP client:")

#获取服务器的ip地址（如192.168.1.107），使用sys.argv可以从命令行参数里面获取

if len(sys.argv) < 2:

#tmp = input("please input server address:")

tmp = "127.0.0.1"

sys.argv.append(tmp)

server\_address = sys.argv[1]

#创建FTP实例，并显示欢迎界面

ftp = ftplib.FTP(server\_address)

print(ftp.getwelcome())

#登录，输入服务器里添加过的用户名和口令

#ftp.login('user', 'pass')

#文件上传

def upload(fname):

fd = open(fname, 'rb')

new\_name = input("input new name:")

#以二进制的形式上传

ftp.storbinary("STOR %s" % new\_name, fd)

fd.close()

print("upload finished")

#文件下载

def download(fname):

#构建文件的存储路径，这里用的是D盘,可以自行设置

new\_path = "D:\\FTPdownload\\" + fname

fd = open(new\_path, 'wb')

#以二进制形式下载，注意第二个参数是fd.write，上传时是fd

ftp.retrbinary("RETR %s" % fname, fd.write)

fd.close()

print("download finished")

def main():

#选择操作，上传、下载、退出

op = input("what do you want?(u/d/q)")

if op == "u":

#输入文件完整路径，必要时可以用绝对路径

fname = input("input the file of path:")

upload(fname)

elif op == "d":

fname = input("input the file name:")

download(fname)

else:

print("quit now!")

ftp.quit()

if \_\_name\_\_ == '\_\_main\_\_':

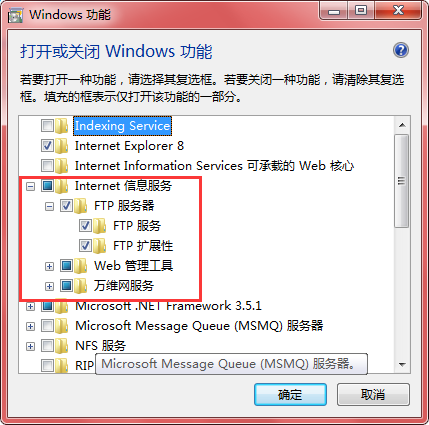
main()

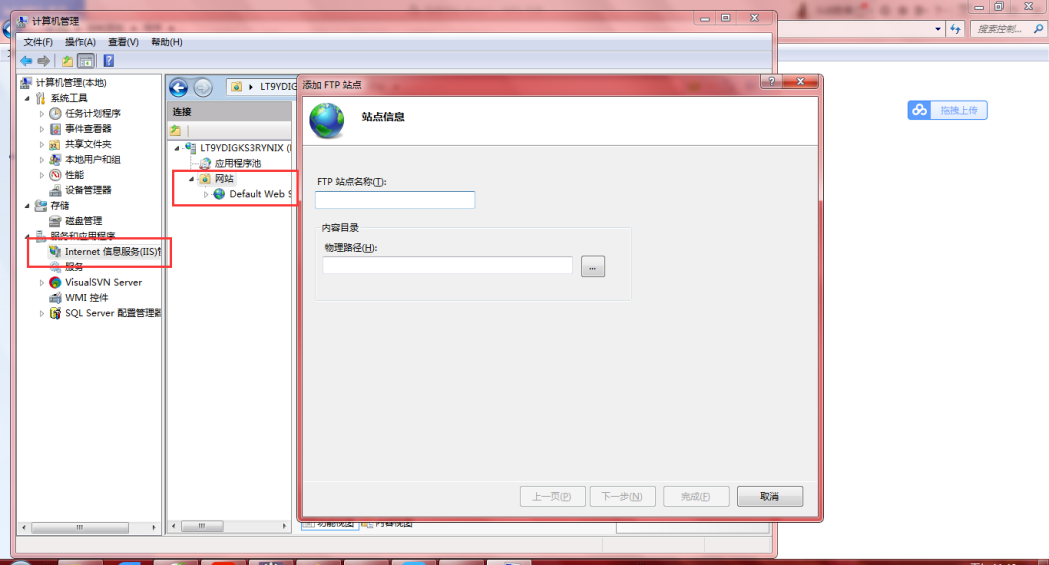
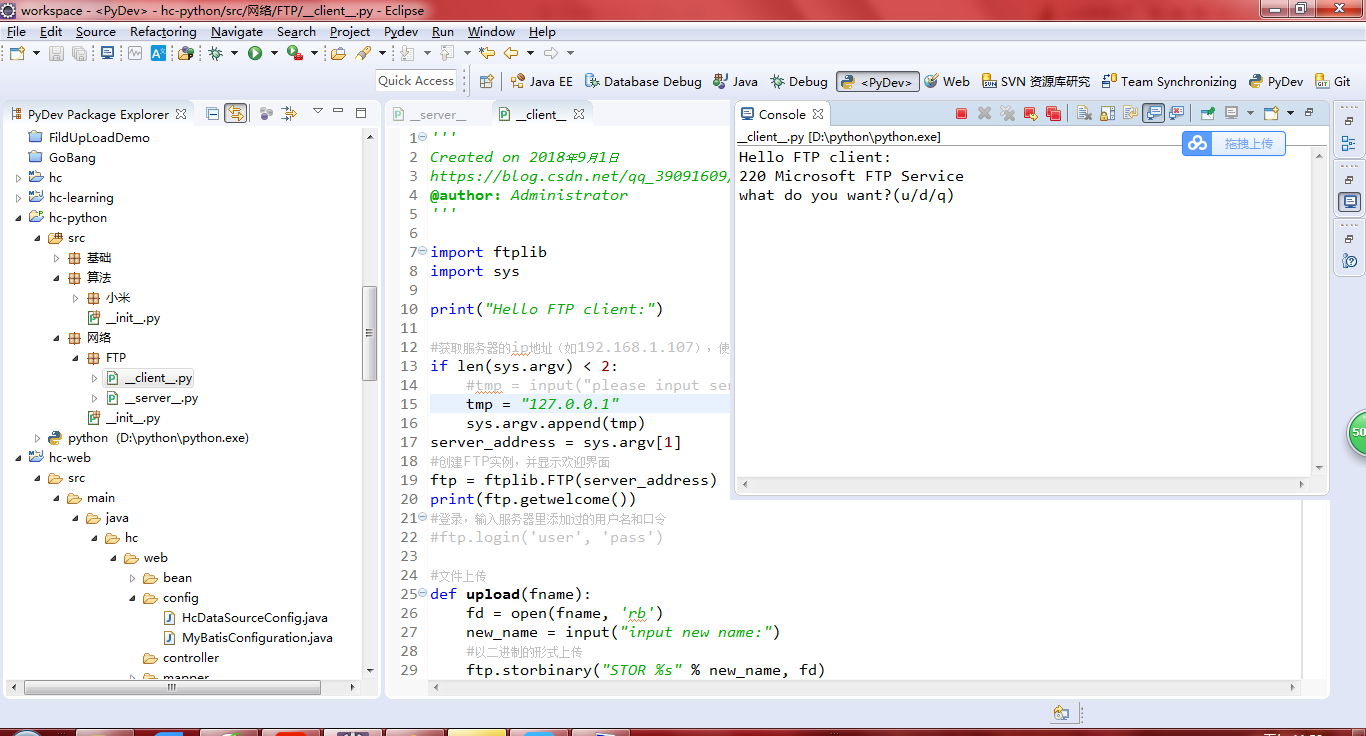
### Windows开启FTP Server服务

笔者使用的是windows7操作系统

https://www.cnblogs.com/liangxuru/p/6148212.html

1、打开控制面板->程序，选择“打开或关闭Windows功能”。

2、

1. 
2. 

# 网络安全

## 防火墙

*只有HTTP协议默认通过防火墙*

防火墙（firewall）是一个硬件和软件的结合体，它将一个机构的内部网络和整个因特网隔离开，允许一些数据分组通过而阻止另一些分组通过。防火墙允许网络管理员控制外部和被管理网络内部资源之间的访问，这种控制是通过管理流入和流出这些资源的流量实现的。

防火墙的三个目标：

1. **从外部到内部和从内部到外部的所有流量都通过防火墙**。许多大型机构可使用多级防火墙或分布式防火墙[ Skoudis2006 ]，但在对该网络的单一接入点处设置一个防火墙，这使得管理和施加安全访问策略更为容易。
2. **仅被授权的流量（由本地安全策略定义）允许通过**。随着进入和离开机构网络的所有流量流经防火墙，该防火墙能够限制对授权流量的访问。
3. **防火墙自身免于渗透**。防火墙自身是一种与网络连接的设备，如果设计或安装不当，将可能危及安全，在这种情况下它仅提供了一种安全的假象。

### 防火墙分类

传统的 **分组过滤器**

一个机构通常都有一个将其内部网络与其ISP相连的网管路由器。所有离开和进入内部网络的流量都要经过这个路由器，而这个路由器正是**分组过滤**出现的地方。

分组过滤器独立地检查每个数据报，然后基于管理员特定的规则决定该数据报应当允许通过还是应当丢弃。过滤决定通过常基于下列因素：

IP源或目的地址

在IP数据报中的协议类型字段：TCP、UDP、ICMP、OSPF等

TCP或UDP的源和目的端口

TCP标志比特：SYN、ACK

ICMP报文类型

数据报离开和进入网络的不同规则

对不同路由器接口的不同规则

20180905

由于书后面的图片笔者无法弄到word上，建议读者自己去动手设置防火墙或者再查找相关资料加深理解。

**状态分组过滤器**

在传统的分组过滤器中，根据每个分组分离地作出过滤决定。状态过滤器实际地跟踪TCP连接，并用这种**知识**作出过滤决定。

这个知识指什么，我也不知道......

**由于分组过滤器中若设置过滤规则不严谨，则有可能被他人恶意利用。**

故需要使用状态过滤器，通过用一张连接表来跟踪所有进行中的TCP连接来解决该问题。以上作法是可实现的：因为防火墙能够通过观察三次握手（SYN 、 SYNACK 、ACK）来观察一条新连接的开始。当它看到该链接的一个FIN分组时，它能观察该连接的结束......

**应用网关程序**

分组级过滤使得一个机构可以根据IP的内容和TCP/UDP首部执行粗粒度过滤。

有关内部用户的身份信息并不包括在 IP、TCP、UDP的首部中。

为了实现更高水平的安全性，防火墙必须把风阻过滤器和应用程序网关结合起来。应用程序网关除了检查 协议 首部外，还基于应用数据来做策略决定。一个应用程序网管是一个应用程序特定的服务器，所有应用程序数据都必须通过它。多个应用程序网关可以在同一个主机上运行，但是每一个网关都是有自己的进程的单独服务器。