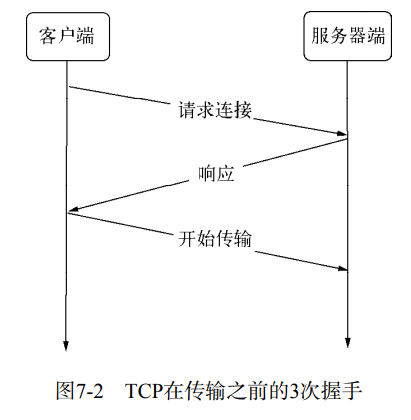
# 网络编程（150）

7.1.1 构建tcp服务

TCP连接时需要三次握手。



开启tcp服务器[openTcp](openTcp.js)，连接tcp服务器[connect](connectedTcp.js)。

7.1.3 tcp服务的事件

服务器事件：

* Listen事件：调用server.listen()绑定服务器端口或者domain socket后触发。绑定方法，server.listen(port,fun)的第二个参数。
* connection事件：每个客户端套接字连接到服务器端时触发。绑定方法，net.createServer（）的最后一个参数（第一或者第二）。
* close事件：服务器关闭时会触发。注意，如果存在连接，这个事件不会被触发直到所有的连接关闭。
* error事件。

连接事件

* + data：一端调用write()后，另一端触发data事件。
  + end：连接中任意一端发送了FIN数据会触发事件。
  + connect：用于客户端，套接字和服务器连接成功触发。
  + drain：调用write()的当前端触发。
  + error：
  + close：套接字完全关闭时触发事件。
  + timeout：一定时间后连接不再活跃，触发该事件。

Tcp套子是可读可写的Stream对象，可以用pipe（）巧妙实现管道操作。

Tcp针对网络中的小数据包采用Nagle算法。只有缓存区的数据到达一定数量时才会发送，但是这有可能造成数据的延迟。可以使用socket.setNoDelay(true)使得write（）可以立即发送数据。调用write（）并不是每次都会触发data事件，关掉Nagle算法后，触发一次data事件。

7.2 构建upd服务器

开启ucp服务器[openUdp](openUdp.js)，连接tcp服务器[connect](connectedUdp.js)。

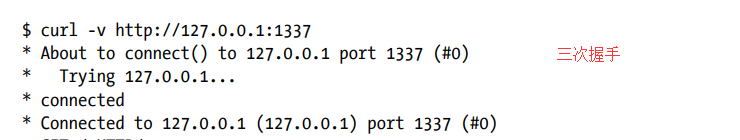
7.2.4 UDP套字事件

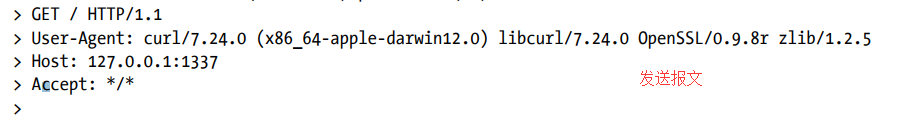
Message：监听端口接收到UDP套字后触发，触发携带带有数据的buffer和一个远程地址。

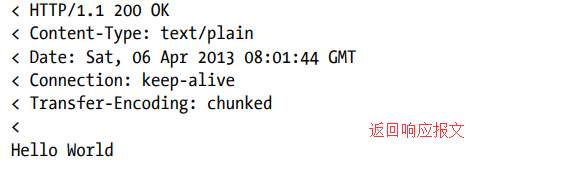
Listening：UDP开始侦听时触发。

7.3 构建HTTP服务

7.1.1 HTTP请求



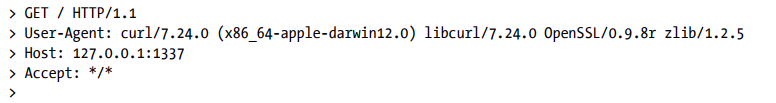




浏览器其实是一个HTTP请求代理。主要做的事情是发送HTTP请求和响应处理HTTP请求。

7.3.2 http模块

1. http请求

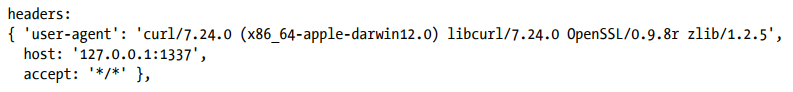


请求报头通过http\_parser进行解析。

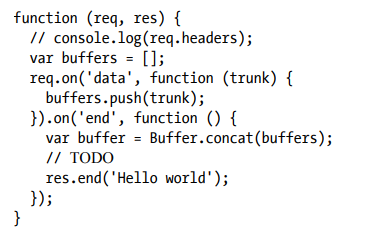
报头第一行GET / HTTP/1.1被解析为req的属性：

* Req.mothod：GET，常见的有GET、POST、DELETE、PUT、CONNECT。
* Req.url：值为/
* Req.httpVersion：1.1

其余报文以key：value形式保存在req.headers属性上。



报文体部分抽象为一个只读流，如果业务逻辑需要读报文体中的数据，需要在数据流结束之后才能进行。



1. http 响应

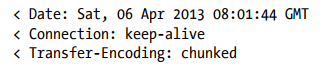
http响应封装了对底层连接的写操作，是一个可写流。



其分为setHeader和writeHeader两步，我们可以调用setHeader多次设置，但只有调用writeHeader报头才会写入连接。实际产生的报文为



http模块还会帮你设置一些头部信息



报文体通过res.write()和res.end()实现。两者差别在于后者调用时会先调用一个write()发送数据，然后发送信号告知服务器响应结束。响应结束后服务器可能会将当前连接用户下一个请求，或者关闭连接。报头的发送在报体前，一但开始传输数据，无法操作报头。

结束是必须调用res.end（）方法结束连接。

7.3.3 http服务事件

与tcp服务一样，http服务也是一个EventEmitter实例。

* Connection：在http请求前，建立tcp连接时触发。
* Request：建立tcp连接后，http模块在数据流中抽象出HTTP请求和HTTP响应。当请求数据发送到服务端，在解析出HTTP请求头时触发。
* Close：与tcp事件一样，在调用server.close()方法停止接收新连接，且在已有连接都关闭之后触发。给server.close()传一个callback来注册该事件。
* checkContinue：某些客户端在发送大数据时，不会直接发送大数据，而是先发送一个头部Expect: 100-continue的请求，服务器会触发checkContinue。如果没有注册改时间，服务器自动响应100 Continue，表示数据可以上传，如果不接受较多的数据时，响应400 Bad Request响应客户端。发生这个事件时不会触发request事件，两个事件互斥。客户端收到100 Continue后重新发起请求时才会触发request事件。
* Connect：客户端发起Connect请求时触发。如果不注册该事件则发起该请求的链接将会关闭。
* Upgrade：客户端要求升级连接的协议时，会在请求头上带upgrade字段，触发服务端upgrade事件。如果没有注册该事件请求的连接将会关闭。
* clientError：连接的客户端触发error事件时触发。

7.3.4 http客户端

http模块的http.request(options, connect)用于构造http客户端，报文头和报文体由他产生。

[OpenHTTP](openHttp.js)。[connectedHttp](connectedHttp.js)。

Option参数决定了请求头的内容，有如下选项：

* + Host：服务器域名或者IP，默认localhost。
  + Hostname：服务器名称。
  + Port：服务器端口，默认80。
  + localAddress：建立网络连接的本地网卡。
  + socketPath：Domain套接字路径。
  + Path：请求路径，默认/。
  + Method：http请求方法，默认GET。
  + Headers：请求头对象。
  + Auth：Basic认证。

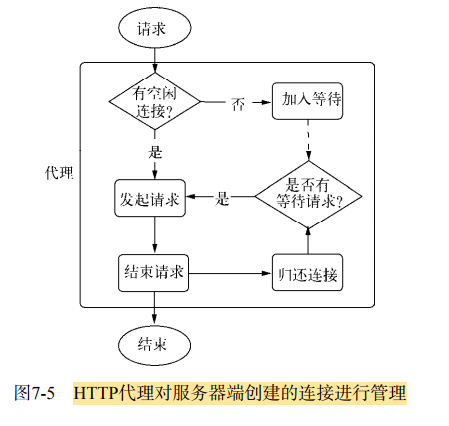
报文体的请求对象的write方法向连接中写入数据，end方法告知报文结束。

1. HTTP响应

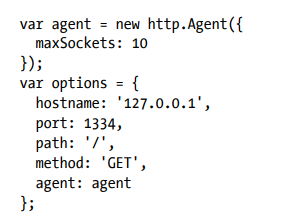
HTTP客户端响应ClientRequest在解析完报文头之后立即出发response事件并提供一个参数。

1. HTTP代理

http提供的ClientRequest对象基于TCP，在keepalive情况下，一个底层会话可以多次请求。为了重连TCP，http用了一个客户端代理对象http.globalAgent，它对每个服务器进行监听。默认情况下最多可以同时进行5次连接。



同时发起10个请求，同时进行的只有5个，剩下的5个进入等待池。如需改变，可以在options里增加agent，也可以将agent设置为false，即不限制。



1. 客户端事件

Response：当服务器返回响应触发事件。

Socket：底层连接池建立连接分配给对象，触发事件。

Connect：客户端向服务器发起CONNECT请求，返回200时触发事件。

Upgrade：客户端发起upgrade，服务器返回了101 Switching Protocols触发。

Continue：客户端欲发送大量数据，发起Expect: 100-continue请求头，服务器返回100 continue触发。

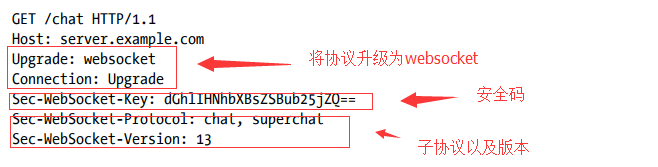
7.4 WebSocket服务

WebSocket基于事件，实现与客户端之间的长连接，node适合处理高并发事件。Websocket好处：

* 只建立一个TCP连接
* 服务器可以推消息到客户端
* 协议头更轻量，减少数据传输

WebSocket是基于TCP完成的，websocket只要有握手和传输两个协议。

7.4.1握手



Sec-WebSocket-Key值为随机生成的BASE64编码字符串，服务器收到后与258EAFA5-E914-47DA-95CA-C5AB0DC85B11相连，然后通过sha1安全散列算法计算，再进行base64编码返回客户端。算法如下





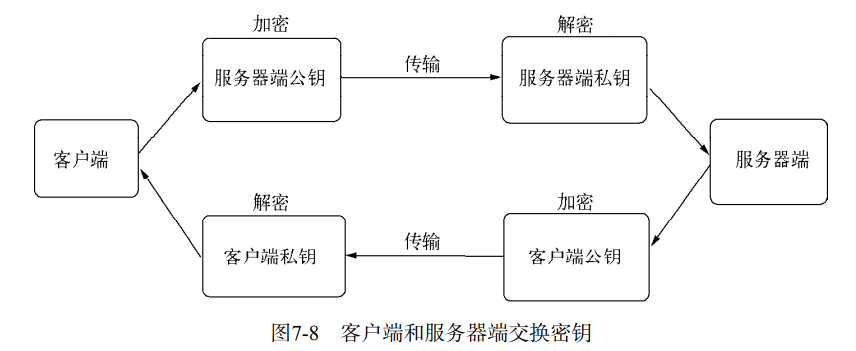
返回之后，客户端验证安全码，如果成功则开始传数据。

7.5 网络服务与安全

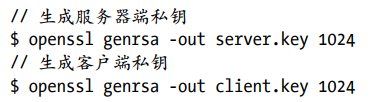
Node在网络安全上主要有crypto、tls、https三个模块，crypto主要用于加密，加密算法有SHA1，MD5。Tls模块和net模块类似，只是在tcp连接上进行了加密，https模块与http模块类似，区别在于连接上加密。

7.5.1 tls/ssl

1.服务器和客户端都有自己的公私钥，公钥和私钥是配对的，在建立安全连接前，服务器和客户端要交换公钥，用对方的公钥加密数据才能发送。

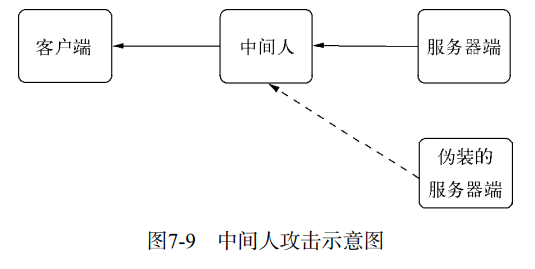


Node底层采用openssl实现TLS/SSL，下面先生成1024位长的RSA私钥文件，再生成公钥。





这种模式还是会被中间人攻击，中间人对服务器扮演客户端，对客户端扮演服务器，所以双方察觉不到中间人存在。

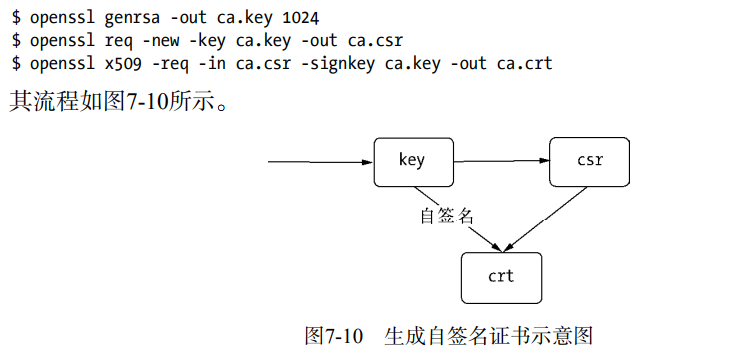


TLS/SSL引入数字证书来进行认证，数字证书中包含服务器名称和主机、服务器密钥、签名颁发机构的名称，来自签名颁发机构的签名。

1. 数字证书

服务器需要通过自己的私钥生成CSR（Certificate Signing Request，证书签名请求），CA机构通过通过这个文件颁发属于服务器的证书，通过CA机构能知道证书是否合法。

颁发证书是一个费事费钱的过程，中小企业可以采用自签名证书。也就是自己扮演CA机构，给自己的服务器颁发证书。以下是生成私钥，CSR文件，通过密钥生成自签名文件过程。



7.5.2 TLS服务

1. 创建服务器端