二、使用Socket发起HTTP请求

本节接上文《Socket通信》

本节目录：

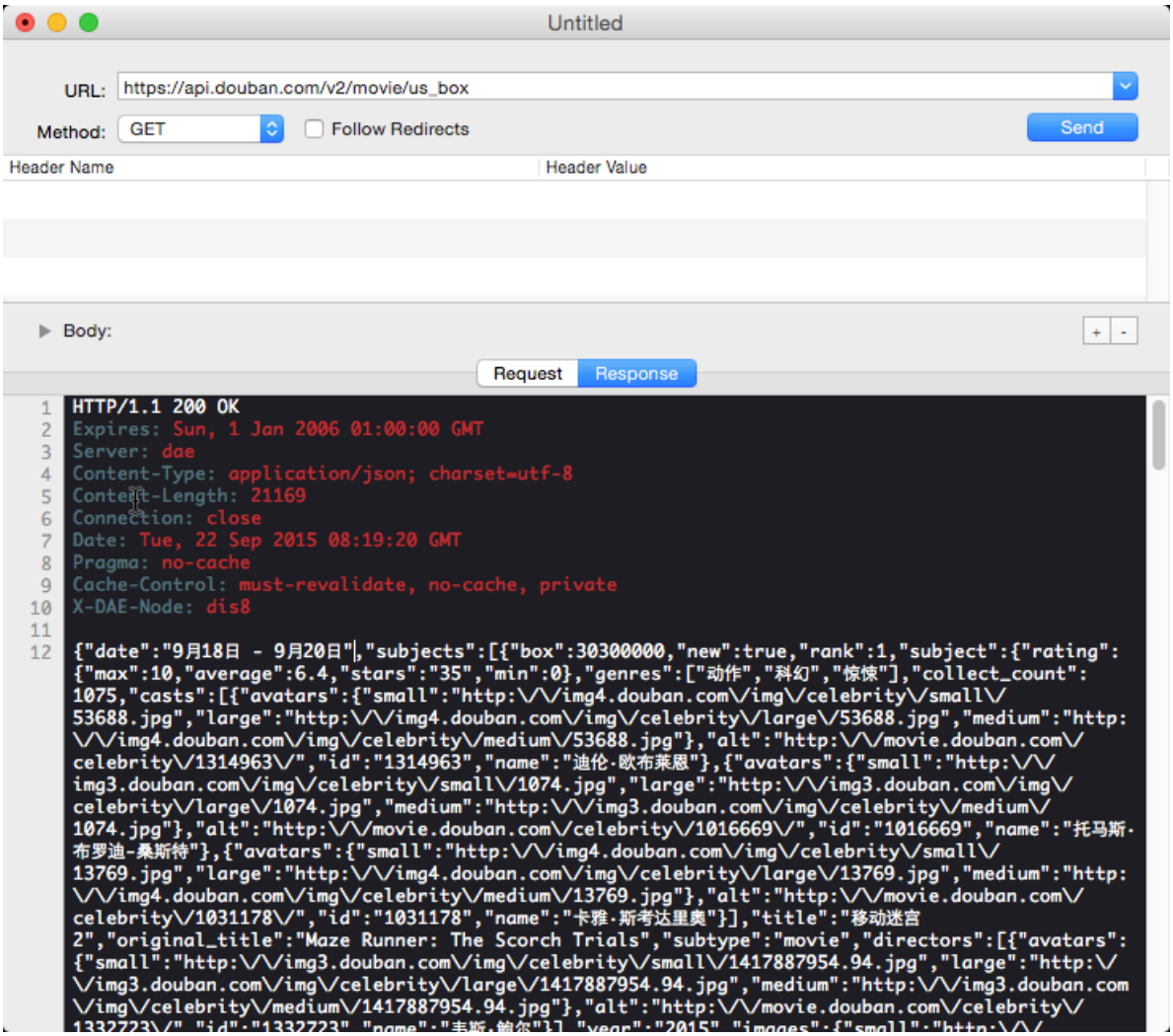
1. HTTP协议概述
2. 使用Socket向服务器发起请求，并接收数据
3. 拼接HTTP GET请求
4. 保存HTTP请求返回的数据

> *上文中提到Socket可以实现基于IP协议的一切请求，本节将学习如何使用Socket发起HTTP请求，并且接收返回数据。本节请求的目标地址是豆瓣开放的API，可以在Safari或者HTTP Client中模拟请求一下。*

API：

```
https://api.douban.com/v2/movie/us_box
```

模拟请求成功后，会收到Json数据:



1. HTTP协议概述

HTTP协议几乎是应用层中应用最为广泛的协议，因为其可扩展性强，对于数据安全也有一定的保证。几乎所有的网站都支持HTTP协议访问。HTTP协议基于TCP/IP协议，通过HTTP协议请求数据，需要建立TCP连接。

步骤如下：

1. 与目标服务器连接
2. 发送请求内容
3. 接收返回的数据
4. 中断连接

每次HTTP请求都是一个新的TCP连接。

HTTP请求有8种请求方式：

1. **OPTIONS** 返回服务器针对特定资源所支持的HTTP请求方法
2. **GET** 向特定的资源发出请求
3. **POST** 向指定资源提交数据进行处理请求（例如提交表单或者上传文件）。数据被包含在请求体中
4. **HEAD** 向服务器索要与GET请求相一致的响应，只不过响应体将不会被返回。
5. **PUT** 向指定资源位置上传其最新内容
6. **DELETE** 请求服务器删除Request-URI所标识的资源
7. **TRACE** 回显服务器收到的请求，主要用于测试或诊断
8. **CONNECT** HTTP/1.1协议中预留给能够将连接改为管道方式的代理服务器。

其中GET和POST请求方式是最为常用请求方式，本文将模拟一个GET请求。

1.1 请求内容

HTTP协议规定：个完整的由客户端发给服务器的HTTP请求中包含以下内容

- 请求行：包含了请求方法、请求资源路径、HTTP协议版本

```
GET /v2/movie/us_box HTTP/1.1
```

- 请求头：包含了对客户端的环境描述、客户端请求的主机地址等信息

```
Host: api.douban.com:8080 // 客户端想访问的服务器主机地址

User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10.9) Firefox/30.0// 客户端的类型，客户端的软件环境

Accept: text/html, */*// 客户端所能接收的数据类型

Accept-Language: zh-cn // 客户端的语言环境

Accept-Encoding: gzip // 客户端支持的数据压缩格式
```

- 请求体：客户端发给服务器的具体数据，比如文件数据

1.2 响应内容

- 状态行：包含了HTTP协议版本、状态码、状态英文名称

```
HTTP/1.1 200 OK
```

状态码详解

- 响应头：包含了对服务器的描述、对返回数据的描述

```
Server: nginx // 服务器的类型

Content-Type: application/json; charset=utf-8// 返回数据的类型

Content-Length: 21670 // 返回数据的长度

Date: Sat, 19 Sep 2015 05:02:56 GMT // 响应的时间
```

- 实体内容：服务器返回给客户端的具体数据，比如文件数据

图解：

2. 构建HTTP请求

2.1 从NSURL解析出服务器IP地址

```
//从URL中获得 主机名和端口号
NSURL *url = [NSURL URLWithString:urlString];
NSString *host = url.host;
NSNumber *port = [url port];


//DNS解析出主机名对应的ip地址
struct hostent * remoteHostEnt = gethostbyname([host UTF8String]);
struct in_addr **addr_list = (struct in_addr **)remoteHostEnt->h_addr_list;
//主机名下对应的所有IP地址
for(int i = 0; addr_list[i] != NULL; i++) {
    printf("%s ", inet_ntoa(*addr_list[i]));
}

//http协议默认端口号
if ([port intValue]==0) {
    port = @80;
}
```

gethostbyname()函数传入URL的host，返回主机地址结构体：

```
struct hostent {
    char    *h_name;    /* official name of host */
    char    **h_aliases;    /* alias list */
    int h_addrtype; /* host address type */
    int h_length;   /* length of address */
    char    **h_addr_list;  /* list of addresses from name server */
};
```

如果NSURL中没有端口号信息，我们就可以指定端口号为80，这是HTTP协议的默认端口。

2.2 从NSURL中解析出HTTP请求头

这里我们默认为GET请求，如果是其他方式需要传入不同方式名，在请求头中表明，一个最基本的请求头如下：

```
GET /v2/movie/us_box HTTP/1.1

Host: api.douban.com

Content-Encoding: utf-8

User-Agent: Mozilla/5.0 (Windows NT 6.1; rv:25.0) Gecko/20100101 Firefox/25.0



Connection: close
```

注意一定要换行，换行符为`\r\n`。

- 第一行分别表示请求方式，请求路径，HTTP协议版本
- 第二行表示主机域名
- 第三行表示编码方式
- 第四行表示客户端浏览器内核版本，服务器可以根据这个参数返回不同数据格式。
- 第五行空白行
- 第六行表示连接完毕后关闭连接。因为HTTP 1.1版本默认对方支持长连接，如果不希望长连接则需要标明。

使用NSString拼接方式如下：

```
+(NSString *)makeRequestMsgWithURL:(NSURL *)url
{
    //   OC拼接方式
    NSString *sendMsg = [NSString stringWithFormat:@"GET %@ HTTP/1.1\r\n",url.path];
    sendMsg =[sendMsg stringByAppendingFormat:@"Host: %@\r\n",url.host];
    sendMsg =[sendMsg stringByAppendingString:@"Content-Encoding: utf-8\r\n"];
    sendMsg =[sendMsg stringByAppendingString:@"User-Agent: Mozilla/5.0 (Windows NT 6.1; rv:25.0)
Gecko/20100101 Firefox/25.0\r\n\r\n"];
    sendMsg =[sendMsg stringByAppendingString:@"Connection: close\r\n"];

    return sendMsg;
}
```

3. Socket实现HTTP请求

如上节所讲，同样需要先建立TCP连接：

```
    sendMsgSocketfd =  socket(AF_INET, SOCK_STREAM, 0);

    struct sockaddr_in  addr;

    const char *ipadd =  [host cStringUsingEncoding:NSUTF8StringEncoding];

    memset(&addr, 0, sizeof(addr));
    addr.sin_family = AF_INET;
    addr.sin_port = htons(port);
    inet_pton(AF_INET, ipadd, &addr.sin_addr);


    if (connect(sendMsgSocketfd, (struct sockaddr*)&addr, sizeof(addr)) == -1) {

        NSLog(@"connect error :%s",strerror(errno));
    }

    NSLog(@" >> Successfully connected to %@:%d",host,port);
```

3.1 发送请求

把拼接好的请求头发送给服务器：

```
    send(sendMsgSocketfd,requestMsg , strlen(requestMsg), 0);
```

3.2 接收返回数据

接收数据时需要使用循环，因为每次接收的长度是有限的，需要持续调用recv()函数,如果recv返回0则标示接收完毕，我们使用NSData保存接收的数据：

```
    NSMutableData * data = [[NSMutableData alloc] init];
    BOOL waitingForData = YES;
    while (waitingForData){

        const char * buffer[4096];
        int length = sizeof(buffer);

        // Read a buffer's amount of data from the socket; the number of bytes read is returned
        long result = recv(sendMsgSocketfd, &buffer, length, 100);
        if (result > 0) {
            [data appendBytes:buffer length:result];

            NSLog(@"===============%s",buffer);
        }
        else {
            // if we didn't get any data, stop the receive loop
            waitingForData = NO;
        }
    }


    block(data);

    close(sendMsgSocketfd);
```

接收完后，关闭连接。

三、 CFNetwork

苹果公司对Socket进行了轻量级的封装，包括 CFNetwork 和 CFNetServices ，由于iOS中大多数情况下作为客户端，下面就重点介绍 CFNetwork。

虽然 CFNetwork 只是对 BSD socket 的进行了轻量级的封装，但在 iOS 中使用 CFNetwork 有一个显著的好处，那就是 CFNetwork 与 run-loop 结合得很好。每一个线程都有自己的 run-loop，因此我们可以 CFNetwork 当中事件源加入到 run-loop 中，这样就可以在线程的 run-loop 中处理网络事件了。大名鼎鼎的 ASIHttpRequest 库就是基于 CFNetwork 封装的，不过很可惜该库不再更新了。

CFNetwork基于CFStream和CFSocket。

1. CFNetwork概述

使用CFNetwork需要引入：

```
CFNetwork.framework
```

CFStream和CFSocket概述：

- CFStream提供了便捷的数据读写接口。注意Stream是单向的，分为读取和写入两种。
- CFSocket是对BSD Socket的封装，支持了Runtime功能，可以使Socket接口不会阻塞所在线程。

CFNetwork中根据不同的应用层协议，实现了很多不同的接口：

- FTP协议接口
- HTTP协议接口 等等

下面我们使用CFNetwork，连接上节课中用BSD Socket创建的服务器，并且接收数据。

2. 接口介绍

2.1 创建Stream

CFNetwork 接口是基于 C 的，下面的接口用于创建一对 socket stream，一个用于读取，一个用于写入：

```
void CFStreamCreatePairWithSocketToHost(CFAllocatorRef alloc, CFStringRef host, UInt32 port,
CFReadStreamRef *readStream, CFWriteStreamRef *writeStream);
```

参数:

1. alloc 内存分配类型，一般为默认的kCFAllocatorDefault
2. host 主机名
3. port 端口号
4. readStream 传入地址，初始化一个读取的Stream
5. writeStream 初始化一个写入的Stream

如果只需要初始化一个Stream，则可以用NULL代替。

示例:

```
NSString * host = [url host];
NSInteger port = [[url port] integerValue];
CFReadStreamRef readStream;
CFStreamCreatePairWithSocketToHost(kCFAllocatorDefault, (__bridge CFStringRef)host, port, &readStream,
NULL);
```

2.2 设置回调函数

上文中提到，CFSocket不会阻塞线程，当然会使用异步的方式回传各个函数的使用结果，我们需要设置回调函数来接收返回结果，设置回调函数接口如下:

```
Boolean CFReadStreamSetClient(CFReadStreamRef stream, CFOptionFlags streamEvents,
CFReadStreamClientCallBack clientCB, CFStreamClientContext *clientContext);

Boolean CFWriteStreamSetClient(CFWriteStreamRef stream, CFOptionFlags streamEvents,
CFWriteStreamClientCallBack clientCB, CFStreamClientContext *clientContext);
```

参数:

1. stream 监听的stream
2. streamEvents 监听的事件类型
3. clientCB CallBack函数
4. clientContext 参数

示例代码:

```
CFStreamClientContext ctx = {0, (__bridge void *)(self), NULL, NULL, NULL};

// Get callbacks for stream data, stream end, and any errors
//
CFOptionFlags registeredEvents = (kCFStreamEventHasBytesAvailable | kCFStreamEventEndEncountered |
kCFStreamEventErrorOccurred);

CFReadStreamSetClient(readStream, registeredEvents, socketCallback, &ctx)
```

回调函数:

```
void socketCallback(CFReadStreamRef stream, CFStreamEventType event, void * myPtr){



}
```

回调函数中，会支持3个参数，分别是监听的stream，事件类别，和我们设置的参数。

2.3 加入Runloop中

上文中提到，CFSocket加入了对Runloop的支持，使Socket的接口不再阻塞当前线程，接口如下：

```
void CFReadStreamScheduleWithRunLoop(CFReadStreamRef stream, CFRunLoopRef runLoop, CFStringRef
runLoopMode);

void CFWriteStreamScheduleWithRunLoop(CFWriteStreamRef stream, CFRunLoopRef runLoop, CFStringRef
runLoopMode);
```

参数：

1. stream 加入Runloop的stream
2. runLoop 加入的run loop
3. runLoopMode 加入Runloop的模式

示例代码：

```
if (CFReadStreamSetClient(readStream, registeredEvents, socketCallback, &ctx)) {
    CFReadStreamScheduleWithRunLoop(readStream, CFRunLoopGetCurrent(), kCFRunLoopCommonModes);

}
```

一般使用时，先判断一下是否设置回调函数成功再加入Runloop中。

2.4 打开Stream

使用Stream需要显式的Open操作，也就是说我们可以创建很多Stream，但是不直接使用，在需要使用时再进行Open操作，接口如下：

```
Boolean CFReadStreamOpen(CFReadStreamRef stream);

Boolean CFWriteStreamOpen(CFWriteStreamRef stream);
```

如果在子线程中，不要忘记Runloop需要我们自己开启，主线程中Runloop默认是开启状态的：

```
    CFRunLoopRun();
```

2.5 连接出错

当Open一个Stream时，相当于做了Connect、SendMsg等操作，这时如果出错我们可以获取到错误信息，代码如下：

```
CFErrorRef error = CFReadStreamCopyError(readStream);
if (error != NULL) {
    if (CFErrorGetCode(error) != 0) {
        NSString * errorInfo = [NSString stringWithFormat:@"Failed to connect stream; error '%@' (code
%ld)", (__bridge NSString*)CFErrorGetDomain(error), CFErrorGetCode(error)];
        [self networkFailedWithErrorMessage:errorInfo];
    }

    CFRelease(error);

    return;
}
```

3. 完整实现接收数据

接口：

```
+(void)requstDataFromHost:(NSString *)host port:(int)port callBack:(DataBlock )block{


    // Keep a reference to self to use for controller callbacks
    //
    CFStreamClientContext ctx = {0, (__bridge void *)(block), NULL, NULL, NULL};

    // Get callbacks for stream data, stream end, and any errors
    //
    CFOptionFlags registeredEvents = (kCFStreamEventHasBytesAvailable | kCFStreamEventEndEncountered |
kCFStreamEventErrorOccurred);

    // Create a read-only socket
    //
    CFReadStreamRef readStream;
    CFStreamCreatePairWithSocketToHost(kCFAllocatorDefault, (__bridge CFStringRef)host, port,
&readStream, NULL);

    // Schedule the stream on the run loop to enable callbacks
    //
    if (CFReadStreamSetClient(readStream, registeredEvents, socketCallback, &ctx)) {
        CFReadStreamScheduleWithRunLoop(readStream, CFRunLoopGetCurrent(), kCFRunLoopCommonModes);

    }
    else {
//        [self networkFailedWithErrorMessage:@"Failed to assign callback method"];
        return;
    }

    // Open the stream for reading
    //
    if (CFReadStreamOpen(readStream) == NO) {
//        [self networkFailedWithErrorMessage:@"Failed to open read stream"];

        return;
    }

    CFErrorRef error = CFReadStreamCopyError(readStream);
    if (error != NULL) {
        if (CFErrorGetCode(error) != 0) {
            NSString * errorInfo = [NSString stringWithFormat:@"Failed to connect stream; error '%@'
(code %ld)", (__bridge NSString*)CFErrorGetDomain(error), CFErrorGetCode(error)];
//            [self networkFailedWithErrorMessage:errorInfo];
            NSLog(@"%@",errorInfo);
        }

        CFRelease(error);

        return;
    }

    NSLog(@"Successfully connected to %@", host);


    // Start processing
    //
    CFRunLoopRun();




}
```

回调函数:

```
void socketCallback(CFReadStreamRef stream, CFStreamEventType event, void * myPtr)
{
    NSLog(@" >> socketCallback in Thread %@", [NSThread currentThread]);

    DataBlock block = (__bridge DataBlock )myPtr;

    switch(event) {
        case kCFStreamEventHasBytesAvailable: {
            // Read bytes until there are no more
            //
            while (CFReadStreamHasBytesAvailable(stream)) {
                char *buffer = malloc(kBufferSize);

                int numBytesRead = CFReadStreamRead(stream, buffer, kBufferSize);

//                [controller didReceiveData:[NSData dataWithBytes:buffer length:numBytesRead]];

                NSLog(@"recv buffer : %s",buffer);

                block([NSData dataWithBytes:buffer length:numBytesRead],0);

            }

            break;
        }

        case kCFStreamEventErrorOccurred: {
            CFErrorRef error = CFReadStreamCopyError(stream);
            if (error != NULL) {
                if (CFErrorGetCode(error) != 0) {
                    NSString * errorInfo = [NSString stringWithFormat:@"Failed while reading stream;
error '%@' (code %ld)", (__bridge NSString*)CFErrorGetDomain(error), CFErrorGetCode(error)];

                    NSLog(@"error %@",errorInfo);

                }

                CFRelease(error);
            }


            break;
        }

        case kCFStreamEventEndEncountered:
            // Finnish receiveing data
            //
            block([NSData data],1);

            // Clean up
            //
            CFReadStreamClose(stream);
            CFReadStreamUnscheduleFromRunLoop(stream, CFRunLoopGetCurrent(), kCFRunLoopCommonModes);
            CFRunLoopStop(CFRunLoopGetCurrent());

            break;

        default:
            break;
    }
}
```

4. CFHTTP

在iOS9中 CFHTTP相关函数已经不能够使用了，原接口使用方法如下：

```
+(CFReadStreamRef )httpStreamWithURL:(NSURL *)url{



    // 创建请求
//    CFStringRef url = CFSTR("api.douban.com");
//    CFURLRef myURL = CFURLCreateWithString(kCFAllocatorDefault, url, NULL);// note: release
    CFStringRef requestMethod = CFSTR("GET");

    //请求信息
    //    GET /v2/movie/us_box HTTP/1.1

    //    Host: api.douban.com


    CFHTTPMessageRef myRequest = CFHTTPMessageCreateRequest(kCFAllocatorDefault, requestMethod,
(__bridge CFURLRef)url, kCFHTTPVersion1_1);// note: release

    // POST请求设置请求提 body
//    const UInt8 bytes[] = "12345";
//    CFDataRef bodyData = CFDataCreate(kCFAllocatorDefault, bytes, 5);// note: release
//    CFHTTPMessageSetBody(myRequest, bodyData);

    // 设置请求头header
//    CFStringRef headerField = CFSTR("name");
//    CFStringRef value = CFSTR("daniate");
//    CFHTTPMessageSetHeaderFieldValue(myRequest, headerField, value);
    CFReadStreamRef requestReadStream = CFReadStreamCreateForHTTPRequest(kCFAllocatorDefault,
myRequest);// note: release



    return requestReadStream;
}
```

所以目前想要使用CF接口实现HTTP协议请求需要使用两个Stream，这里就不做延伸，替代方式苹果已经给出：

```
CFReadStreamCreateForHTTPRequest(CFAllocatorRef __nullable alloc, CFHTTPMessageRef request)
CF_DEPRECATED(10_2, 10_11, 2_0, 9_0, "Use NSURLSession API for http requests");
```

没错，就是NSURLSession。下节中，我们来学习NSURLSession，看苹果给我们提供的最高层网络封装库。