**一、正则表达式**

**1.1 re模块操作**

在Python中需要通过正则表达式对字符串进行匹配的时候，可以使用一个模块，名字为re

**1. re模块的使用过程**

#coding=utf-8

# 导入re模块

import re

# 使用match方法进行匹配操作

result = re.match(正则表达式,要匹配的字符串)

# 如果上一步匹配到数据的话，可以使用group方法来提取数据

result.group()

**2. re模块示例(匹配以itcast开头的语句)**

#coding=utf-8

import re

result = re.match("itcast","itcast.cn")

result.group()

运行结果为：

itcast

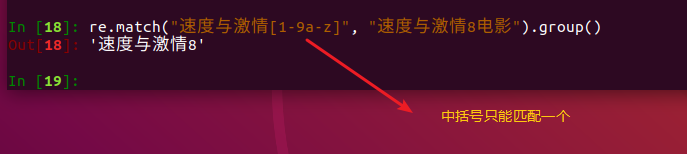
**3. 说明**

* re.match() 能够匹配出以xxx开头的字符串

**1.2 匹配单个字符**

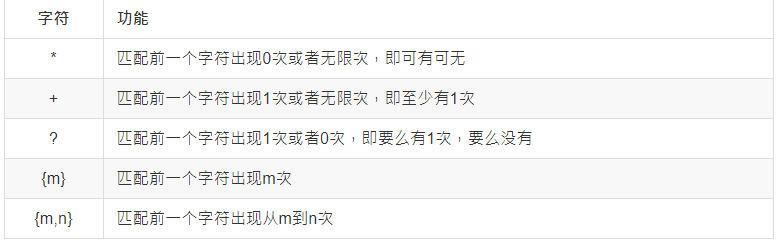
在上一小节中，了解到通过re模块能够完成使用正则表达式来匹配字符串本小节，将要讲解正则表达式的单字符匹配

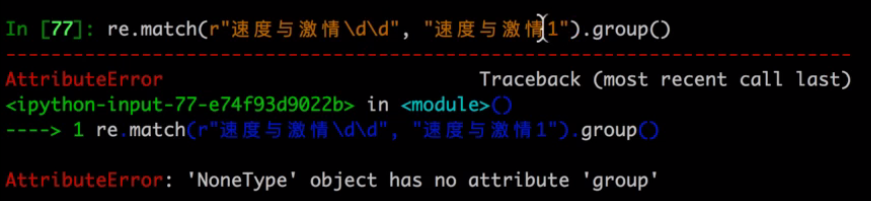


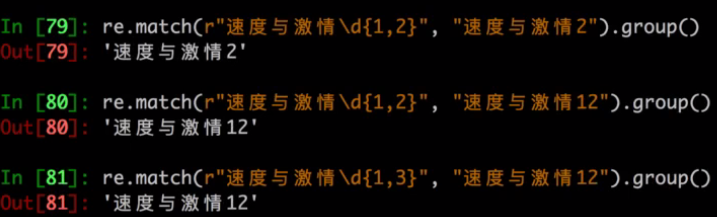


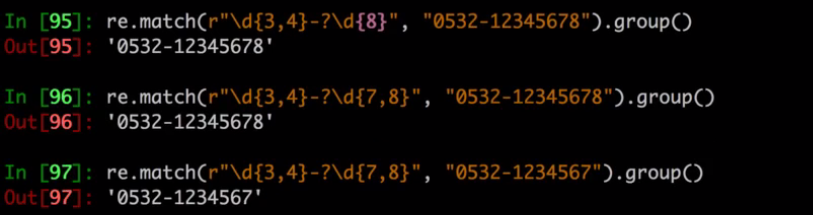
**1.3 匹配多个字符**

匹配多个字符的相关格式







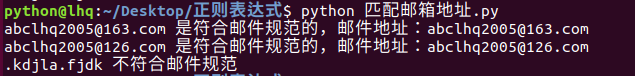


**1.4 匹配开头结尾**



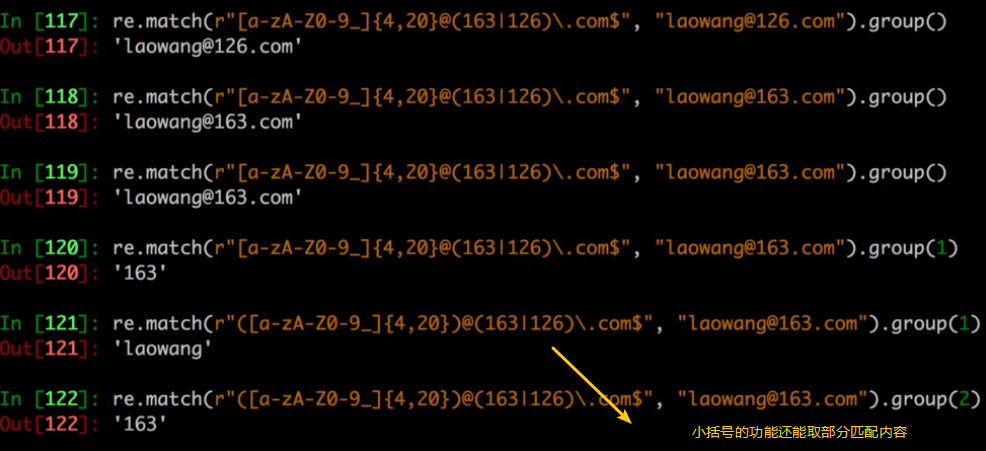






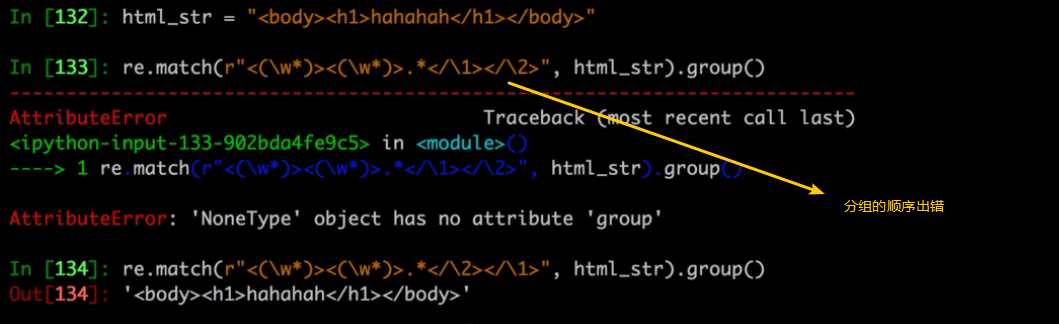
**1.5 匹配分组**



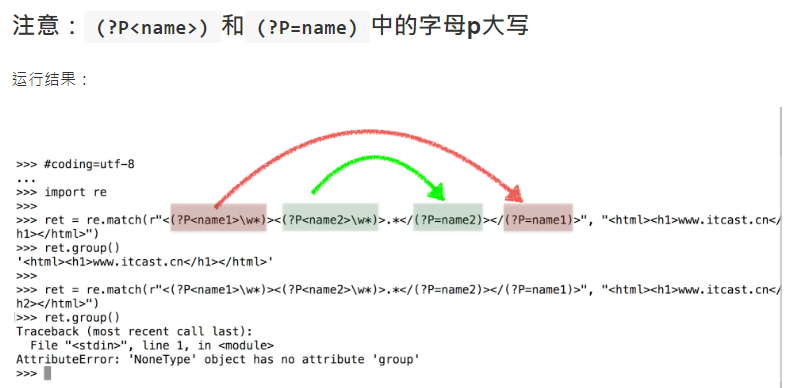






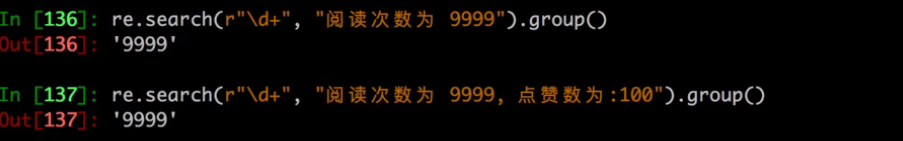






**1.6 re模块的高级用法**

**1. search**



**2. findall**



**3. sub 将匹配到的数据进行替换**





**4. split 根据匹配进行切割字符串，并返回一个列表**

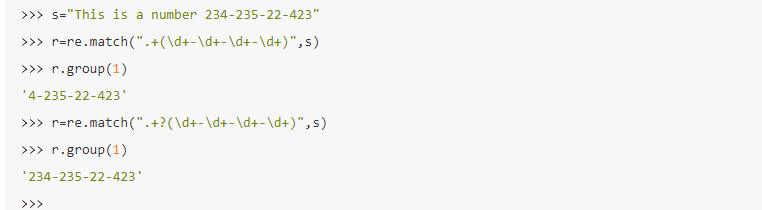


**1.7 python贪婪和非贪婪**

Python里数量词默认是贪婪的（在少数语言里也可能是默认非贪婪），总是尝试匹配尽可能多的字符；

非贪婪则相反，总是尝试匹配尽可能少的字符。

在"\*","?","+","{m,n}"后面加上？，使贪婪变成非贪婪。



正则表达式模式中使用到通配字，那它在从左到右的顺序求值时，会尽量“抓取”满足匹配最长字符串，在我们上面的例子里面，“.+”会从字符串的启始处抓取满足模式的最长字符，其中包括我们想得到的第一个整型字段的中的大部分，“\d+”只需一位字符就可以匹配，所以它匹配了数字“4”，而“.+”则匹配了从字符串起始到这个第一位数字4之前的所有字符。

解决方式：非贪婪操作符“？”，这个操作符可以用在"\*","+","?"的后面，要求正则匹配的越少越好。



**二、HTTP协议简介**

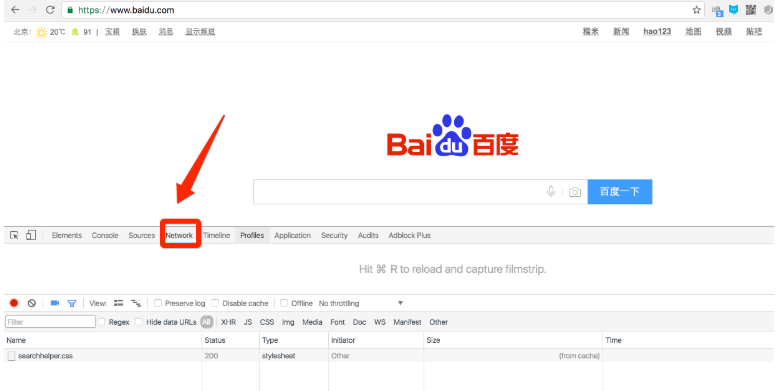
**2.1. 使用谷歌/火狐浏览器分析**

在Web应用中，服务器把网页传给浏览器，实际上就是把网页的HTML代码发送给浏览器，让浏览器显示出来。而浏览器和服务器之间的传输协议是HTTP，所以：

* HTML是一种用来定义网页的文本，会HTML，就可以编写网页；
* HTTP是在网络上传输HTML的协议，用于浏览器和服务器的通信。

Chrome浏览器提供了一套完整地调试工具，非常适合Web开发。

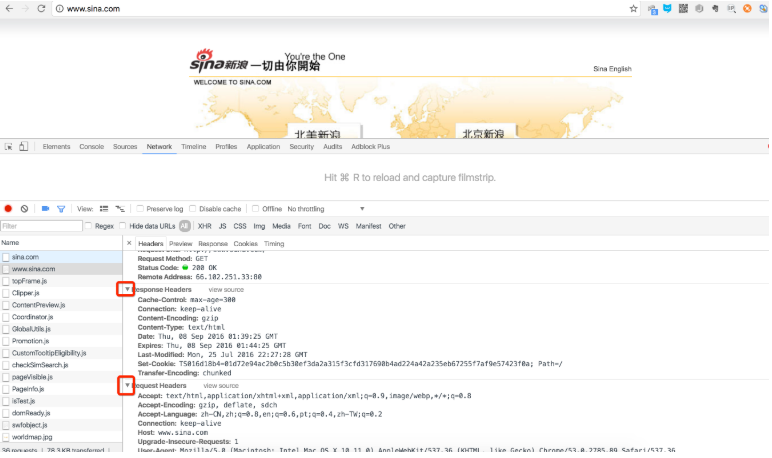
安装好Chrome浏览器后，打开Chrome，在菜单中选择“视图”，“开发者”，“开发者工具”，就可以显示开发者工具：



说明

* Elements显示网页的结构
* Network显示浏览器和服务器的通信

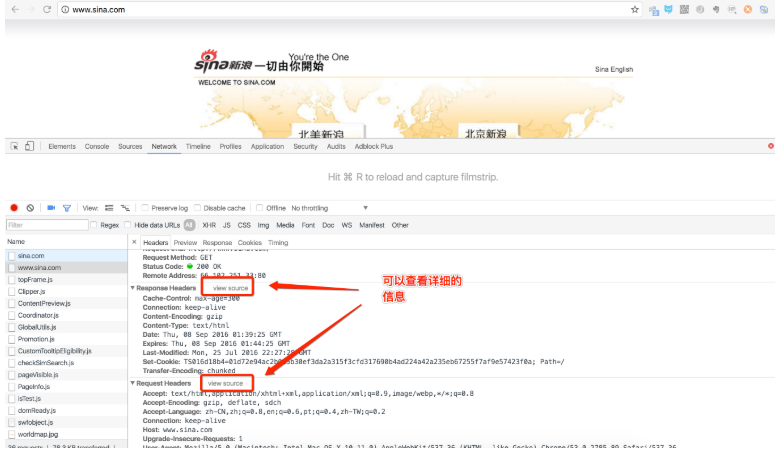
我们点Network，确保第一个小红灯亮着，Chrome就会记录所有浏览器和服务器之间的通信：



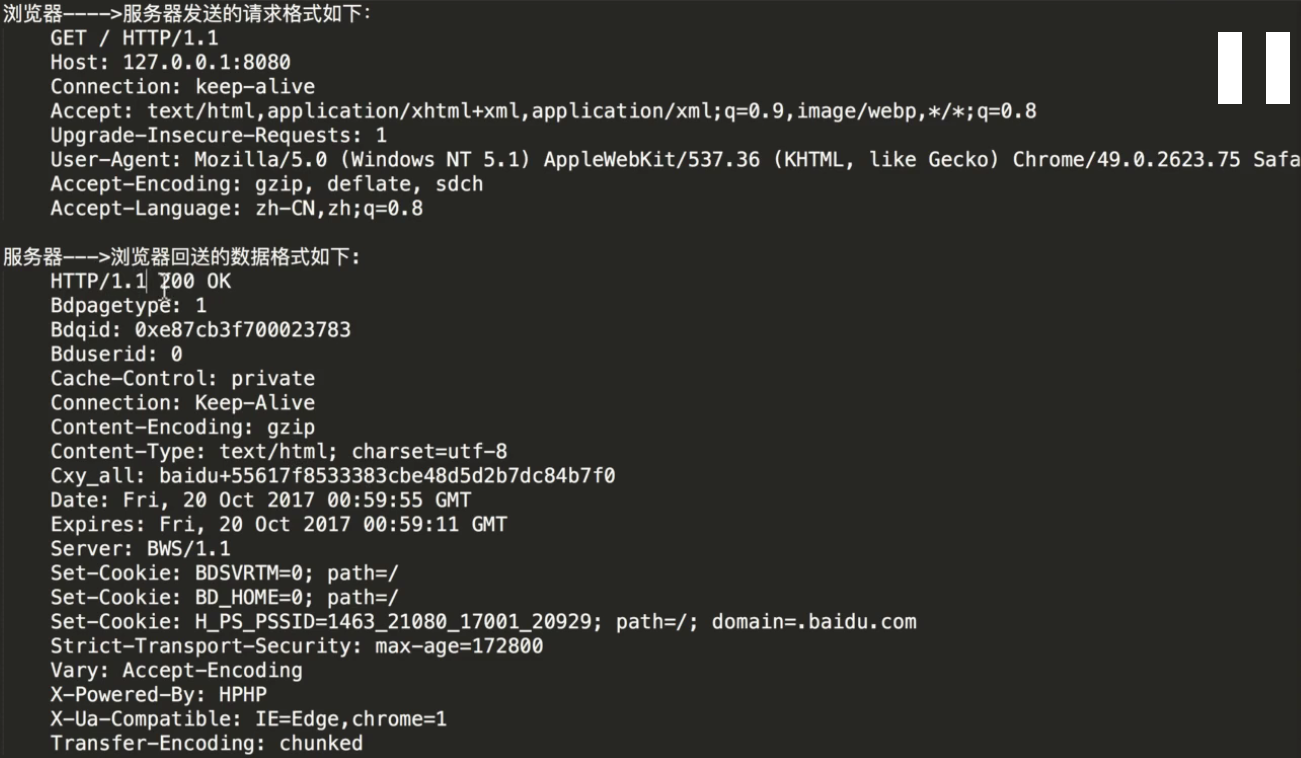
**2.2. http协议的分析**

当我们在地址栏输入www.sina.com时，浏览器将显示新浪的首页。在这个过程中，浏览器都干了哪些事情呢？通过Network的记录，我们就可以知道。在Network中，找到www.sina.com那条记录，点击，右侧将显示Request Headers，点击右侧的view source，我们就可以看到浏览器发给新浪服务器的请求：

**2.2.1 浏览器请求**







说明

最主要的头两行分析如下，第一行：

GET / HTTP/1.1

GET表示一个读取请求，将从服务器获得网页数据，/表示URL的路径，URL总是以/开头，/就表示首页，最后的HTTP/1.1指示采用的HTTP协议版本是1.1。目前HTTP协议的版本就是1.1，但是大部分服务器也支持1.0版本，主要区别在于1.1版本允许多个HTTP请求复用一个TCP连接，以加快传输速度。

从第二行开始，每一行都类似于Xxx: abcdefg：

Host: www.sina.com

表示请求的域名是www.sina.com。如果一台服务器有多个网站，服务器就需要通过Host来区分浏览器请求的是哪个网站。

**2.2.2 服务器响应**

继续往下找到Response Headers，点击view source，显示服务器返回的原始响应数据：



HTTP响应分为Header和Body两部分（Body是可选项），我们在Network中看到的Header最重要的几行如下：

HTTP/1.1 200 OK

200表示一个成功的响应，后面的OK是说明。

如果返回的不是200，那么往往有其他的功能，例如

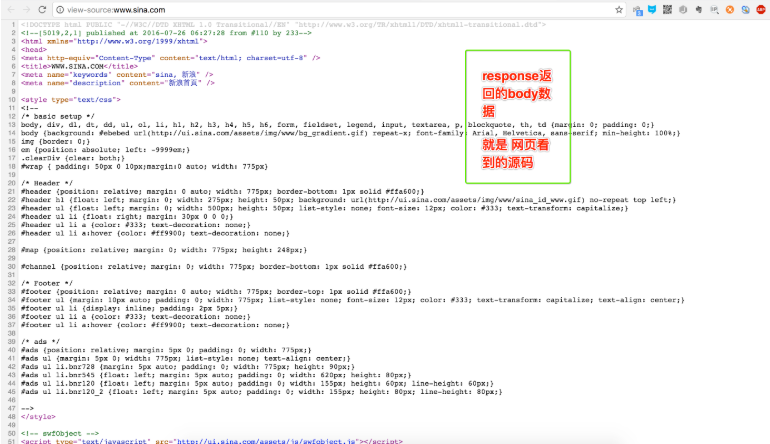
* 失败的响应有404 Not Found：网页不存在
* 500 Internal Server Error：服务器内部出错
* ...等等...

Content-Type: text/html

Content-Type指示响应的内容，这里是text/html表示HTML网页。

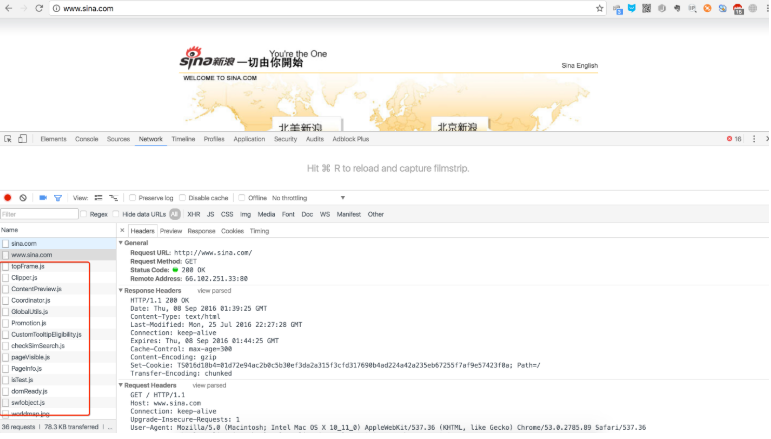
请注意，浏览器就是依靠Content-Type来判断响应的内容是网页还是图片，是视频还是音乐。浏览器并不靠URL来判断响应的内容，所以，即使URL是http://www.baidu.com/meimei.jpg，它也不一定就是图片。

HTTP响应的Body就是HTML源码，我们在菜单栏选择“视图”，“开发者”，“查看网页源码”就可以在浏览器中直接查看HTML源码：



**2.2.3 浏览器解析过程**

**当浏览器读取到新浪首页的HTML源码后，它会解析HTML，显示页面，然后，根据HTML里面的各种链接，再发送HTTP请求给新浪服务器，拿到相应的图片、视频、Flash、JavaScript脚本、CSS等各种资源，最终显示出一个完整的页面。所以我们在Network下面能看到很多额外的HTTP请求。**



**2.3. 总结**

**2.3.1 HTTP请求**

跟踪了新浪的首页，我们来总结一下HTTP请求的流程：

#### 步骤1：浏览器首先向服务器发送HTTP请求，请求包括：

方法：GET还是POST，GET仅请求资源，POST会附带用户数据；

路径：/full/url/path；

域名：由Host头指定：Host: www.sina.com

以及其他相关的Header；

如果是POST，那么请求还包括一个Body，包含用户数据

#### 步骤2：服务器向浏览器返回HTTP响应，响应包括：

响应代码：200表示成功，3xx表示重定向，4xx表示客户端发送的请求有错误，5xx表示服务器端处理时发生了错误；

响应类型：由Content-Type指定；

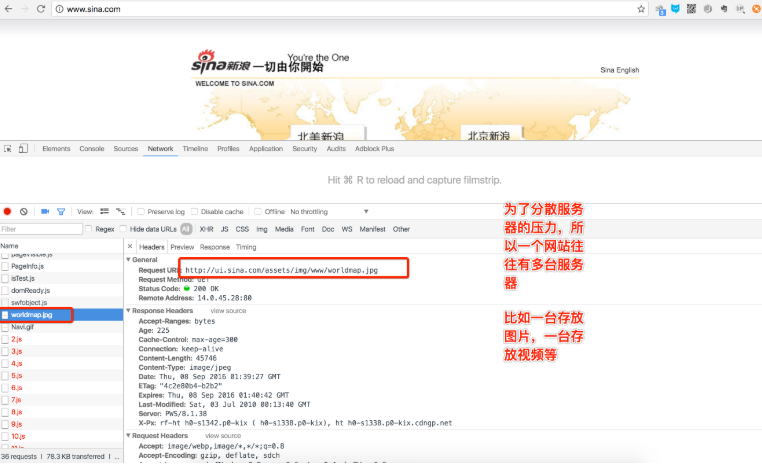
以及其他相关的Header；

通常服务器的HTTP响应会携带内容，也就是有一个Body，包含响应的内容，网页的HTML源码就在Body中。

#### 步骤3：如果浏览器还需要继续向服务器请求其他资源，比如图片，就再次发出HTTP请求，重复步骤1、2。

Web采用的HTTP协议采用了非常简单的请求-响应模式，从而大大简化了开发。当我们编写一个页面时，我们只需要在HTTP请求中把HTML发送出去，不需要考虑如何附带图片、视频等，浏览器如果需要请求图片和视频，它会发送另一个HTTP请求，因此，一个HTTP请求只处理一个资源(此时就可以理解为TCP协议中的短连接，每个链接只获取一个资源，如需要多个就需要建立多个链接)

HTTP协议同时具备极强的扩展性，虽然浏览器请求的是http://www.sina.com的首页，但是新浪在HTML中可以链入其他服务器的资源，比如<img src="http://i1.sinaimg.cn/home/2013/1008/U8455P30DT20131008135420.png">，从而将请求压力分散到各个服务器上，并且，一个站点可以链接到其他站点，无数个站点互相链接起来，就形成了World Wide Web，简称WWW。



**2.3.2 HTTP格式**

每个HTTP请求和响应都遵循相同的格式，一个HTTP包含Header和Body两部分，其中Body是可选的。

HTTP协议是一种文本协议，所以，它的格式也非常简单。

#### HTTP GET请求的格式：

GET /path HTTP/1.1

Header1: Value1

Header2: Value2

Header3: Value3

每个Header一行一个，换行符是\r\n。

#### HTTP POST请求的格式：

POST /path HTTP/1.1

Header1: Value1

Header2: Value2

Header3: Value3

body data goes here...

当遇到连续两个\r\n时，Header部分结束，后面的数据全部是Body。

#### HTTP响应的格式：

200 OK

Header1: Value1

Header2: Value2

Header3: Value3

body data goes here...

HTTP响应如果包含body，也是通过\r\n\r\n来分隔的。

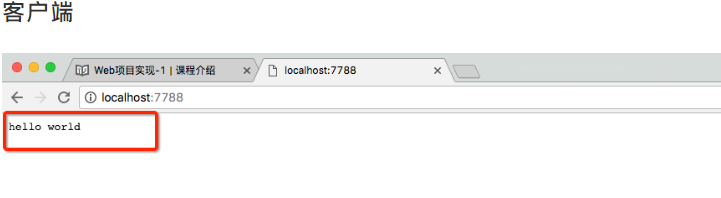
请再次注意，Body的数据类型由Content-Type头来确定，如果是网页，Body就是文本，如果是图片，Body就是图片的二进制数据。

当存在Content-Encoding时，Body数据是被压缩的，最常见的压缩方式是gzip，所以，看到Content-Encoding: gzip时，需要将Body数据先解压缩，才能得到真正的数据。压缩的目的在于减少Body的大小，加快网络传输。

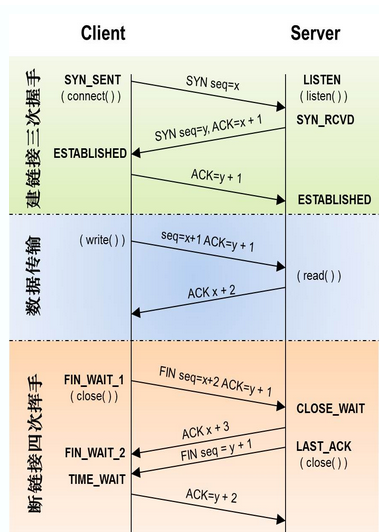
**三、Web服务器**

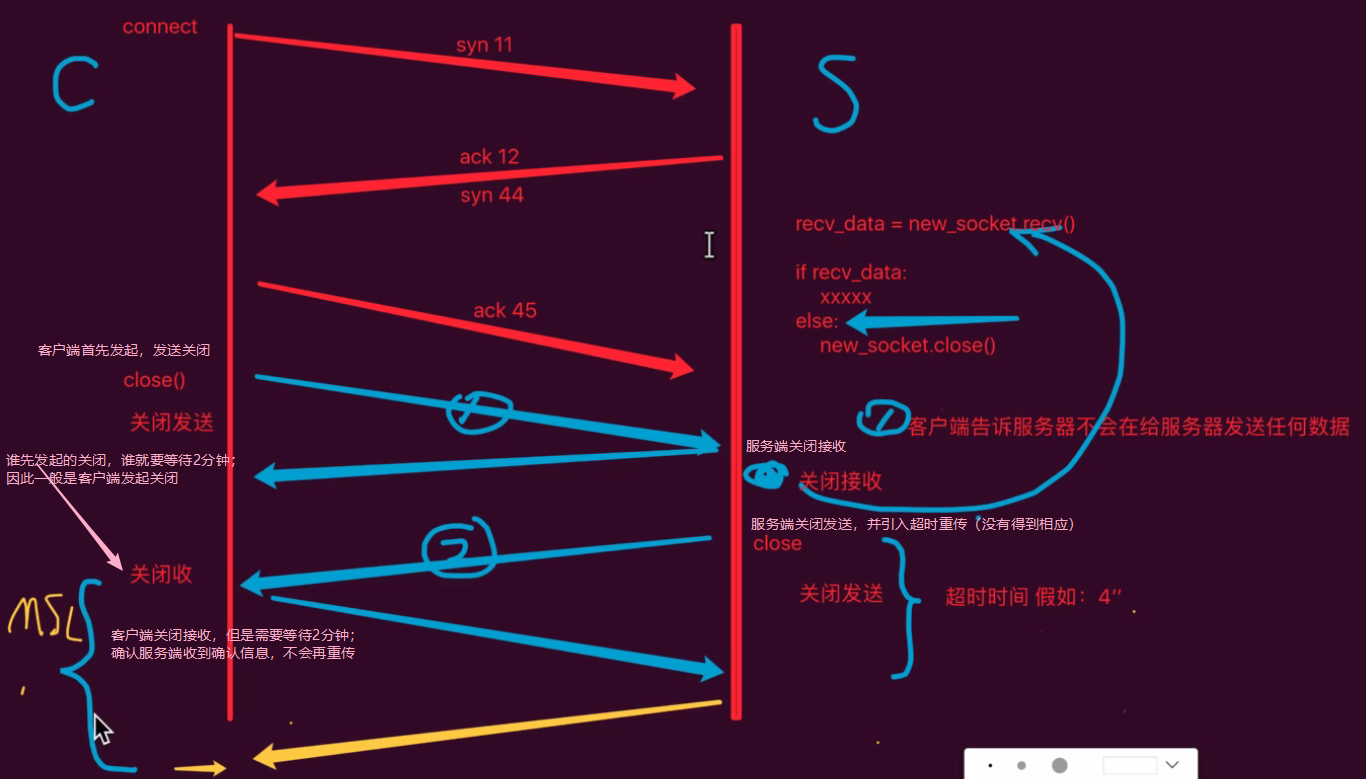
**3.1 Web静态服务器-显示固定的页面**



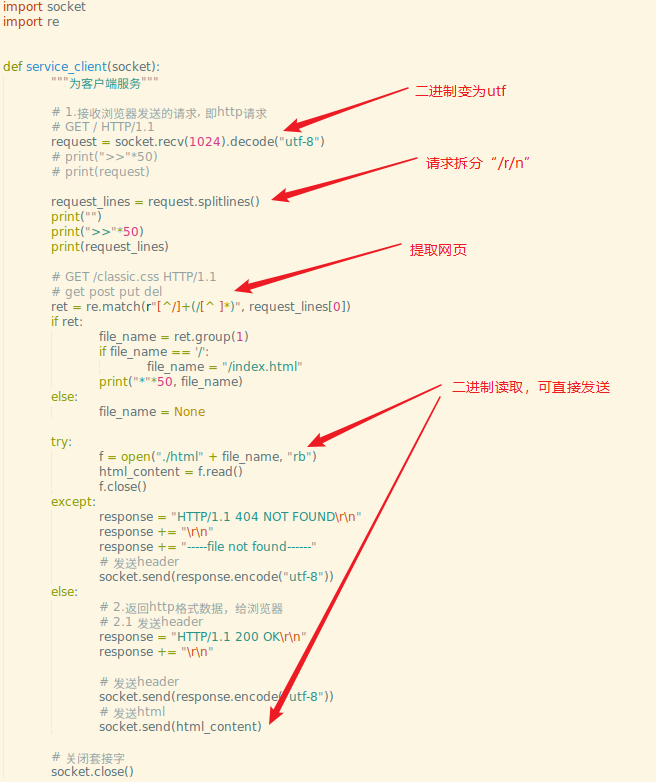


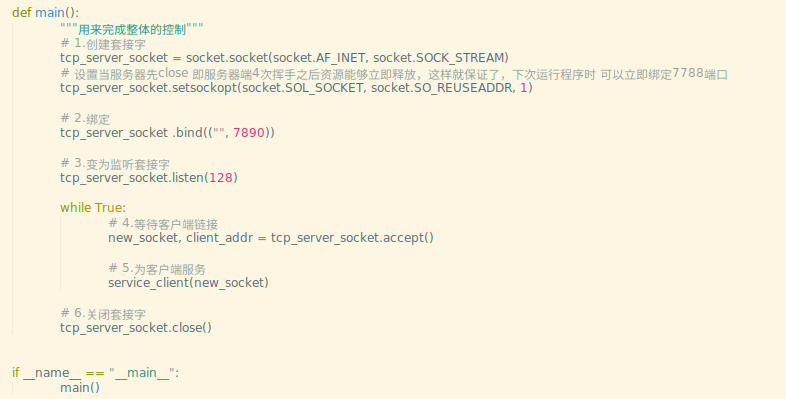
**3.2 tcp的3次握手和4次挥手**





**3.3 Web静态服务器-显示需要的页面**

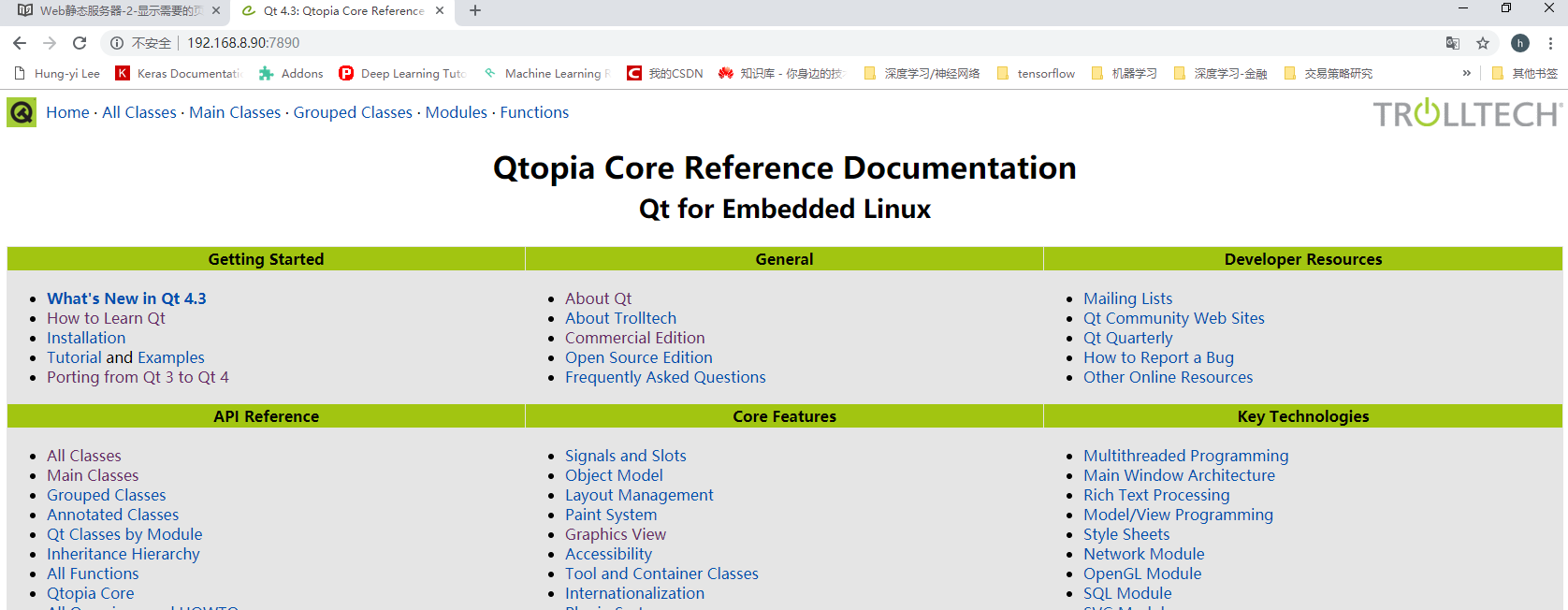




服务端：



客户端：



**3.4 Web静态服务器-多进程**



**3.5 Web静态服务器-多线程**



**3.5 Web静态服务器-多协程**



**3.6 Web静态服务器-单进程单线程非堵塞模式**





同时处理两个客户端，同时因为程序里面有等待，导致了快速发送数据时，接收缓存。

**3.7 tcp长连接和短连接**

TCP在真正的读写操作之前，server与client之间必须建立一个连接，当读写操作完成后，双方不再需要这个连接时它们可以释放这个连接，连接的建立通过三次握手，释放则需要四次握手，所以说每个连接的建立都是需要资源消耗和时间消耗的。

**3.7.1. TCP短连接**

模拟一种TCP短连接的情况:

* client 向 server 发起连接请求
* server 接到请求，双方建立连接
* client 向 server 发送消息
* server 回应 client
* 一次读写完成，此时双方任何一个都可以发起 close 操作

在步骤5中，一般都是 client 先发起 close 操作。当然也不排除有特殊的情况。

从上面的描述看，短连接一般只会在 client/server 间传递一次读写操作！

**3.7.2. TCP长连接**

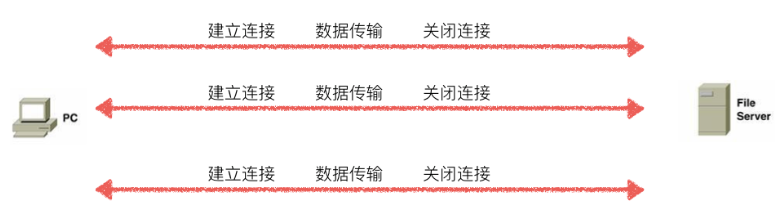
再模拟一种长连接的情况:

* client 向 server 发起连接
* server 接到请求，双方建立连接
* client 向 server 发送消息
* server 回应 client
* 一次读写完成，连接不关闭
* 后续读写操作...
* 长时间操作之后client发起关闭请求

**3.7.3. TCP长/短连接操作过程**

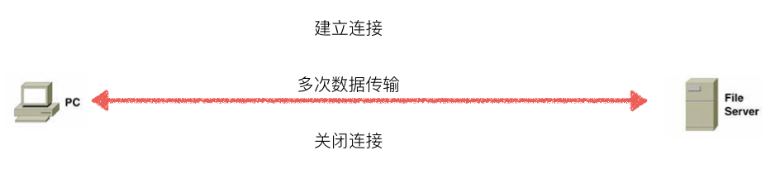
1 短连接的操作步骤是：

建立连接——数据传输——关闭连接...建立连接——数据传输——关闭连接



2 长连接的操作步骤是：

建立连接——数据传输...（保持连接）...数据传输——关闭连接



**4. TCP长/短连接的优点和缺点**

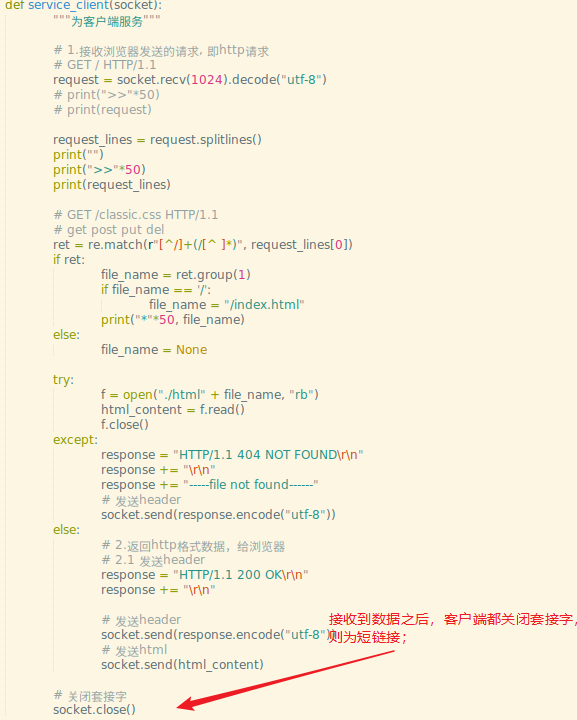
* 长连接可以省去较多的TCP建立和关闭的操作，减少浪费，节约时间。对于频繁请求资源的客户来说，较适用长连接。
* client与server之间的连接如果一直不关闭的话，会存在一个问题，随着客户端连接越来越多，server早晚有扛不住的时候，这时候server端需要采取一些策略，如关闭一些长时间没有读写事件发生的连接，这样可以避免一些恶意连接导致server端服务受损；
* 如果条件再允许就可以以客户端机器为颗粒度，限制每个客户端的最大长连接数，这样可以完全避免某个蛋疼的客户端连累后端服务。
* 短连接对于服务器来说管理较为简单，存在的连接都是有用的连接，不需要额外的控制手段。
* 但如果客户请求频繁，将在TCP的建立和关闭操作上浪费时间和带宽。

**5. TCP长/短连接的应用场景**

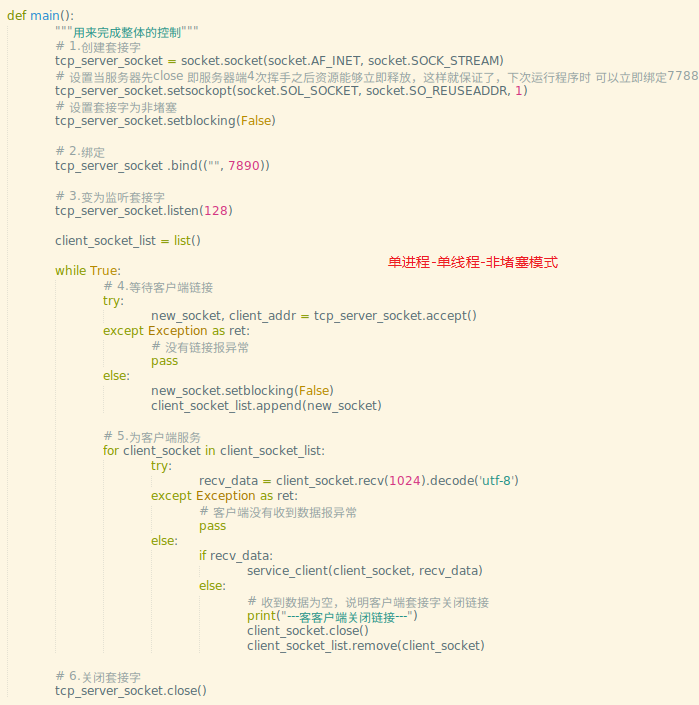
* 长连接多用于操作频繁，点对点的通讯，而且连接数不能太多情况。每个TCP连接都需要三次握手，这需要时间，如果每个操作都是先连接，再操作的话那么处理速度会降低很多，所以每个操作完后都不断开，再次处理时直接发送数据包就OK了，不用建立TCP连接。
* 例如：数据库的连接用长连接，如果用短连接频繁的通信会造成socket错误，而且频繁的socket 创建也是对资源的浪费。
* 而像WEB网站的http服务一般都用短链接，因为长连接对于服务端来说会耗费一定的资源，而像WEB网站这么频繁的成千上万甚至上亿客户端的连接用短连接会更省一些资源，如果用长连接，而且同时有成千上万的用户，如果每个用户都占用一个连接的话，那可想而知吧。所以并发量大，但每个用户无需频繁操作情况下需用短连好。

**6. 之前代码分析**

之前的多进程-多线程-多协程版本，用的处理函数：



**3.8 Web静态服务器-单进程单线程非堵塞-长连接模式**





**3.9 Web静态服务器-epoll**

什么是epoll

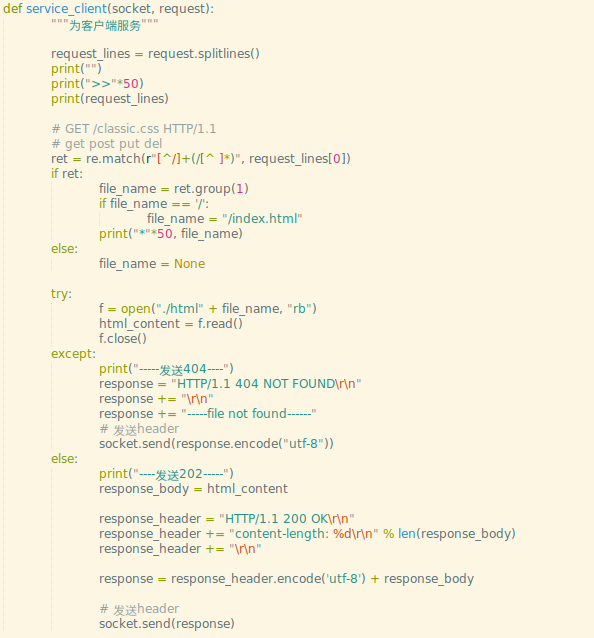
epoll是什么？按照man手册的说法：是为处理大批量句柄而作了改进的poll。当然，这不是2.6内核才有的，它是在2.5.44内核中被引进的(epoll(4) is a new API introduced in Linux kernel 2.5.44)，它几乎具备了之前所说的一切优点，被公认为Linux2.6下性能最好的多路I/O就绪通知方法。

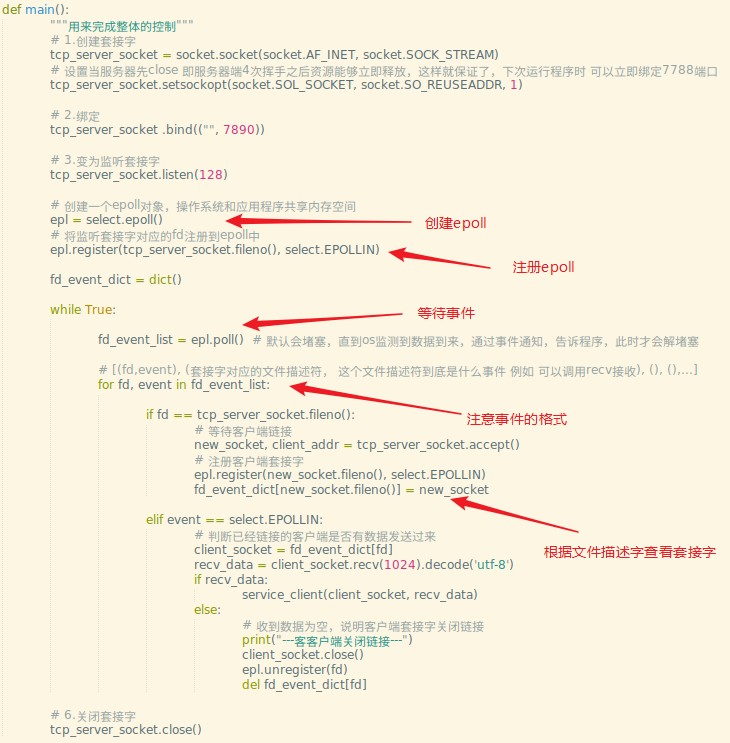
IO 多路复用

就是我们说的select，poll，epoll，有些地方也称这种IO方式为event driven IO。

select/epoll的好处就在于单个process就可以同时处理多个网络连接的IO。

它的基本原理就是select，poll，epoll这个function会不断的轮询所负责的所有socket，当某个socket有数据到达了，就通知用户进程。





说明

* EPOLLIN （可读）
* EPOLLOUT （可写）
* EPOLLET （ET模式）

epoll对文件描述符的操作有两种模式：LT（level trigger）和ET（edge trigger）。LT模式是默认模式，LT模式与ET模式的区别如下：

* LT模式：当epoll检测到描述符事件发生并将此事件通知应用程序，应用程序可以不立即处理该事件。下次调用epoll时，会再次响应应用程序并通知此事件。
* ET模式：当epoll检测到描述符事件发生并将此事件通知应用程序，应用程序必须立即处理该事件。如果不处理，下次调用epoll时，不会再次响应应用程序并通知此事件。

### 小总结

I/O 多路复用的特点：

通过一种机制使一个进程能同时等待多个文件描述符，而这些文件描述符（套接字描述符）其中的任意一个进入读就绪状态，epoll()函数就可以返回。 所以, IO多路复用，本质上不会有并发的功能，因为任何时候还是只有一个进程或线程进行工作，它之所以能提高效率是因为select\epoll 把进来的socket放到他们的 '监视' 列表里面，当任何socket有可读可写数据立马处理，那如果select\epoll 手里同时检测着很多socket， 一有动静马上返回给进程处理，总比一个一个socket过来,阻塞等待,处理高效率。

当然也可以多线程/多进程方式，一个连接过来开一个进程/线程处理，这样消耗的内存和进程切换页会耗掉更多的系统资源。 所以我们可以结合IO多路复用和多进程/多线程 来高性能并发，IO复用负责提高接受socket的通知效率，收到请求后，交给进程池/线程池来处理逻辑。

### 参考资料

* 如果想了解下epoll在Linux中的实现过程可以参考：<http://blog.csdn.net/xiajun07061225/article/details/9250579>

### 知识扩展-C10K问题

参考文章 :

《单台服务器并发TCP连接数到底可以有多少》 <http://www.52im.net/thread-561-1-1.html>

《上一个10年，著名的C10K并发连接问题》 <http://www.52im.net/thread-566-1-1.html>