网络编程

本章目录：

一、网络编程基础

> *网络编程对于任何联网终端对应的平台来说都是至关重要的，包括iOS平台。在笔者目前的开发过程中，几乎全部应用程序都是和网络打交道的（除了一款毕设游戏外），掌握网络编程对于一个开发者的重要性无需再多解释。本章我们将由深入浅再到深去了解网络编程。4、5、6节应该是最简单的章节，也是目前大多数程序员掌握的部分，如果想要更加灵活的使用网络，必须学会2、3、4，熟练使用后，最终我们就可以拔高，自己开发一套网络框架给自己、给公司或者全人类一起使用。*

本节我们将从宏观理解网络的构成，以及一些专业术语。本节不仅限于iOS或Mac OS开发，而是整个互联网。

1. 沟通的升级

当我们尝试和其他人或事物进行沟通时，我们的沟通方式会逐渐加深，由最初非常简单方式演变为更加熟练的沟通方式。网络的传输也是如此在发展，诞生了一个一个的协议，并得到了公认。

目前实现网络交互一共需要7层结构，由下至上分别是：

1. 物理层
2. 数据链路层
3. 网络层
4. 传输层
5. 会话层
6. 表示层
7. 应用层

2. 什么是协议

简单的理解，协议就是大家都要遵从的一套规则，需要我们在各个层上都遵守对应的协议，已达到传递数据的统一性，系统的稳定性。就像两个人沟通时，需要按照约定好的方式沟通，比如用声音作为媒介沟通，我们可能要遵循同一种语言协议，否则两者根本无法沟通。

3. 常见的协议

公认的协议有非常多，应用在不同的层级上。我们当然也可以定义自己的协议，可以只在自己的工程中使用。如果说协议制定的非常好而且开源，那么有可能将来会被广泛的使用，最终成为公认的协议。

3.1 IP协议

- IP协议是网络层的协议，目前常用的是IPv4传输协议，下一代网络层协议为IPv6协议，是对IPv4的加强，提高了很多方面的性能，比如IPv6把IP地址由 32 位增加到 128 位，平均分布在地球表面上的话，每平米有 4*10^18 个 IPv6 地址，用完是几乎不可能的事，等建立外星殖民地吧。
- IP协议主要解决终端的唯一身份认证问题，就像电话号码协议。

网络层还有一些其他的协议，比如IPX，AppleTalk DDP等等

3.2 TCP/IP协议

- TCP/IP是传输层协议，可以从名字上看出，TCP/IP协议的前提是网络层满足IP协议。
- TCP/IP协议主要解决的问题是数据在网络上如何传输的问题。如同数据在电话中可以用声音信号传输，后来移动电话也可以用SMS协议即短信方式的传输。
- TCP/IP是双向传输协议，有去有回。

传输层协议还有像UDP，UGP协议等等。

3.3 HTTP协议

- HTTP（超文本）协议，主要应用于web应用，属于应用层协议。HTTP协议是基于TCP/IP协议的。
- 解决的问题是数据的格式规范问题

应用层协议数量是最多的，也是我们自定义协议最成熟的一层。我们使用公认的传输层、会话层和表示层协议，就可以自定义出很多的应用层协议。比如常用的FTP协议

3.4 其他层的协议

- 第一层和第二层基本和硬件相关，笔者知之甚少，但是有些也耳熟能详的，比如IEEE 802.5/802.2 就是一种Wi-Fi协议,属于数据链路层。
- 会话层：基本上XMPP协议是最熟知的会话层协议，比如GTalk就是用了该协议。以前

的MSN和企鹅公司的QQ都使用了自己的协议。

- 表示层：这里面基本上就是用来描述数据的格式的，这里面协议太多了：TIFF,GIF,JPEG,PICT,ASCII,EBCDIC,encryption,MPEG,MIDI,HTML等等

学习后面的内容，清楚的知道我们处理的是哪一层非常关键。

二、Socket(套接字)通信

网络进程间通信在传输层中，基本上都是用的Socket，Cocoa中使用Socket有三种方式：

- Cocoa层：NSURL
- Core Foundation层：基于 C 的 CFNetwork 和 CFNetServices
- OS层:基于 C 的 BSD socket

解释：

1. Cocoa层是最上层的基于 Objective-C 的 API，比如 URL访问，NSStream，Bonjour，GameKit等，这是大多数情况下我们常用的 API。Cocoa 层是基于 Core Foundation 实现的。
2. Core Foundation层：因为直接使用 socket 需要更多的编程工作，所以苹果对 OS 层的 socket 进行简单的封装以简化编程任务。该层提供了 CFNetwork 和 CFNetServices，其中 CFNetwork 又是基于 CFStream 和 CFSocket。
3. OS层：最底层的 BSD socket 提供了对网络编程最大程度的控制，但是编程工作也是最多的。因此，苹果建议我们使用 Core Foundation 及以上层的 API 进行编程。

其中os层中提供的BSD Socket最为灵活，但是抽象层次比较低，使用最为复杂。本节我们将学习如何使用BSD Socket基础接口编程，最终实现两个设备间的通信。

1. 什么是Socket

Socket起源于 Unix，而Unix/Linux基本哲学之一就是“一切皆文件”，都可以用“打开open –> 读写write/read –> 关闭close”模式来操作。我的理解就是Socket就是该模式的一个实现，socket即是一种特殊的文件，一些socket函数就是对其进行的操作 （读/写IO、打开、关闭），这些函数我们在后面进行介绍。

2. Socket编程接口

2.1 Socket()

接口定义：

```
int socket(int domain, int type, int protocol);
```

socket()函数对应于普通文件的打开操作。普通文件的打开操作返回一个文件描述字，而socket()用于创建一个socket描述符（socket descriptor），它唯一标识一个socket。这个socket描述字跟文件描述字一样，后续的操作都有用到它，把它作为参数，通过它来进行一些

读写操作。

socket()函数的三个参数分别为：

- domain：即协议域，又称为协议族（family）。常用的协议族有，AF_INET、AF_INET6、AF_LOCAL（或称AF_UNIX，Unix域socket）、AF_ROUTE等等。协议族决定了socket的地址类型，在通信中必须采用对应的地址，如AF_INET决定了要用ipv4地址（32位的）与端口号（16位的）的组合、AF_UNIX决定了要用一个绝对路径名作为地址。
- type：指定socket类型。常用的socket类型有，SOCK_STREAM、SOCK_DGRAM、SOCK_RAW、SOCK_PACKET、SOCK_SEQPACKET等等
- protocol：故名思意，就是指定协议。常用的协议有，IPPROTO_TCP、IPPTOTO_UDP、IPPROTO_SCTP、IPPROTO_TIPC等，它们分别对应TCP传输协议、UDP传输协议、STCP传输协议、TIPC传输协议。通常使用0，会自动选择type类型对应的默认协议。

2.2 bind()

接口定义：

```
int bind(int sockfd, const struct sockaddr *addr, socklen_t addrlen);
```

bind()函数把一个地址族中的特定地址赋给socket。例如对应AF_INET、AF_INET6就是把一个ipv4或ipv6地址和端口号组合赋给socket。

函数的三个参数分别为：

- sockfd：即socket描述字，它是通过socket()函数创建了，唯一标识一个socket。bind()函数就是将给这个描述字绑定一个名字。
- addr：一个const struct sockaddr *指针，指向要绑定给sockfd的协议地址。这个地址结构根据地址创建socket时的地址协议族的不同而不同，如ipv4对应的是：

  ```
  struct sockaddr_in {
      sa_family_t    sin_family;
      in_port_t      sin_port;
      struct in_addr sin_addr;
  };

  struct in_addr {
      uint32_t       s_addr;
  };
  ```

- addrlen：对应的是地址的长度。

通常服务器在启动的时候都会绑定一个众所周知的地址（如ip地址+端口号），用于提供服

务，客户就可以通过它来接连服务器；而客户端就不用指定，有系统自动分配一个端口号和自身的ip地址组合。这就是为什么通常服务器端在listen之前会调用bind()，而客户端就不会调用，而是在connect()时由系统随机生成一个。

2.3 listen()、connect()

接口定义：

```
int listen(int sockfd, int backlog);
int connect(int sockfd, const struct sockaddr *addr, socklen_t addrlen);
```

如果作为一个服务器，在调用socket()、bind()之后就会调用listen()来监听这个socket，如果客户端这时调用connect()发出连接请求，服务器端就会接收到这个请求。如果使用udp协议，则不需要connect()，直接发送即可。

listen()函数的参数分别是：

- sockfd：即socket描述字，它是通过socket()函数创建了，唯一标识一个socket。
- backlog 最大监听数

connect()函数的参数分别是：

- sockfd：即socket描述字，它是通过socket()函数创建了，唯一标识一个socket。
- addr：连接地址
- addrlen：地址长度

socket()函数创建的socket默认是一个主动类型的，listen()函数将socket变为被动类型的，等待客户的连接请求。

2.4 accept()

接口定义：

```
int accept(int sockfd, struct sockaddr *addr, socklen_t *addrlen);
```

TCP服务器端依次调用socket()、bind()、listen()之后，就会监听指定的socket地址了。TCP客户端依次调用socket()、connect()之后就想TCP服务器发送了一个连接请求。TCP服务器监听到这个请求之后，就会调用accept()函数取接收请求，这样连接就建立好了。之后就可以开始网络I/O操作了，即类同于普通文件的读写I/O操作。

参数：

- sockfd：即socket描述字，它是通过socket()函数创建了，唯一标识一个socket。
- addr：用于返回客户端的协议地址
- addrlen： 地址长度

如果accpet成功，那么其返回值是由内核自动生成的一个全新的描述字，代表与返回客户的

TCP连接。

> *注意：accept的第一个参数为服务器的socket描述字，是服务器开始调用socket()函数生成的，称为监听socket描述字；而accept函数返回的是已连接的socket描述字。一个服务器通常通常仅仅只创建一个监听socket描述字，它在该服务器的生命周期内一直存在。内核为每个由服务器进程接受的客户连接创建了一个已连接socket描述字，当服务器完成了对某个客户的服务，相应的已连接socket描述字就被关闭。*

2.5 read()、write()等

接口定义：

```
ssize_t send(int sockfd, const void *buf, size_t len, int flags);
ssize_t recv(int sockfd, void *buf, size_t len, int flags);

ssize_t sendto(int sockfd, const void *buf, size_t len, int flags,
               const struct sockaddr *dest_addr, socklen_t addrlen);
ssize_t recvfrom(int sockfd, void *buf, size_t len, int flags,
                 struct sockaddr *src_addr, socklen_t *addrlen);

ssize_t sendmsg(int sockfd, const struct msghdr *msg, int flags);
ssize_t recvmsg(int sockfd, struct msghdr *msg, int flags);
```

以上3组方法，可以随意使用，以recv()/send()为例，参数：

- sockfd：即socket描述字，它是通过socket()函数创建了，唯一标识一个socket。
- buf：写入/发送的信息
- len：信息长度
- flags：标示

返回值表示本次已读（写）的长度，如果为0则表示读取（写入）完毕，如果小于0则表示读取（写入）错误

2.6 close()

接口定义：

```
int close(int fd);
```

close一个TCP socket的缺省行为时把该socket标记为以关闭，然后立即返回到调用进程。该描述字不能再由调用进程使用，也就是说不能再作为read或write的第一个参数。

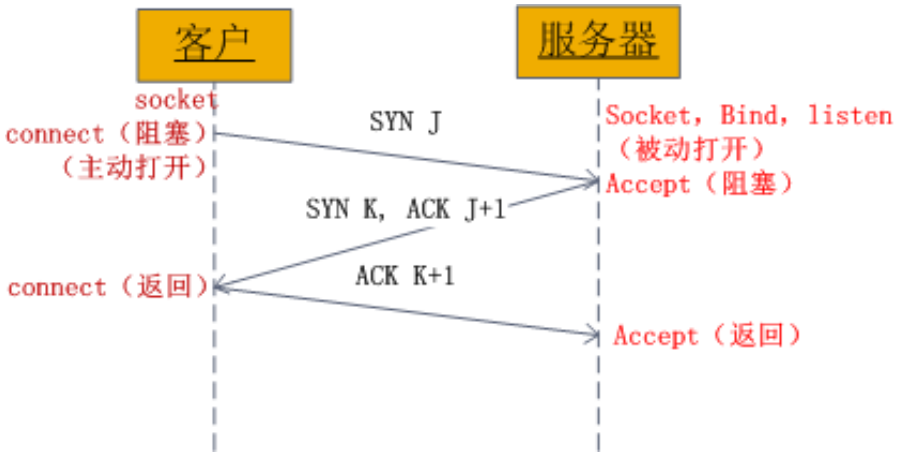
> *注意：close操作只是使相应socket描述字的引用计数-1，只有当引用计数为0的时候，才会触发TCP客户端向服务器发送终止连接请求。*

3. TCP/IP协议

TCP和UDP两种协议的区别在于，TCP需要连接，UDP不需要。TCP消息如果发生错误，会自动重发，UDP不会。所以TCP连接更加稳定，而UDP会更加节约数据量。我们也可以采取自己的方案实现UDP的数据安全。比如QQ的文件传输就是使用的UDP协议，但是同样很安全。而MSN传输文件使用的是TCP协议，所以会慢一些。

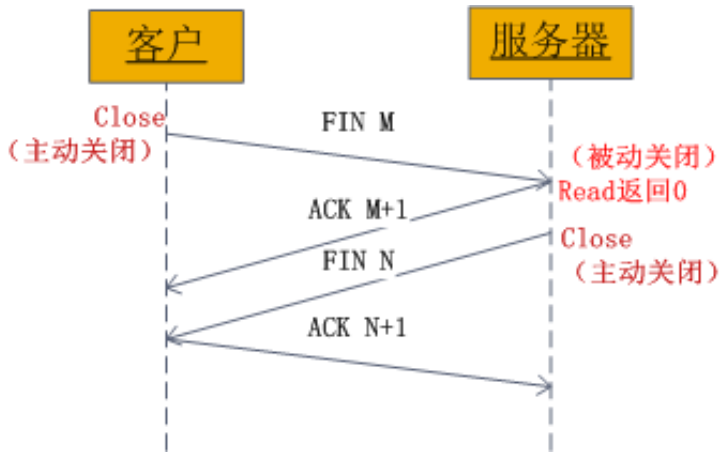
可以想象成两个人在人很多的黑房间内，如果想要交流，需要建立连接。

3.1 TCP协议建立连接的过程俗称为3次握手：



- 客户端向服务器发送一个SYN J
- 服务器向客户端响应一个SYN K，并对SYN J进行确认ACK J+1
- 客户端再想服务器发一个确认ACK K+1

3.2 TCP断开连接需要4次握手：



- 某个应用进程首先调用close主动关闭连接，这时TCP发送一个FIN M；
- 另一端接收到FIN M之后，执行被动关闭，对这个FIN进行确认。它的接收也作为文件结束符传递给应用进程，因为FIN的接收意味着应用进程在相应的连接上再也接收不到额外数据；
- 一段时间之后，接收到文件结束符的应用进程调用close关闭它的socket。这导致它的TCP也发送一个FIN N；
- 接收到这个FIN的源发送端TCP对它进行确认。

4. 代码演练

实现建立TCP服务器和TCP客户端，并且交互信息。具体参照代码。