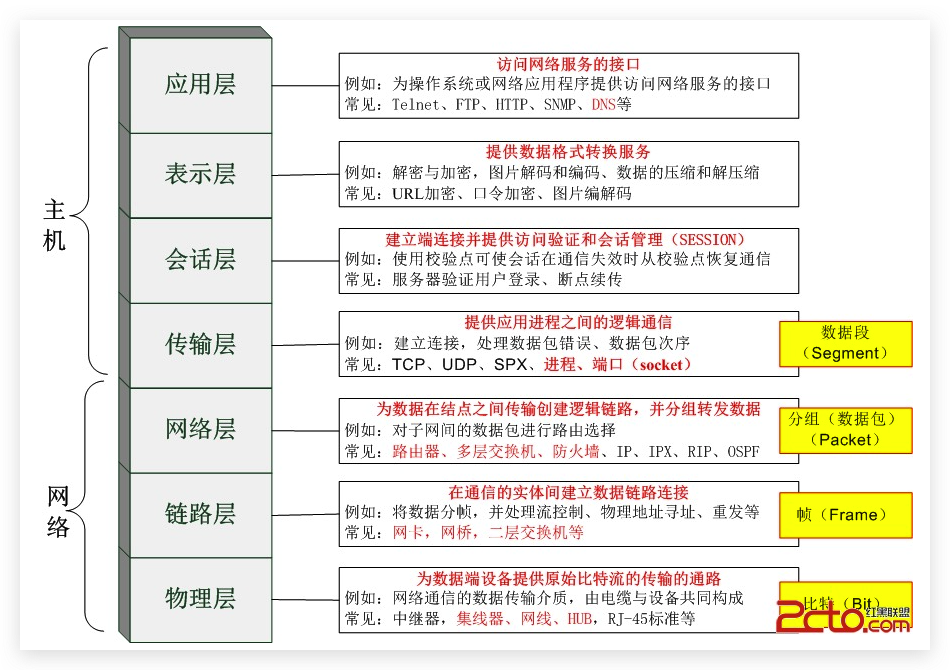
iOS网络数据请求

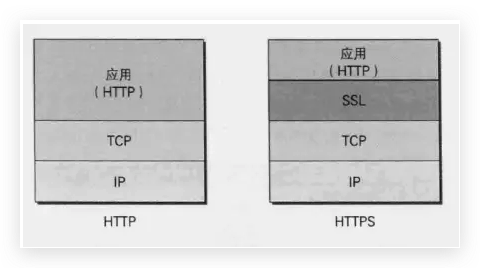
参考资料：https://macdev.io/ebook/HTTPNetwork.html

先从底层讲起，由浅入深：



# 1、HTTP是应用层协议，定义的是传输数据的内容的规范。

# 2、Http 和 Https：



## 2.1、TCP：只是单纯建立连接

## 2.2、SSL = Https - http

## 2.3、注意：

2.3.1、TCP/IP是Internet协议族，而不单单是TCP和IP；

2.3.2、[HTTP](https://zh.wikipedia.org/wiki/HTTP)的[URL](https://zh.wikipedia.org/wiki/%E7%BB%9F%E4%B8%80%E8%B5%84%E6%BA%90%E5%AE%9A%E4%BD%8D%E7%AC%A6)是由“http://”起始与默认使用[端口](https://zh.wikipedia.org/wiki/TCP/UDP%E7%AB%AF%E5%8F%A3%E5%88%97%E8%A1%A8)80，而HTTPS的URL则是由“[https://”起始与默认使用端口443](https://xn--443-6o0a031zj2b2umsvmm53ducmey8bbuetu2e)；

2.3.3、http/https 是无状态的，需要有状态衍生出cookie 和 Session来对用户状态进行记录，二次上报的时候可以带状态请求；

## 2.4、SSL原理：

2.4.1、最终是对称加密完成数据的传输（非对称加密非常损耗性能）

2.4.2、TCP 3次握手

## 2.5、http的协议方法（“8大件”）：

2.5.1、get/head/post/put/delete/trace/options/connect

2.5.2、get/post方法能满足一般的常规需求，客户端的请求协议方式必须和服务器配置相吻合（协议可以理解为公开的一个加解密的一个方式）

意味着：一个好的第三方网络请求框架最好能兼容“8大件”，虽然只用get/post方法,为了强调兼容性，避免服务器使用一些不常用的协议，而不配合兼容get/post方法,比如有些上载大文件用put方法;

附：服务器状态码可以自定义，但是一般用公认标准：

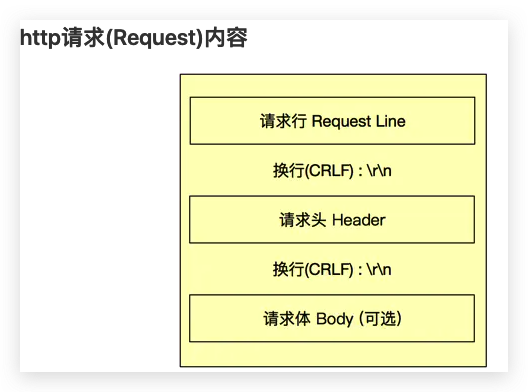
[1xx消息](https://zh.wikipedia.org/wiki/HTTP%E7%8A%B6%E6%80%81%E7%A0%81" \l "1xx%E6%B6%88%E6%81%AF)——请求已被服务器接收，继续处理

[2xx成功](https://zh.wikipedia.org/wiki/HTTP%E7%8A%B6%E6%80%81%E7%A0%81" \l "2xx%E6%88%90%E5%8A%9F)——请求已成功被服务器接收、理解、并接受

[3xx重定向](https://zh.wikipedia.org/wiki/HTTP%E7%8A%B6%E6%80%81%E7%A0%81" \l "3xx%E9%87%8D%E5%AE%9A%E5%90%91)——需要后续操作才能完成这一请求

[4xx请求错误](https://zh.wikipedia.org/wiki/HTTP%E7%8A%B6%E6%80%81%E7%A0%81" \l "4xx%E8%AF%B7%E6%B1%82%E9%94%99%E8%AF%AF)——请求含有词法错误或者无法被执行

[5xx服务器错误](https://zh.wikipedia.org/wiki/HTTP%E7%8A%B6%E6%80%81%E7%A0%81" \l "5xx%E6%9C%8D%E5%8A%A1%E5%99%A8%E9%94%99%E8%AF%AF)——服务器在处理某个正确请求时发生错误



3、请求头：

3.1、通用头字段(General Header Fields)

3.2、请求头字段(Request Header Fields)

3.3、响应头字段(Response Header Fields)

3.4、实体头字段(Entity Header Fields)

请求参数可以在请求头（Header）里面，也可以在 请求体（Body）里面。主要看服务器的解析规则。

意味着，一个好的网络处理框架需要能自定义请求头（Header）

4、相关时间线：

主要是围绕NSURLConnection

**iOS7和OSX10.9 是个分界点**

新增接口NSURLSessionTask

5、核心类描述：(重点)

## 5.1、NSURLConnection:

5.1.1、设置HTTP请求相关的各种参数(包括设置缓存数据的规则) ——> NSURLRequest对象；

5.1.2、如果需要Cookie：设置NSURLHTTPCookieStorage，设置完成后对当前应用中后续所有的请求都有效，是全局的；

5.1.3、请求类型：

5.1.3.1、同步请求：则等待数据返回后结束任务

5.1.3.2、异步请求：创建NSURLConnection对象，设置它的代理，发起网络请求

5.1.4、根据需要实现相关的代理方法

5.1.4.1、处理HTTP 的请求响应NSHTTPURLResponse

5.1.4.2、对接收到的数据可以在前面第一步中的缓存设置规则上做额外的处理

5.1.4.3、如果是收控的资源的访问，则提供验证的方式

5.1.4.4、接收数据

**值得注意：NSURLDownload和NSURLDownloadDelegate仅在OS X系统支持,iOS不支持。**

5.2、NSURLSessionTask

5.2.1、任务分类

5.2.1.1、简单的数据请求任务；

5.2.1.2、上传任务；

5.2.1.3、下载任务；

5.2.1.4、流处理任务；

5.2.2、关于NSURLSession

NSURLSessionTask <— NSURLSession（http是无状态的，Session 是cookie的升级版，用于记录状态）

5.2.2.1、默认的sessions:类似于NSURLConnection的行为，将缓存和Cookie存储到硬盘，证书存储到钥匙串keyChain中；

5.2.2.2、临时sessions:缓存数据,Cookie,证书等信息全部存储在内存中，当 session失效后自动清除；

5.2.2.3、后台sessions:类似于默认的sessions，不同的是当应用挂起在后台时能继续运行任务这个配置同时可以对Cookie,Cache，认证方式做统一的配置；

【可以简单的理解NSURLSession为一组相同配置类型的任务队列,Session中可以创建多个任务去并发异步的执行】

## 5.3、基于NSURLSession的流程图：

5.3.1、简单的数据请求

5.3.1.1、 NSURLSession <— NSURLSessionConfiguration

5.3.1.2、NSMutableURLRequest <— NSURL，配置NSMutableURLRequest相关参数（比如HTTPMethod、 forHTTPHeaderField、HTTPBody…）

request.HTTPBody = postData;

5.3.1.3、NSURLSessionTask <—【NSMutableURLRequest + NSURLSession】（请求的Body中设置POST的Data）

  NSURLSessionTask \*task =  [session dataTaskWithRequest:request...

5.3.1.4、[NSURLSessionTask resume];

5.3.2、POST方式传递数据

5.3.2.1、NSURLSession <— NSURLSessionConfiguration

5.3.2.2、NSMutableURLRequest <— NSURL，配置NSMutableURLRequest相关参数（比如HTTPMethod、forHTTPHeaderField、HTTPBody…）

5.3.2.3、NSURLSessionUploadTask <— NSMutableURLRequest

  NSURLSessionUploadTask \*uploadTask = [session uploadTaskWithRequest:request fromData:postData

5.3.2.4、[NSURLSessionUploadTask resume];

5.3.3、文件下载

文件下载过程中会边下载边写入临时文件，下载完成后需要copy到指定的地方，因此在下载过程中对内存的占用不会随着文件增大而无限增大，因此使用它可以来进行大文件的下载.

5.3.3.1、 NSURLSession <— NSURLSessionConfiguration

5.3.3.2、NSURLSessionDownloadTask <—  NSURLSession + NSURL

5.3.3.3、[NSURLSessionDownloadTask resume];

5.3.4、文件上传

5.3.4.1、流式文件上传

其中设置Content-Length和Content-Type的代码可以省略，系统会默认设置。

5.3.4.1.1、NSMutableURLRequest <— NSURL（上传终点 URLWithString）

配置NSMutableURLRequest相关参数（比如HTTPMethod、forHTTPHeaderField…）

5.3.4.1.2、NSURL（上传起点 fileURLWithPath）

5.3.4.1.3、NSMutableURLRequest <— NSURL（上传终点）

5.3.4.1.4、NSURLSession <— NSURLSessionConfiguration

5.3.4.1.5、NSURLSessionUploadTask <— NSURLSession + NSMutableURLRequest + NSURL（上传起点）

5.3.4.1.6、[NSURLSessionUploadTask resume];

5.3.4.1.7、上传进度:

-(void)URLSession:(NSURLSession \*)session

              task:(NSURLSessionTask \*)task

   didSendBodyData:(int64\_t)bytesSent

    totalBytesSent:(int64\_t)totalBytesSent

totalBytesExpectedToSend:(int64\_t)totalBytesExpectedToSend；

5.3.4.2、Form表单文件上传

5.3.4.2.1、NSURLSession <— NSURLSessionConfiguration;

5.3.4.2.2、NSURL（上传终点），NSData（从路径获取到文件再转成NSData格式）;

5.3.4.2.3、NSMutableData（dataSend）装数据，例：

[dataSend appendData:[[NSString stringWithFormat:@"--%@\r\n", boundary] dataUsingEncoding:NSUTF8StringEncoding]];

[dataSend appendData:[[NSString stringWithFormat:@"Content-Disposition: form-data; name=\"file\"; filename=\"%@\"\r\n", fileName] dataUsingEncoding:NSUTF8StringEncoding]];

[dataSend appendData:[@"Content-Type: application/octet-stream\r\n\r\n" dataUsingEncoding:NSUTF8StringEncoding]];

[dataSend appendData:fileData];

[dataSend appendData:[@"\r\n" dataUsingEncoding:NSUTF8StringEncoding]];

[dataSend appendData:[[NSString stringWithFormat:@"--%@--\r\n\r\n", boundary] dataUsingEncoding:NSUTF8StringEncoding]];

5.3.4.2.4、NSMutableURLRequest <— NSURL（上传终点）;

配置NSMutableURLRequest相关参数（比如HTTPMethod、forHTTPHeaderField、HTTPBody…）;

5.3.4.2.5、NSURLSessionUploadTask <—— NSURLSession + NSMutableURLRequest + NSMutableData（dataSend）;

5.3.4.2.6、[NSURLSessionUploadTask resume];

# 6、缓存:

## 6.1.缓存策略:NSURLRequestCachePolicy定义了4种缓存策略

6.1.1、NSURLRequestUseProtocolCachePolicy:协议规定的缓存策略【系统默认的缓存策略】；

6.1.2、NSURLRequestReloadIgnoringLocalCacheData:不使用缓存，永远使用网络数据；

6.1.3、NSURLRequestReturnCacheDataElseLoad:使用缓存的数据，不管数据事否过期;如果缓存没有数据才请求网络数据；

NSURLRequestReturnCacheDataDontLoad:仅仅使用缓存数据；

6.2、使用缓存：

6.2.1、NSURLSession中使用缓存，调用协议：

-(void)URLSession:(NSURLSession \*)session dataTask:(NSURLSessionDataTask \*)dataTask

 willCacheResponse:(NSCachedURLResponse \*)proposedResponse

 completionHandler:(void (^)(NSCachedURLResponse \* \_\_nullable cachedResponse))completionHandler;

6.2.2、NSURLConnection 中使用缓存,调用协议：

-(NSCachedURLResponse \*)connection:(NSURLConnection \*)connection

                 willCacheResponse:(NSCachedURLResponse \*)cachedResponse;

6.3、缓存大小设置：

//设置内存缓存2M,硬盘缓存10M,路径为系统默认

NSURLCache \*URLCache = [[NSURLCache alloc] initWithMemoryCapacity:2 \* 1024 \* 1024 diskCapacity:10 \* 1024 \* 1024 diskPath:nil];

[NSURLCache setSharedURLCache:URLCache];

# 7、Cookie：

## 7.1、Cookie的基本常识:

7.1.1、Cookie最大字节长度不能超过4096k；

7.1.2、每个Cookie有多个属性，其中最主要的几个属性为名称name,值value,超期时间expires,域domain,路径path,浏览器自动保存这些Cookie信息；

7.1.3、当从浏览器中访问任何资源(页面,图片,文件等)时，浏览器首先从所有存储的Cookie中通过Domain,Path去匹配，如果匹配成功并且Cookie在有限期内没有过期，就会添加到请求头Header中去；

7.1.4、Cookie是Web 服务器和浏览器之间传递的key-value形式的文本数据信息；

7.1.5、基本流程是客户端请求url访问Web 服务器时，Web 服务器会在响应的消息头中Set-Cookie字段中附带Cookie信息。客户端会保留这个Cookie信息，后续客户端请求Web 服务器时会在请求头Header中附加保留的Cookie信息;

7.2、Cookie 编程:

NSHTTPCookie用来描述定义Cookie的属性，NSHTTPCookieStorage是一个单例对象，提供增加删除Cookie 的接口，用来对NSHTTPCookie实例进行管理。

7.3、Cookie的一些使用场景:

7.3.1、简化API接口参数设计：

多个接口有公共参数时，发起HTTP请求前可以将固定不变的参数通过NSHTTPCookieStorage存储到Cookie中，这样每个请求中Header中会自动填充Cookie字段，服务器可以从Cookie中获取这些信息。

7.3.2、与H5 Web页面无缝集成：

客户端和H5页面都需要登录访问的页面，如果使用Native完成了登录，可以将登录的用户Session或token认证信息通过Cookie存储，Web 服务器检查如果Cookie中有登录信息，后续访问Web页面就可以实现自动登录了(无需再次弹出Web登录页面验证)。

# 断点续传:

8.1、开始下载任务后，可以使用NSURLSessionDownloadTask的cancelByProducingResumeData方法来停止下载任务;

8.2、同时其回调的函数会返回resumeData；

8.3、将这个resumeData保存，甚至可以将resumeData持久化存储，后续恢复下载时可以重新使用resumeData创建新的下载任务;

# 后台任务处理:

在iOS中可以创建后台下载任务，当应用切换到后台时下载任务可以继续进行。可能同时存在多个后台任务，因此每个后台任务创建时有独立的标识identifier。