（一）ASIHttpRequest 库简介、配置和安装

发布者： Seven's - 2011/10/12 - 分类：ASIHTTPRequest 中文文档

使用 ASIHTTPRequest 可以很方便的进行以下操作：

同步/异步方式下载数据
定义下载队列，让队列中的任务按指定的并发数来下载（队列下载必须是异步的)
提交表单，文件上传
处理 cookie
设置代理
上下载进度条
重定向处理
请求与响应的 GZIP
验证与授权

等等，只要跟 HTTP 有关，只有你想不到的，没有她做不到的~

配置方法：

ASIHTTPRequestConfig.h
ASIHTTPRequestDelegate.h
ASIProgressDelegate.h
ASICacheDelegate.h
ASIHTTPRequest.h
ASIHTTPRequest.m
ASIDataCompressor.h
ASIDataCompressor.m
ASIDataDecompressor.h
ASIDataDecompressor.m
ASIFormDataRequest.h
ASIInputStream.h
ASIInputStream.m
ASIFormDataRequest.m
ASINetworkQueue.h
ASINetworkQueue.m
ASIDownloadCache.h
ASIDownloadCache.m

iPhone 工程还需要:

ASIAuthenticationDialog.h
ASIAuthenticationDialog.m
Reachability.h (在 External/Reachability 目录下)

Reachability.m (在 External/Reachability 目录下)

库引用：

CFNetwork.framework
SystemConfiguration.framework
MobileCoreServices.framework
CoreGraphics.framework
和 libz.dylib

另外，还需要 libxml2.dylib(libxml2 还需要设置连接选项-lxml2 和头文件搜索路径 /usr/include/libxml2)

（二）ASIHttpRequest-创建和执行 request

发布者： Seven's - 2011/10/12 - 分类：ASIHTTPRequest 中文文档

同步请求

同步请求会在当前线程中执行，使用 error 属性来检查结束状态（要下载大文件，则需要设定 downloadDestinationPath 来保存文件到本地）：

```
- (IBAction)grabURL:(id)sender
{
   NSURL *url = [NSURL URLWithString:@"http://www.dreamingwish.com"];
   ASIHTTPRequest *request = [ASIHTTPRequest requestWithURL:url];
   [request startSynchronous];
   NSError *error = [request error];
   if (!error) {
      NSString *response = [request responseString];
   }
}
```

同步请求会阻塞主线程的执行，这导致用户界面不响应用户操作，任何动画都会停止渲染。

异步请求

下面是最简单的异步请求方法，这个 request 会在全局的 NSOperationQueue 中执行，若要进行更复杂的操作，我们需要自己创建 NSOperationQueue 或者 ASINetworkQueue，后面会讲到。

```
- (IBAction)grabURLInBackground:(id)sender
{
    NSURL *url = [NSURL URLWithString:@"http://www.dreamingwish.com"];
    ASIHTTPRequest *request = [ASIHTTPRequest requestWithURL:url];
    [request setDelegate:self];
    [request startAsynchronous];
}

- (void)requestFinished:(ASIHTTPRequest *)request
{
    // Use when fetching text data
    NSString *responseString = [request responseString];

    // Use when fetching binary data
    NSData *responseData = [request responseData];
}

- (void)requestFailed:(ASIHTTPRequest *)request
{
    NSError *error = [request error];
}
```

使用 block

在平台支持情况下，ASIHTTPRequest1.8 以上支持 block。

```
- (IBAction)grabURLInBackground:(id)sender
{
    NSURL *url = [NSURL URLWithString:@"http://allseeing-i.com"];
    __block ASIHTTPRequest *request = [ASIHTTPRequest requestWithURL:url];
    [request setCompletionBlock:^{
        // Use when fetching text data
        NSString *responseString = [request responseString];

        // Use when fetching binary data
        NSData *responseData = [request responseData];
    }];
    [request setFailedBlock:^{
        NSError *error = [request error];
    }];
    [request startAsynchronous];
}
```

注意，声明 request 时要使用__block 修饰符，这是为了告诉 block 不要 retain request，以免出现 retain 循环，因为 request 是会 retain block 的。

使用队列

创建 NSOperationQueue 或者 ASINetworkQueue 队列，我们还可以设定最大并发连接数：maxConcurrentOperationCount

```
- (IBAction)grabURLInTheBackground:(id)sender
{
    if (![self queue]) {
        [self setQueue:[[[NSOperationQueue alloc] init] autorelease]];
        [self queue].maxConcurrentOperationCount = 4;
    }

    NSURL *url = [NSURL URLWithString:@"http://www.dreamingwish.com"];
    ASIHTTPRequest *request = [ASIHTTPRequest requestWithURL:url];
    [request setDelegate:self];
    [request setDidFinishSelector:@selector(requestDone:)];
    [request setDidFailSelector:@selector(requestWentWrong:)];
    [[self queue] addOperation:request]; //queue is an NSOperationQueue
}

- (void)requestDone:(ASIHTTPRequest *)request
{
    NSString *response = [request responseString];
}

- (void)requestWentWrong:(ASIHTTPRequest *)request
{
    NSError *error = [request error];
}
```

如果不设定 selector，那么系统会使用默认的 requestFinished: 和 requestFailed:方法

如果需要对队列里面的每个 request 进行区分，那么可以设定 request 的 userInfo 属性，它是个 NSDictionary，或者更简单的方法是设定每个 request 的 tag 属性，这两个属性都不会被发送到服务器。

不要使用 request 的 URL 来区分每个 request，因为 URL 可能会改变（例如重定向），如果需要使用 request 的 URL，使用[request originalURL]，这个将永远返回第一个 url。

对于 ASINetworkQueue

ASINetworkQueue 是 NSOperationQueue 的子类，提供更高级的特性（ASINetworkQueue 的代理函数）：

requestDidStartSelector
当一个 request 开始执行时，这个代理函数会被调用。
requestDidReceiveResponseHeadersSelector
当队列中的 request 收到服务器返回的头信息时，这个代理函数会被调用。对于下载很大的文件，这个通常比整个 request 的完成要早。
requestDidFinishSelector
当每个 request 完成时，这个代理函数会被调用。
requestDidFailSelector
当每个 request 失败时，这个代理函数会被调用。
queueDidFinishSelector
当队列完成（无论 request 失败还是成功）时，这个代理函数会被调用。

ASINetworkQueues 与 NSOperationQueues 稍有不同，加入队列的 request 不会立即开始执行。如果队列打开了进度开关，那么队列开始时，会先对所有 GET 型 request 进行一次 HEAD 请求，获得总下载大小，然后真正的 request 才被执行。

向一个已经开始进行的 ASINetworkQueue 加入 request 会怎样？

如果你使用 ASINetworkQueue 来跟踪若干 request 的进度，只有当新的 request 开始执行时，总进度才会进行自适应调整（向后移动）。ASINetworkQueue 不会为队列开始后才加入的 request 进行 HEAD 请求，所以如果你一次向一个正在执行的队列加入很多 request，那么总进度不会立即被更新。

如果队列已经开始了，不需要再次调用[queue go]。

当 ASINetworkQueue 中的一个 request 失败时，默认情况下，ASINetworkQueue 会取消所有其他的 request。要禁用这个特性，设置 [queue setShouldCancelAllRequestsOnFailure:NO]。

ASINetworkQueues 只可以执行 ASIHTTPRequest 操作，二不可以用于通用操作。试图加入一个不是 ASIHTTPRequest 的 NSOperation 将会导致抛出错误。

取消异步请求

取消一个异步请求（无论 request 是由[request startAsynchronous]开始的还是从你创建的队列中开始的），使用[request cancel]即可。注意同步请求不可以被取消。

注意，如果你取消了一个 request，那么这个 request 将会被视为请求失败，并且 request 的代理或者队列的代理的失败代理函数将被调用。如果你不想让代理函数被调用，那么将 delegate 设置为 nil，或者使用 clearDelegatesAndCancel 方法来取消 request。

clearDelegatesAndCancel 将会首先清除所有的代理和 block。

当使用 ASINetworkQueue 时，如果取消了队列中的一个 request，那么队列中其他所有 request 都会被取消，可以设置 shouldCancelAllRequestsOnFailure 的值为 NO 来避免这个现象。

安全地控制 delegate 防止 request 完成之前代理被释放

request 并不 retain 它们的代理，所以有可能你已经释放了代理，而之后 request 完成了，这将会引起崩溃。大多数情况下，如果你的代理即将被释放，你一定也希望取消所有 request，因为你已经不再关心它们的返回情况了。如此做：

```
// 代理类的 dealloc 函数
- (void)dealloc
{
    [request clearDelegatesAndCancel];
    [request release];
    ...
    [super dealloc];
}
```

（三）ASIHttpRequest-发送数据

发布者： Seven's - 2011/10/12 - 分类：ASIHTTPRequest 中文文档

设定 request 头

```
ASIHTTPRequest *request = [ASIHTTPRequest requestWithURL:url];
[request addRequestHeader:@"Referer" value:@"http://www.dreamingwish.com/"];
```

使用 ASIFormDataRequest POST 表单

通常数据是以’application/x-www-form-urlencoded’格式发送的，如果上传了二进制数据或者文件，那么格式将自动变为‘multipart/form-data’

文件中的数据是需要时才从磁盘加载，所以只要 web server 能处理，那么上传大文件是没有问题的。

```
ASIFormDataRequest *request = [ASIFormDataRequest requestWithURL:url];
[request setPostValue:@"Ben" forKey:@"first_name"];
```

```
[request setPostValue:@"Copsey" forKey:@"last_name"];
[request setFile:@"/Users/ben/Desktop/ben.jpg" forKey:@"photo"];
```

数据的 mime 头是自动判定的，但是如果你想自定义 mime 头，那么这样：

```
ASIFormDataRequest *request = [ASIFormDataRequest requestWithURL:url];

// Upload a file on disk
[request setFile:@"/Users/ben/Desktop/ben.jpg" withFileName:@"myphoto.jpg"
andContentType:@"image/jpeg"
forKey:@"photo"];

// Upload an NSData instance
[request setData:imageData withFileName:@"myphoto.jpg"
andContentType:@"image/jpeg" forKey:@"photo"];
```

你可以使用 addPostValue 方法来发送相同 name 的多个数据（梦维：服务端会以数组方式呈现）：

```
ASIFormDataRequest *request = [ASIFormDataRequest requestWithURL:url];
[request addPostValue:@"Ben" forKey:@"names"];
[request addPostValue:@"George" forKey:@"names"];
[request addFile:@"/Users/ben/Desktop/ben.jpg" forKey:@"photos"];
[request addData:imageData withFileName:@"george.jpg"
andContentType:@"image/jpeg" forKey:@"photos"];
```

PUT 请求、自定义 POST 请求

如果你想发送 PUT 请求，或者你想自定义 POST 请求，使用 appendPostData: 或者 appendPostDataFromFile:

```
ASIHTTPRequest *request = [ASIHTTPRequest requestWithURL:url];
[request appendPostData:[@"This is my data"
dataUsingEncoding:NSUTF8StringEncoding]];
// Default becomes POST when you use appendPostData: / appendPostDataFromFile: /
setPostBody:
[request setRequestMethod:@"PUT"];
```

（四）ASIHTTPRequest-下载数据

发布者： Seven's - 2011/10/16 - 分类：ASIHTTPRequest 中文文档

将服务器响应数据直接下载到文件

如果你请求的资源很大，你可以直接将数据下载到文件中来节省内存。此时，ASIHTTPRequest 将不会一次将返回数据全部保持在内存中。

```
ASIHTTPRequest *request = [ASIHTTPRequest requestWithURL:url];
[request setDownloadDestinationPath:@"/Users/ben/Desktop/my_file.txt"];
```

当我们把数据下载到 downloadDestinationPath 时，数据将首先被存在临时文件中。此时文件的路径名存储在 temporaryFileDownloadPath 中(梦维：如果不设置这个值，会自动生成一个文件名，在模拟器中，文件被创建在$TMPDIR 中)。当 request 完成时，会发生下面两件事之一：

如果数据是被压缩过（gzip）的，那么这个压缩过的文件将被解压到 downloadDestinationPath，临时文件会被删除。

如果数据未被压缩，那么这个文件将被移动到 downloadDestinationPath，冲突解决方式是：覆盖已存在的文件。

注意，如果服务器响应数据为空，那么文件是不会被创建的。如果你的返回数据可能为空，那么你应该先检查下载文件是否存在，再对文件进行操作。

处理收到的服务器响应数据

如果你想处理服务器响应的数据（例如，你想使用流解析器对正在下载的数据流进行处理），你应该实现代理函数 request:didReceiveData:。注意如果你这么做了，ASIHTTPRequest 将不会填充 responseData 到内存，也不会将数据写入文件（downloadDestinationPath ）——你必须自己搞定这两件事（之一）。

获取 HTTP 状态码

ASIHTTPRequest 并不对 HTTP 状态码做任何处理（除了重定向和授权状态码，下面会介绍到），所以你必须自己检查状态值并正确处理。

```
ASIHTTPRequest *request = [ASIHTTPRequest requestWithURL:url];
[request startSynchronous];
int statusCode = [request responseStatusCode];
NSString *statusMessage = [request responseStatusMessage];
```

读取响应头

```
ASIHTTPRequest *request = [ASIHTTPRequest requestWithURL:url];
[request startSynchronous];
NSString *poweredBy = [[request responseHeaders] objectForKey:@"X-Powered-By"];
NSString *contentType = [[request responseHeaders] objectForKey:@"Content-Type"];
```

处理文本编码

ASIHTTPRequest 会试图读取返回数据的编码信息（Content-Type 头信息）。如果它发现了编码信息，它会将编码信息设定为合适的 NSStringEncoding.如果它没有找到编码信息，它会将编码设定为默认编码（NSISOLatin1StringEncoding）。

当你调用[request responseString]，ASIHTTPRequest 会尝试以 responseEncoding 将返回的 Data 转换为 NSString。

处理重定向

当遇到以下 HTTP 状态码之一时，ASIHTTPRequest 会自动重定向到新的 URL：

301 Moved Permanently
302 Found
303 See Other

当发生重定向时，响应数据的值（responseHeaders，responseCookies，responseData，responseString 等等）将会映射为最终地址的相应返回数据。

当 URL 发生循环重定向时，设置在这个 URL 上的 cookie 将被储存到全局域中，并在适当的时候随重定向的请求发送到服务器。

Cookies set on any of the urls encountered during a redirection cycle will be stored in the global cookie store, and will be represented to the server on the redirected request when appropriate.

你可以关闭自动重定向：将 shouldRedirect 设置为 NO。

默认情况下，自动重定向会使用 GET 请求（请求体为空）。这种行为符合大多数浏览器的行为，但是 HTTP spec 规定 301 和 302 重定向必须使用原有方法。

要对 301、302 重定向使用原方法（包含请求体），在发起请求之前，设置 shouldUseRFC2616RedirectBehaviour 为 YES。

（五）ASIHTTPRequest-进度追踪

发布者： Seven's - 2011/10/17 - 分类：ASIHTTPRequest 中文文档

每个 ASIHTTPRequest 有两个 delegate 用来追踪进度：

downloadProgressDelegate 下载)
uploadProgressDelegate (上载).

进度 delegate 可以是 NSProgressIndicators (Mac OS X) 或者 UIProgressViews (iPhone).ASIHTTPRequest 会自适应这两个 class 的行为。你也可以使用自定义 class 作为进度 delegate，只要它响应 setProgress:函数。

如果你执行单个 request，那么你需要为该 request 设定 upload/download 进度 delegate
如果你在进行多个请求，并且你想要追踪整个队列中的进度，你必须使用 ASINetworkQueue 并设置队列的进度 delegate
如果上述两者你想同时拥有，恭喜你，0.97 版以后的 ASIHTTPRequest，这个可以有 ^^

IMPORTANT:如果你向一个要求身份验证的网站上传数据，那么每次授权失败，上传进度条就会被重置为上一次的进度值。因此，当与需要授权的 web 服务器交互时，建议仅当 useSessionPersistence 为 YES 时才使用上传进度条，并且确保你在追踪大量数据的上传进度之前，先使用另外的 request 来进行授权。

追踪小于 128KB 的数据上传进度目前无法做到，而对于大于 128kb 的数据，进度 delegate 不会收到第一个 128kb 数据块的进度信息。这是因为 CFNetwork 库 API 的限制。我们曾向 apple 提交过 bug 报告（bug id 6596016），希望 apple 能修改 CFNetwork 库以便实现上述功能。

2009-6-21：Apple 的哥们儿们真棒！iPhone 3.0 SDK 里，buffer 大小已经被减小到 32KB 了，我们的上传进度条可以更精确了。

追踪单个 request 的下载进度

这个例子中， myProgressIndicator 是个 NSProgressIndicator.

```
ASIHTTPRequest *request = [ASIHTTPRequest requestWithURL:url];
[request setDownloadProgressDelegate:myProgressIndicator];
[request startSynchronous];
NSLog(@"Max: %f, Value: %f", [myProgressIndicator maxValue],[myProgressIndicator
```

doubleValue]);

追踪一系列 request 的下载进度

在这个例子中, myProgressIndicator 是个 UIProgressView, myQueue 是个 ASINetworkQueue.

```
- (void)fetchThisURLFiveTimes:(NSURL *)url
{
    [myQueue cancelAllOperations];
    [myQueue setDownloadProgressDelegate:myProgressIndicator];
    [myQueue setDelegate:self];
    [myQueue setRequestDidFinishSelector:@selector(queueComplete:)];
    int i;
    for (i=0; i<5; i++) {
        ASIHTTPRequest *request = [ASIHTTPRequest requestWithURL:url];
        [myQueue addOperation:request];
    }
    [myQueue go];
}

- (void)queueComplete:(ASINetworkQueue *)queue
{
    NSLog(@"Value: %f", [myProgressIndicator progress]);
}
```

这个例子中，我们已经为 ASINetworkQueues 调用过[myQueue go]了。

追踪单个 request 的上传进度

在这个例子中, myProgressIndicator 是个 UIProgressView。

```
ASIFormDataRequest *request = [ASIFormDataRequest requestWithURL:url];
[request setPostValue:@"Ben" forKey:@"first_name"];
[request setPostValue:@"Copsey" forKey:@"last_name"];
[request setUploadProgressDelegate:myProgressIndicator];
[request startSynchronous];
NSLog(@"Value: %f",[myProgressIndicator progress]);
```

追踪一系列 request 的上传进度

这个例子中, myProgressIndicator 是个 NSProgressIndicator, myQueue 是个 ASINetworkQueue.

```
- (void)uploadSomethingFiveTimes:(NSURL *)url
{
    [myQueue cancelAllOperations];
    [myQueue setUploadProgressDelegate:myProgressIndicator];
    [myQueue setDelegate:self];
    [myQueue setRequestDidFinishSelector:@selector(queueComplete:)];
    int i;
    for (i=0; i<5; i++) {
        ASIHTTPRequest *request = [ASIFormDataRequest requestWithURL:url];
        [request setPostBody:[@"Some data"
dataUsingEncoding:NSUTF8StringEncoding]];
        [myQueue addOperation:request];
    }
    [myQueue go];
}

- (void)queueComplete:(ASINetworkQueue *)queue
{
    NSLog(@"Max: %f, Value: %f", [myProgressIndicator
maxValue],[myProgressIndicator doubleValue]);
}
```

精确进度条 vs 简单进度条

ASIHTTPRequest 提供两种进度条显示，简单进度条和精确进度条，使用 ASIHTTPRequests 和 ASINetworkQueues 的 showAccurateProgress 来控制。为一个 request 设置 showAccurateProgress 只会对该 request 有效。如果你为一个队列设置 showAccurateProgress，那么会影响队列里所有的 request。

简单进度条

当使用简单进度条时，进度条只会在一个 request 完成时才更新。对于单个 request，这意味着你只有两个进度状态：0%和 100%。对于一个有 5 个 request 的队列来说，有五个状态：0%，25%，50%，75%，100%，每个 request 完成时，进度条增长一次。

简单进度条（showAccurateProgress = NO）是 ASINetworkQueue 的默认值，适用于大量小数据请求。

精确进度条

当使用精确进度条时，每当字节被上传或下载时，进度条都会更新。它适用于上传/下载大块数据的请求，并且会更好的显示已经发送/接收的数据量。

使用精确进度条追踪上传会轻微降低界面效率，因为进度 delegate（一般是 UIProgressViews 或 NSProgressIndicators）会更频繁地重绘。

使用精确进度条追踪下载会更影响界面效率，因为队列会先为每个 GET 型 request 进行 HEAD 请求，以便统计总下载量。强烈推荐对下载大文件的队列使用精确进度条，但是要避免对大量小数据请求使用精确进度条。

精确进度条（showAccurateProgress = YES）是以同步方式执行的 ASIHTTPRequest 的默认值。

自定义进度追踪

ASIProgressDelegate 协议定义了所有能更新一个 request 进度的方法。多数情况下，设置你的 uploadProgressDelegate 或者 downloadProgressDelegate 为 NSProgressIndicator 或者 UIProgressView 会很好。但是，如果你想进行更复杂的追踪，你的进度 delegate 实现下列函数要比 setProgress: (iOS) 或者 setDoubleValue: / setMaxValue: (Mac)好：

这些函数允许你在实际量的数据被上传或下载时更新进度，而非简单方法的 0 到 1 之间的数字。

downloadProgressDelegates 方法

request:didReceiveBytes: 每次 request 下载了更多数据时，这个函数会被调用（注意，这个函数与一般的代理实现的 request:didReceiveData:函数不同）。

request:incrementDownloadSizeBy: 当下载的大小发生改变时，这个函数会被调用，传入的参数是你需要增加的大小。这通常发生在 request 收到响应头并且找到下载大小时。

uploadProgressDelegates 方法

request:didSendBytes: 每次 request 可以发送更多数据时，这个函数会被调用。注意：当一个 request 需要消除上传进度时（通常是该 request 发送了一段数据，但是因为授权失败或者其他什么原因导致这段数据需要重发）这个函数会被传入一个小于零的数字。

（六）ASIHTTPRequest-身份验证

发布者： Seven's - 2011/10/17 - 分类：ASIHTTPRequest 中文文档

你可以查阅 ASIHTTPRequest 授权流程图来了解 ASIHTTPRequest 如何找到授权凭据，并将授权凭据应用到 request 上。

为 URL 指定要使用的用户名和密码

```
NSURL *url = [NSURL URLWithString:@"http://www.dreamingwish.com/"];
ASIHTTPRequest *request = [ASIHTTPRequest requestWithURL:url];
```

为 request 指定要使用的用户名和密码

```
NSURL *url = [NSURL URLWithString:@"http://www.dreamingwish.com/"];
ASIHTTPRequest *request = [ASIHTTPRequest requestWithURL:url];
[request setUsername:@"username"];
[request setPassword:@"password"];
```

将凭据存储到 keychain

如果打开了 keychainPersistence，所有提供的可用的用户名和密码将被存储到 keychain 中，以后的 request 将会重用这些用户名密码，即使你关闭程序后重新打开也不影响。

```
NSURL *url = [NSURL URLWithString:@"http://www.dreamingwish.com/"];
ASIHTTPRequest *request = [ASIHTTPRequest requestWithURL:url];
[request setUseKeychainPersistence:YES];
[request setUsername:@"username"];
[request setPassword:@"password"];
```

如果你使用 keychain 但是想要自己管理它，你可以在 ASIHTTPRequest.h 文件里找到相关的类方法。

将凭据存储到 session 中

如果打开了 useSessionPersistence（默认即是如此），ASIHTTPRequest 会把凭据存储到内存中，后来的 request 将会重用这些凭据。

```
NSURL *url = [NSURL URLWithString:@"http://www.dreamingwish.com/"];
ASIHTTPRequest *request = [ASIHTTPRequest requestWithURL:url];
[request setUsername:@"username"];
[request setPassword:@"password"];
[request setUseSessionPersistence:YES]; //这一项是默认的，所以并不必要

//将会重用我们的 username 和 password
request = [ASIHTTPRequest requestWithURL:url];
```

NTLM 授权

要使用 NTLM 授权的 Windows 服务器，你还需要指定你要进行授权域。

```
NSURL *url = [NSURL URLWithString:@"http://www.dreamingwish.com/"];
ASIHTTPRequest *request = [ASIHTTPRequest requestWithURL:url];
[request setUsername:@"username"];
[request setPassword:@"password"];
[request setDomain:@"my-domain"];
```

使用代理来提供凭据

你不一定非要提前指定授权凭据，你还可以让每个 request 在无法从 session 或 keychain 中找到凭据时向它们的代理请求凭据。如果你要连接到一个你并不清楚授权类型的服务器时，这是很有用的。

你的 delegate 必须实现 authenticationNeededForRequest:方法，当 request 等待凭据时，ASIHTTPRequest 将会暂停这个 request。如果你持有你需要的凭据，那么先为 request 设定凭据，然后调用[request retryUsingSuppliedCredentials]即可。如果你想取消授权，调用[request cancelAuthentication]，此时，这个 request 也会被取消。

从 1.0.8 版开始，一次只能有一个 request 的 delegate 收到 authenticationNeededForRequest: 或者 proxyAuthenticationNeededForRequest:。当 delegate 处理第一个 request 时，其他需要授权的 request 将会被暂停。如果提供了一个凭据，当前进程中所有其他的 request 将会假定这个凭据对这个 URL 有效，并尝试重用这个凭据。如果 delegate 取消了授权，并且队列的 shouldCancelAllRequestsOnFailure 值为 YES，所有其他的 request 都将被取消（它们也不会尝试请求凭据）。

当进行同步请求时，你不可以使用代理模式来授权。

在较老的版本中，这么做会导致程序假死，从 1.0.8 开始，即使你这么做了，代理函数也不会被调用。

使用内建的授权对话框（目前只对 iOS 有效）

这个特性归功于 1.0.8 版本的新类 ASIAuthenticationDialog 。这个特性主要是用于授权代理（后面会介绍到），但是它也可以用来向用户取得授权凭据。

为了更好的用户体验，大多数（连接单一服务的）app 必须为 request 的 delegate 实现 authenticationNeededForRequest:方法，或者避免同时使用代理式授权。

most apps that connect to a single service should implement authenticationNeededForRequest: in their request delegates, or avoid the use of delegation-style authentication altogether.

但是，会有一些情况下，为普通的授权使用 ASIHTTPRequest 的标准授权对话框更好：

你不想创建你自己的登录表单
你可能需要从外部资源获取数据，但是你不清楚你需不需要进行授权

对于这些情况，为 request 设置 shouldPresentAuthenticationDialog 为 YES，此时，如果你的代理没有实现

asihttprequest 授权对话框

authenticationNeededForRequest:方法，那么用户将会看到这个对话框。

一次同时只有一个对话框可以显示出来，所以当一个对话框显示时，所有其他需要授权的 request 将会暂停。如果提供了一个凭据，当前进程中所有其他的 request 将会假定这个凭据对这个 URL 有效，并尝试重用这个凭据。如果 delegate 取消了授权，并且队列的 shouldCancelAllRequestsOnFailure 值为 YES，所有其他的 request 都将被取消（它们也不会尝试请求凭据）。

对于同步请求的 request，授权对话框不会显示出来。

这个对话框部分模仿了 iPhone 上 Safari 使用的授权对话框，它包含以下内容：

一段信息来说明这些凭据是用于 websever（而非一个 proxy）
你将要连接到服务器的主机名或者 IP
授权域（如果提供的话）
填写用户名和密码的区域
当连接到 NTLM 授权模式的服务器时，还会包含一个填写 domain 的区域
一个说明信息，指明凭据是否将会被以明文方式发送（例如：“只有当使用基于非 SSL 的基本授权模式时才会以明文方式发送”）

如果你想改变它的外观，你必须继承 ASIHTTPRequest，并重写 showAuthenticationDialog 来显示你自己的对话框或 ASIAuthenticationDialog 子类。

在服务器请求凭据前向服务器发送凭据

IMPORTANT

从 1.8.1 开始，使用基本授权模式的 request 时，这个特性的行为改变了。你可能需要修改你的代码。

在第一次生成 request 时，ASIHTTPRequest 可以先向服务器发送凭据（如果有的话），而不是等服务器要求提供凭据时才提供凭据。这个特性可以提高使用授权的程序的执行效率，因为这个特性避免了多余的 request。

对于基本授权模式，要触发这个行为，你必须手动设置 request 的 authenticationScheme 为 kCFHTTPAuthenticationSchemeBasic：

```
[request setAuthenticationScheme:(NSString *)kCFHTTPAuthenticationSchemeBasic];
```

对于其他授权方案，凭据也可以在服务器要求之前被发送，但是仅当有另一个 request 成功授权之后才行。

在以下情况下，你也许想要禁用这个特性：

你的程序可能会一次使用一系列凭据来与服务器对话

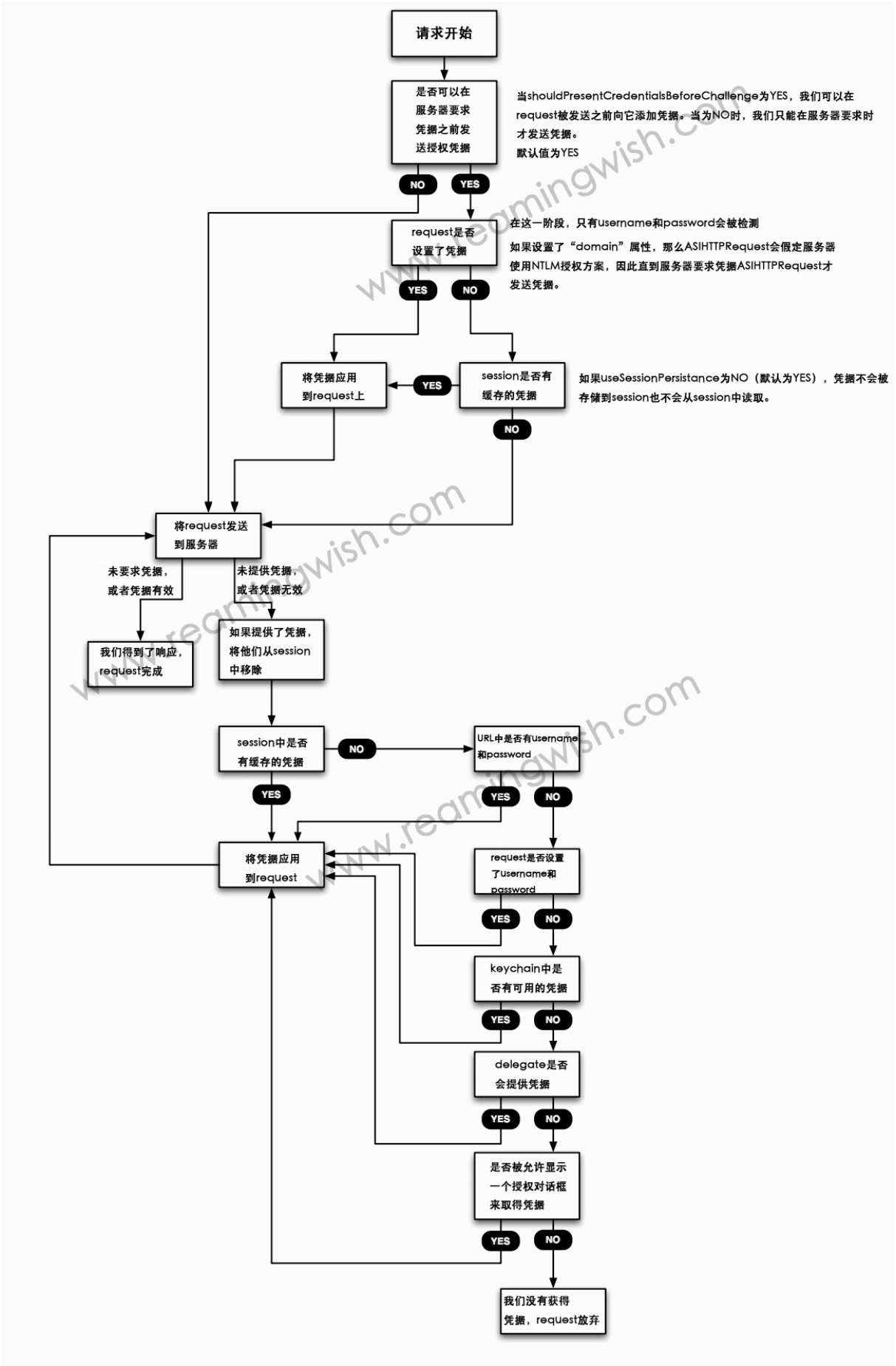
安全性对于你的程序来说非常重要。使用这个特性是相对不安全的，因为你不能在凭据被发送前验证你是否连接到了正确的服务器。

要禁用这个特性，这样做：

```
[request setShouldPresentCredentialsBeforeChallenge:NO];
```

（七）ASIHTTPRequest-HTTP 授权-流程图

发布者： Seven's - 2011/10/17 - 分类：ASIHTTPRequest 中文文档


请求开始
是否可以在服务器要求凭据之前发送授权凭据
当shouldPresentCredentialsBeforeChallenge为YES，我们可以在request被发送之前向它添加凭据。当为NO时，我们只能在服务器要求时才发送凭据。
默认值为YES
NO
YES
request是否设置了凭据
在这一阶段，只有username和password会被检测
如果设置了“domain”属性，那么ASIHTTPRequest会假定服务器使用NTLM授权方案，因此直到服务器要求凭据ASIHTTPRequest才发送凭据。
YES
NO
将凭据应用到request上
YES
session是否有缓存的凭据
如果useSessionPersistance为NO（默认为YES），凭据不会被存储到session也不会从session中读取。
NO
将request发送到服务器
未要求凭据，或者凭据有效
未提供凭据，或者凭据无效
我们得到了响应，request完成
如果提供了凭据，将他们从session中移除
session中是否有缓存的凭据
NO
URL中是否有username和password
YES
YES
NO
将凭据应用到request
request是否设置了username和password
YES
NO
keychain中是否有可用的凭据
YES
NO
delegate是否会提供凭据
YES
NO
是否被允许显示一个授权对话框来取得凭据
YES
NO
我们没有获得凭据，request放弃

（八）ASIHTTPRequest-Cookie 的使用

发布者： Seven's - 2011/10/17 - 分类：ASIHTTPRequest 中文文档

持久化 cookie

ASIHTTPRequest 允许你使用全局存储来和所有使用 CFNetwork 或者 NSURLRequest 接口的程序共享 cookie。

如果设置 useCookiePersistence 为 YES（默认值），cookie 会被存储在共享的 NSHTTPCookieStorage 容器中，并且会自动被其他 request 重用。值得一提的是，ASIHTTPRequest 会向服务器发送其他程序创建的 cookie（如果这些 cookie 对特定 request 有效的话）。

你可以清空 session 期间创建的所有 cookie：

[ASIHTTPRequest setSessionCookies:nil];

这里的‘session cookies’指的是一个 session 中创建的所有 cookie，而非没有过期时间的 cookie（即通常所指的会话 cookie，这种 cookie 会在程序结束时被清除）。

另外，有个方便的函数 clearSession 可以清除 session 期间产生的所有的 cookie 和缓存的授权数据。

自己处理 cookie

如果你愿意，你大可以关闭 useCookiePersistence，自己来管理某个 request 的一系列 cookie：

```
//创建一个 cookie
NSDictionary *properties = [[[NSMutableDictionary alloc] init] autorelease];
[properties setValue:[@"Test Value" encodedCookieValue]
forKey:NSHTTPCookieValue];
[properties setValue:@"ASIHTTPRequestTestCookie" forKey:NSHTTPCookieName];
[properties setValue:@".dreamingwish.com" forKey:NSHTTPCookieDomain];
[properties setValue:[NSDate dateWithTimeIntervalSinceNow:60*60]
forKey:NSHTTPCookieExpires];
[properties setValue:@"/asi-http-request/tests" forKey:NSHTTPCookiePath];
NSHTTPCookie *cookie = [[[NSHTTPCookie alloc] initWithProperties:properties]
autorelease];
```

```
//这个 url 会返回名为'ASIHTTPRequestTestCookie'的 cookie 的值
url = [NSURL URLWithString:@"http://www.dreamingwish.com/"];
request = [ASIHTTPRequest requestWithURL:url];
[request setUseCookiePersistence:NO];
[request setRequestCookies:[NSMutableArray arrayWithObject:cookie]];
[request startSynchronous];

//将会打印: I have 'Test Value' as the value of 'ASIHTTPRequestTestCookie'
NSLog(@"%@",[request responseString]);
```

（九）ASIHTTPRequest-数据压缩

发布者： Seven's - 2011/10/17 - 分类：ASIHTTPRequest 中文文档

使用 gzip 来处理压缩的响应数据

从 0.9 版本开始，ASIHTTPRequest 会提示服务器它可以接收 gzip 压缩过的数据。

许多 web 服务器可以在数据被发送之前压缩这些数据——这可以加快下载速度减少流量使用，但会让服务器的 cpu（压缩数据）和客户端（解压数据）付出代价。总的来说，只有特定的几种数据会被压缩——许多二进制格式的文件像 jpeg，gif，png，swf 和 pdf 已经压缩过他们的数据了，所以向客户端发送这些数据时不会进行 gzip 压缩。文本文件例如网页和 xml 文件会被压缩，因为它们通常有大量的数据冗余。

怎样设置 apache 的 mod_deflate 来使用 gzip 压缩数据

apache 2.x 以上版本已经配备了 mod_deflate 扩展，这使得 apache 可以透明地压缩特定种类的数据。要开启这个特性，你需要在 apache 的配置文件中启用 mod_deflate。并将 mod_deflate 命令添加到你的虚拟主机配置或者.htaccess 文件中。

在 ASIHTTPRequest 中使用 gzip

```
- (IBAction)grabURL:(id)sender
{
    NSURL *url = [NSURL URLWithString:@"http://www.dreamingwish.com"];
    ASIHTTPRequest *request = [ASIHTTPRequest requestWithURL:url];
```

```
    // 默认为 YES, 你可以设定它为 NO 来禁用 gzip 压缩
    [request setAllowCompressedResponse:YES];
    [request startSynchronous];
    BOOL *dataWasCompressed = [request isResponseCompressed]; // 响应是否被 gzip 压缩过？
    NSData *compressedResponse = [request rawResponseData]; // 压缩的数据
    NSData *uncompressedData = [request responseData]; // 解压缩后的数据
    NSString *response = [request responseString]; // 解压缩后的字符串
}
```

当 allowCompressedResponse 设置为 YES 时，ASIHTTPRequest 将向 request 中增加一个 Accept-Encoding 头，表示我们可以接收 gzip 压缩过的数据。如果响应头中包含一个 Content-Encoding 头指明数据是压缩过的，那么调用 responseData 或者 responseString 将会得到解压缩后的数据。你也可以通过调用 rawResponseData 来获得原始未压缩的数据。

相应数据的实时解压缩

默认情况下，ASIHTTPRequest 会等到 request 完成时才解压缩返回的数据。若设置 request 的 shouldWaitToInflateCompressedResponses 属性为 NO，ASIHTTPRequest 将会对收到的数据进行实时解压缩。 在某些情况下，这会稍稍提升速度，因为数据可以在 reqeust 等待网络数据时进行处理。

如果你需要对响应数据流进行流处理（例如 XML 和 JSON 解析），这个特性会很有用。如果启用了这个选项，你可以通过实现代理函数 request:didReceiveData:来将返回的网络数据一点一点喂给解析器。

注意，如果 shouldWaitToInflateCompressedResponses 被设置为 NO，那么原始（未解压）的数据会被抛弃。具体情况请查阅 ASIHTTPRequest.h 的代码注释。

使用 gzip 压缩 request 数据

1.0.3 版本的新特性就是 gzip 压缩 request 数据。使用这个特性，你可以通过设置 shouldCompressRequestBody 为 YES 来使你的程序压缩 POST/PUT 的内容，默认值为 NO。

apache 的 mod_deflate 可以自动解压缩 gzip 压缩的请求体（通过合适的设置）。这个方法适用于 CGI 内容，但不适用于内容过滤器式的模块（例如 mod PHP），这种情况下，你就必须自己解压缩数据。

ASIHTTPRequest 无法检测一个服务器是否能接收压缩过的请求体。当你确定服务器可以解压缩 gzip 包时，再使用这个特性。

请避免对已经压缩过的格式（例如 jpeg/png/gif/pdf/swf）进行压缩，你会发现压缩后的数据比原数据更大。（梦维：因为压缩包都有头信息）

（十）ASIHTTPRequest-断点续传（下载）

发布者： Seven's - 2011/10/18 - 分类：ASIHTTPRequest 中文文档

从 0.94 版本开始，ASIHTTPRequest 可以恢复中断的下载

```
- (IBAction)resumeInterruptedDownload:(id)sender
{
   NSURL *url = [NSURL URLWithString:

@"http://www.dreamingwish.com/wp-content/uploads/2011/10/asihttprequest-auth.
png"];
   ASIHTTPRequest *request = [ASIHTTPRequest requestWithURL:url];

   NSString *downloadPath = @"/Users/ben/Desktop/asi.png";

   //当 request 完成时，整个文件会被移动到这里
   [request setDownloadDestinationPath:downloadPath];

   //这个文件已经被下载了一部分
   [request
setTemporaryFileDownloadPath:@"/Users/ben/Desktop/asi.png.download"];
   [request setAllowResumeForFileDownloads:YES];
   [request startSynchronous];

   //整个文件将会在这里
   NSString *theContent = [NSString stringWithContentsOfFile:downloadPath];
}
```

这个特性只对下载数据到文件中有效，你必须为一下情况的 request 设置 allowResumeForFileDownloads 为 YES：

任何你希望将来可以断点续传的下载（否则，ASIHTTPRequest 会在取消或者释放内存时将临时文件删除）

任何你要进行断点续传的下载

另外，你必须自己设置一个临时下载路径（setTemporaryFileDownloadPath），这个路径是未完成的数据的路径。新的数据将会被添加到这个文件，当下载完成时，这个文件将被移动到 downloadDestinationPath 。

断点续传的工作原理是读取 temporaryFileDownloadPath 的文件的大小，并使用 Range: bytes=x HTTP 头来请求剩余的文件内容。

ASIHTTPRequest 并不检测是否存在 Accept-Ranges 头（因为额外的 HEAD 头请求会消耗额外的资源），所以只有确定服务器支持断点续传下载时，再使用这个特性。

（十一）ASIHTTPRequest-直接读取磁盘数据流的请求体

发布者： Seven's - 2011/10/18 - 分类：ASIHTTPRequest 中文文档

从 0.96 版本开始，ASIHTTPRequest 可以使用磁盘上的数据来作为请求体。这意味着不需要将文件完全读入内存中，这就避免的当使用大文件时的严重内存消耗。

使用这个特性的方法有好几种：
ASIFormDataRequests

```
NSURL *url = [NSURL URLWithString:@"http://www.dreamingwish.com/"];
ASIFormDataRequest *request = [ASIFormDataRequest requestWithURL:url];
[request setPostValue:@"foo" forKey:@"post_var"];
[request setFile:@"/Users/ben/Desktop/bigfile.txt" forKey:@"file"];
[request startSynchronous];
```

当使用 setFile:forKey:时，ASIFormDataRequests 自动使用这个特性。request 将会创建一个包含整个 post 体的临时文件。文件会一点一点写入 post 体。这样的 request 是由 CFReadStreamCreateForStreamedHTTPRequest 创建的，它使用文件读取流来作为资源。

普通 ASIHTTPRequest

如果你明白自己的 request 体会很大，那么为这个 request 设置流式读取模式。

```
[request setShouldStreamPostDataFromDisk:YES];
```

下面的例子中，我们将一个 NSData 对象添加到 post 体。这有两个方法：从内存中添加（appendPostData:），或者从文件中添加（appendPostDataFromFile:）；

```
NSURL *url = [NSURL URLWithString:@"http://www.dreamingwish.com/"];
ASIHTTPRequest *request = [ASIHTTPRequest requestWithURL:url];
[request setShouldStreamPostDataFromDisk:YES];
[request appendPostData:myBigNSData];
[request appendPostDataFromFile:@"/Users/ben/Desktop/bigfile.txt"];
[request startSynchronous];
```

这个例子中，我们想直接 PUT 一个大文件。我们得自己设置 setPostBodyFilePath ，ASIHTTPRequest 将使用这个文件来作为 post 体。

```
NSURL *url = [NSURL URLWithString:@"http://www.dreamingwish.com"];
ASIHTTPRequest *request = [ASIHTTPRequest requestWithURL:url];
[request setRequestMethod:@"PUT"];
[request setPostBodyFilePath:@"/Users/ben/Desktop/another-big-one.txt"];
[request setShouldStreamPostDataFromDisk:YES];
[request startSynchronous];
```

IMPORTANT:切勿对使用上述函数的 request 使用 setPostBody——他们是互斥的。只有在你要自己建立 request 的请求体，并且还准备在内存中保持这个请求体时，才应该使用 setPostBody。

（十二）ASIHTTPRequest-使用 download cache

发布者： Seven's - 2011/10/18 - 分类：ASIHTTPRequest 中文文档

从 1.8 版本开始，ASIDownloadCache 和 ASICacheDelegate 的 API 改变了，你可能需要修改你的代码。

尤其是，cache 策略的可用选项发生了改变，你现在可以对单一 request 使用结合的 cache 策略

ASIHTTPRequest 可以自动缓存下载的数据。在很多情况下这很有用：

当你离线时，你无法再次下载数据，而你又需要访问这些数据
从上次下载这些数据后，你只想在数据更新后才下载新的数据
你处理的数据永远不会发生改变，所以你只想下载一次数据

在之前版本的 ASIHTTPRequest 里，遇到上述情况，你只能自己处理这些策略。在一些情况下，使用 download cache 可以让你不用再写本地缓存机制。

ASIDownloadCache 是个简单的 URL cache，可以用来缓存 GET 请求的相应数据。一个 request 要被缓存，它首先必须请求成功（没有发送错误），服务器必须返回 200HTTP 状态值。或者，从 1.8.1 版本开始，301,302,303,307 重定向状态码都可以。

要打开响应值的 cache 机制很简单：

```
[ASIHTTPRequest setDefaultCache:[ASIDownloadCache sharedCache]];
```

这样做以后，所有的 request 都会自动使用 cache。如果你愿意，你可以让不同的 request 使用共享的 cache：

```
ASIHTTPRequest *request = [ASIHTTPRequest requestWithURL:url];
[request setDownloadCache:[ASIDownloadCache sharedCache]];
```

你不会被局限于使用单一的 cache，你可以想创建多少 cache 就创建多少 cache，只要你喜欢 ^^。当你自己创建一个 cache，你必须设定 cache 的路径——这路径必须是一个你拥有写权限的目录。

```
ASIDownloadCache *cache = [[[ASIDownloadCache alloc] init] autorelease];
[cache setStoragePath:@"/Users/ben/Documents/Cached-Downloads"];

//别忘了 - 你必须自己 retaining 你自己的 cache!
[self setMyCache:cache];

ASIHTTPRequest *request = [ASIHTTPRequest requestWithURL:url];
[request setDownloadCache:[self myCache]];
```

cache 策略

cache 策略是你控制 cache 中信息的主要方法，控制何时使用 cache 数据而非重新下载数据。

每个 request 的 cache 策略可是由 request 的 cachePolicy 属性来控制的。cache 策略使用掩码来定义，所以你可以二进制“与”操作他们。

```
// 每次都向服务器询问是否有新的内容可用，
// 如果请求失败，使用 cache 的数据，即使这个数据已经过期了
[request
setCachePolicy:ASIAskServerIfModifiedCachePolicy|ASIFallbackToCacheIfLoadFailsCac
hePolicy];
```

你可以使用下列 cache 策略选项来控制 request 的缓存策略：
ASIUseDefaultCachePolicy

默认的 cache 策略。请勿将这一项与其他项结合使用。当你设置一个 request 使用 cache，它会使用 cache 的 defaultCachePolicy. ASIDownloadCache 的默认 cache 策略是‘ASIAskServerIfModifiedWhenStaleCachePolicy’.
ASIDoNotReadFromCacheCachePolicy

使用这一项，request 将不会从 cache 中读取数据
ASIDoNotWriteToCacheCachePolicy

使用这一项，request 将不会把数据存入 cache

ASIAskServerIfModifiedWhenStaleCachePolicy

这是 ASIDownloadCaches 的默认 cache 策略。使用这个策略时，request 会先查看 cache 中是否有可用的缓存数据。如果没有，request 会像普通 request 那样工作。

如果有缓存数据并且缓存数据没有过期，那么 request 会使用缓存的数据，而且不会向服务器通信。如果缓存数据过期了，request 会先进行 GET 请求来想服务器询问数据是否有新的版本。如果服务器说缓存的数据就是当前版本，那么缓存数据将被使用，不会下载新数据。在这种情况下，cache 的有效期将被设定为服务器端提供的新的有效期。如果服务器提供更新的内容，那么新内容会被下载，并且新的数据以及它的有效期将被写入 cache。
ASIAskServerIfModifiedCachePolicy

这一项与 ASIAskServerIfModifiedWhenStaleCachePolicy 相同，除了一点：request 将会每次都询问服务器端数据是否有更新。
ASIOnlyLoadIfNotCachedCachePolicy

使用这一项，cache 数据将一直被使用，无视过期时间
ASIDontLoadCachePolicy

使用这一项时，只有当响应数据有缓存时，request 才会成功。如果一个 request 没有缓存的响应数据，那么这个 request 将会停止，并且不会有错误设置在 request 上。
ASIFallbackToCacheIfLoadFailsCachePolicy

当使用这一项时，当 request 失败时，request 会回头请求 cache 数据。如果请求失败后，request 使用的 cache 数据，那么这个 request 会成功（没有错误）。你通常会将这一项与其他项结合使用，因为它适用于指定当发生错误时 request 的行为。

当你设定了一个 cache 对象的 defaultCachePolicy 属性，所有使用这个 cache 对象的 request 都会使用这个 cache 策略，除非你为 request 设置了另外的策略。

存储策略

存储策略允许你定义一个 cache 可以存储特定的相应数据多久。ASIHTTPRequest 目前支持两种存储策略：

ASICacheForSessionDurationCacheStoragePolicy 是默认值。相应数据只会在会话期间被存储，在第一次使用 cache 时，或者在调用 [ASIHTTPRequest clearSession]时，数据会被清除。

使用 ASICachePermanentlyCacheStoragePolicy，缓存的相应数据会被永久存储。要使用这个存储策略，向 request 设置：

```
ASIHTTPRequest *request = [ASIHTTPRequest requestWithURL:url];
[request setCacheStoragePolicy:ASICachePermanentlyCacheStoragePolicy];
```

要手动清除 cache，调用函数 clearCachedResponsesForStoragePolicy:,传入要清除的 cache 数据的存储策略：

```
[[ASIDownloadCache sharedCache]
clearCachedResponsesForStoragePolicy:ASICachePermanentlyCacheStoragePolicy];
```

其他 cache 相关的特性

```
// 当你关闭 shouldRespectCacheControlHeaders,cache 对象会存储响应数据，而无视
// 服务器的显式“请勿缓存”声明 (例如：cache-control 或者 pragma: no-cache 头)
[[ASIDownloadCache sharedCache] setShouldRespectCacheControlHeaders:NO];

// 可以设定 request 的 secondsToCache 来覆盖服务器设定的内容有效期, 这时，响应数据
// 会一直被缓存，直到经过 secondsToCache 秒
ASIHTTPRequest *request = [ASIHTTPRequest requestWithURL:url];
[request setSecondsToCache:60*60*24*30]; // 缓存 30 天

//当 request 开始执行后,如果响应数据是从缓存中取得的，didUseCachedResponse 会返回 YES
[request didUseCachedResponse];

// 向 cache 对象索取一个路径来存储相应数据. 这是使用 download cache 的最有效率的方法,
// 因为此时，当 request 完成后，数据不需要被复制到 cache 中.
[request setDownloadDestinationPath:
    [[ASIDownloadCache sharedCache]
pathToStoreCachedResponseDataForRequest:request]];
```

编写自己的 cache

如果你已经持有一个 download cache 并且想将他插入 ASIHTTPRequest 中，或者你喜欢自己写自己的 download cache，那么让你的 cache 实现 ASICacheDelegate 协议。

（十三）ASIHTTPRequest-流量控制

发布者： Seven's - 2011/10/18 - 分类：ASIHTTPRequest 中文文档

从 1.0.7 版本开始，ASIHTTPRequest 可以控制流量，使得所有 request 的流量不会超过用户定义的限制范围。这可以使得发送/接收大量数据的 iphone 程序更容易通过苹果的 app store 的审核。

流量是由一个全局的数量限制（字节）来控制的——每秒钟可以传送多少流量的数据。所有 request 共享这个限制。在发送或接收数据时，ASIHTTPRequest 保持追踪上一秒所发送/接收的数据量。如果一个 request 达到了限制，其他正在执行的 request 将会等待。 在 iOS 上，你可以让 ASIHTTPRequest 在使用 WWAN (GPRS/Edge/3G) 连接时自动打开流量控制，而当使用 WiFi 连接时自动关闭流量限制。

```
// 这将会对 WWAN 连接下的 request 进行流量控制（控制到预定义的值）
// Wi-Fi 连接下的 request 不会受影响
// 这个方法仅在 iOS 上可用
[ASIHTTPRequest setShouldThrottleBandwidthForWWAN:YES];

// 这将会对 WWAN 连接下的 request 进行流量控制（控制到自定义的值）
// 这个方法仅在 iOS 上可用
[ASIHTTPRequest throttleBandwidthForWWANUsingLimit:14800];

// 这将会控移动应用（mobile applications）的流量到预定义的值.
// 会限制所有 requests, 不管 request 是不是 WiFi 连接下的 - 使用时要注意
[ASIHTTPRequest setMaxBandwidthPerSecond:ASIWWANBandwidthThrottleAmount];

// 记录每秒有多少字节的流量 (过去 5 秒内的平均值)
NSLog(@"%qi",[ASIHTTPRequest averageBandwidthUsedPerSecond]);
```

IMPORTANT：在启用流量控制前，请参阅以下条目：

流量控制特性是试验型的特性：你自己得承担风险

不要把带宽限制设置得很低——最好不要低于 ASIWWANBandwidthThrottleAmount

实际流量往往会比你程序设置的流量稍稍偏高，因为流量的测量并不包含 HTTP 头。

ASIWWANBandwidthThrottleAmount 的值是非官方的，据我所知，官方并没有公布流量限制大小

除非你的程序会下载或者上传大量的数据，否则不要开启流量控制。最好是当即将下载或上传大量数据时才启用它，而其他时间应该禁用它。

这玩意应该会按我描述的情况来工作，但是我并不保证你的 app 使用了我的流量控制就不会被驳回。

（十四）ASIHTTPRequest-客户端证书支持

发布者： Seven's - 2011/10/19 - 分类：ASIHTTPRequest 中文文档

有时服务器要求提供客户端证书，从 1.8 版本开始，你可以随 request 发送证书。

```
// Will send the certificate attached to the identity (identity is a SecIdentityRef)
[request setClientCertificateIdentity:identity];

// Add an additional certificate (where cert is a SecCertificateRef)
[request setClientCertificates:[NSArray arrayWithObject:(id)cert]];
```

在 iPhone/iPad 示例工程中的 ClientCertificateTests.m 中有一个很有用的函数用来从 PKCS12 数据创建 SecIdentityRef （这个函数仅适用于 iOS）。

（十五）ASIHTTPRequest-使用代理连接

发布者： Seven's - 2011/10/19 - 分类：ASIHTTPRequest 中文文档

ASIHTTPRequest 检测系统的 proxy 设置并自动将 proxy 用于 request。从 1.0.6 版本开始，它还支持 PAC 文件和要求授权的 proxy。

默认情况下，ASIHTTPRequest 将尝试自动检测 proxy 设置。当然，你可以看自己手动设置：

```
//  手动设置代理服务器
NSURL *url = [NSURL URLWithString:@"http://www.dreamingwish.com"];
ASIHTTPRequest *request = [ASIHTTPRequest requestWithURL:url];
[request setProxyHost:@"192.168.0.1"];
[request setProxyPort:3128];

//  另一种方法, 使用代理配置脚本文件
// (最好使用本地 pac 文件)
[request setPACurl:[NSURL URLWithString:@"path/to/test.pac"]];
```

要求授权的 proxy

在 Mac OS 上，ASIHTTPRequest 可以自动检测到要求授权的 proxy 的凭据（前提是在系统设定中设置好）。在 iOS 上，ASIHTTPRequest 则无法自动检测出授权凭据，所以你要么手动使用 delegate 来向你的 controller 或者用户索取合适的凭据，要么让 ASIAuthenticationDialog 向用户索取凭据。一旦获得了一个有效的 proxy 凭据，那么该凭据将被存储到 keychian 中（前提是启用 useKeychainPersistence ）并自动重用。

手动为 proxy 指定凭据

```
NSURL *url = [NSURL URLWithString:@"http://www.dreamingwish.com"];
ASIHTTPRequest *request = [ASIHTTPRequest requestWithURL:url];
[request setProxyHost:@"192.168.0.1"];
[request setProxyPort:3128];

//为要求授权的 proxy 设置 username  和 password
[request setProxyUsername:@"bencopsey"];
[request setProxyPassword:@"password"];

//  对于 NTLM proxy,还要设置  domain (NTLM proxy 功能是未经测试的)
[request setProxyDomain:@"la.la.land"];
```

使用 delegate 来提供 proxy 凭据

这个特性的工作原理和“使用 delegate 提供 HTTP 授权”一样，只有一点不同：你的 delegate 要响应 proxyAuthenticationNeededForRequest:函数。

asihttprequest 授权对话框

使用内建的授权对话框（仅适用于 iOS）

这个特性归功于 1.0.8 版本的新类 ASIAuthenticationDialog 。用来向用户索取凭据来授权 webserver 或者 proxy。

如果你的 delegate 不响应 proxyAuthenticationNeededForRequest:函数，那么默认情况下，ASIHTTPRequest 将会显示一个对客户来提示用户输入授权凭据。使用 ASIHTTPRequest，开发者不再需要写额外的代码来显示授权对话框，因为默认情况下，ASIHTTPRequest 就会显示它。

使用同步 request 时 proxy 授权对话框不会显示出来。

如果你不限使用 proxy 授权对话框，那么你要么实现 proxyAuthenticationNeededForRequest:，要么设置 shouldPresentProxyAuthenticationDialog 为 false（此时你的程序将无法连接到 proxy）。如果你要改变对话框的样式，你得继承 ASIHTTPRequest 类，重写 showProxyAuthenticationDialog 来显示你自己的对话框或者 ASIAuthenticationDialog 子类.

（十六）ASIHTTPRequest-其他特性

发布者： Seven's - 2011/10/19 - 分类：ASIHTTPRequest 中文文档

设置 user agent

这样设置用户代理：

[ASIHTTPRequest setDefaultUserAgentString:@"MyApp 1.0"]

如果不设置 user agent，ASIHTTPRequest 会为你创建一个。例如（Mac OS 程序）：

My Application 1.0 (Macintosh; Mac OS X 10.5.7; en_GB)

你也可以为每个 request 设置 user agent：

[request setUserAgent:@"MyApp 1.0"]

当程序进入后台运行时，继续执行 request（iOS）

```
// iOS 4 以上
[request setShouldContinueWhenAppEntersBackground:YES];
```

监视网络活动

```
//记录过去 5 秒的平均流量字节/秒
NSLog(@"%llu",[ASIHTTPRequest averageBandwidthUsedPerSecond]);

if ([ASIHTTPRequest isNetworkInUse]) {
    // ASIHTTPRequest  进程中有 requests 正在使用网络
}
```

禁用自动更新网络连接标示符状态（iOS）

默认情况下，ASIHTTPRequest 在 request 使用网络连接时，会自动显示网络连接标示符（iOS 状态条中）。如果你想自己控制标示符，你可以禁用这个特性：

[ASIHTTPRequest setShouldUpdateNetworkActivityIndicator:NO];

超时自动重试

设置超时自动重试最大次数为 2：

[request setNumberOfTimesToRetryOnTimeout:2];

设置持久连接

默认情况下，ASIHTTPRequest 将会尝试保持对一个服务器的连接以便往后的连接到该服务器的 request 可以重用这个连接（这个特性可以显著地提高速度，尤其是当你会要进行很多小数据量 request 时）。当连接到 HTTP 1.1 服务器或者服务器发送 keep-alive 头时，持久连接会自动被使用。当服务器显式地发送”Connection:close”头时，持久连接就不会被使用。另外，默认情况下，ASIHTTPRequest 不会对包含请求体（例如 POST/PUT）的 request 使用持久连接（从 1.8.1 版本开始）。通过设置 request，你可以强制让此类 request 使用持久连接：

[request setRequestMethod:@"PUT"];
[request setShouldAttemptPersistentConnection:YES];

很多服务器不会在响应头中规定持久连接的持久时间，它们可能会在任何一个 request 完成时候关闭连接。如果一个服务器没有规定持久连接的持久时间，ASIHTTPRequest 将会在一个 request 完成后，保持连接 60 秒。对于你的服务器设置来时，60 可能很长，也可能很短。

如果这个超时时间太长，那么可能一个 request 使用这个连接时，服务器可能已经关闭了这个连接。当 ASIHTTPRequest 遇到连接已关闭错误，它就会在一个新的连接上重试这个 request。

如果这个超时时间太短，而服务器却更想让这个连接保持更长时间，但是 ASIHTTPRequest 又开启了不必要的新连接，那么这将导致效率降低。

// 设置持久连接的超时时间为 120 秒
[request setPersistentConnectionTimeoutSeconds:120];

// 彻底禁用持久连接
[request setShouldAttemptPersistentConnection:NO];

强制使用 HTTP 1.0

[request setUseHTTPVersionOne:YES];

禁用安全证书校验

如果你有一个自签名的证书，你可能想禁用证书校验来做测试。这里我建议你从一个可信的 CA 购买证书，并为生产(production)期的 app（梦维：app 还有测试期等等阶段不是？）启用证书校验。

[request setValidatesSecureCertificate:NO];

（十七）ASIHTTPRequest-Debug 选项

发布者： Seven's - 2011/10/19 - 分类：ASIHTTPRequest 中文文档

ASIHTTPRequest 提供少量的有助于调试 request 行为的宏标记。这些宏可以从 ASIHTTPRequestConfig.h 文件中找到。

当打开这些标志时，request 将会打印一些信息到控制台，显示它们正在做什么。

DEBUG_REQUEST_STATUS

打印 request 的生命周期的所有信息，开始，结束上载，结束下载。

DEBUG_THROTTLING

打印 request 使用了多少流量（大致），如果 request 的流量被控制，打印如何被控制。当与 DEBUG_REQUEST_STATUS 结合使用时，这一项可以用来调试“超时”，你可以看到 request 停止发送或接收数据的时间点。

DEBUG_PERSISTENT_CONNECTIONS

打印 request 如何重用持久连接的信息，如果你看到这样的信息：

Request attempted to use connection #1, but it has been closed – will retry with a new connection

这说明你设置的 persistentConnectionTimeoutSeconds 太大了。

DEBUG_HTTP_AUTHENTICATION

1.8.1 版本的新特性：开启这一项会打印 request 如何处理 HTTP 授权（Basic，Digest 或者 NTLM）的相关信息。

DEBUG_FORM_DATA_REQUEST

打印出 ASIFormDataRequest 将发送的整个 request 体。使用 ASIFormDataRequest 时，这一项很有用。

【相关网络资料】

（一）升级到 iOS5 后 ASIHttpRequest 库问题及解决方法

由于正式版的 iOS5 出来了，所以我也试着去升级了。于是下载了最新的 Xcode，才 1.7G 左右，比以往的安装包要小许多。

升级 Xcode 后，打开以前创建的工程， 运气好，一个错误都没有，程序也能正常跑起来。由于我程序中用了 ASIHttpRequest 这个库，让我发现了一个小问题，就是

ASIAuthenticationDialog 这个内置对话框在网络有代理的情况下出现，然后无论点 cancle 或是 login 都不能 dismiss。在 4.3 的 SDK 中完全没问题，在 5.0 的 SDK 中就会在 Console 中看到输出：

Unbalanced calls to begin/end appearance transitions for <ASIAutorotatingViewController:>

很明显示在 sdk5 中， 用这个库有问题，还有在停止调式的时候，程序会有异常产生。

于是很明显示是 SDK5 的变化影响了 ASIHttpRequest 的正常使用。于是我要 fix 这个问题，经过我研究发现，dismiss 不起作用是由于 UIViewController 的 parentViewController 不再返回正确值了，返回的是 nil，而在 SDK5 中被 presentingViewController 取代了。于是在 ASIAuthenticationDialog.m 中找到+(void)dismiss 这个方法并修改为：

```
+ (void)dismiss
{
#if __IPHONE_OS_VERSION_MAX_ALLOWED > __IPHONE_4_3
    UIViewController *theViewController = [sharedDialog
presentingViewController];
    [theViewController dismissModalViewControllerAnimated:YES];
#else
```

```
        UIViewController *theViewController = [sharedDialog parentViewController];
        [theViewController dismissModalViewControllerAnimated:YES];
    #endif
    }
```

这样编译出来的程序能在 ios5 设备上正确运行，但是在 ios5 以下的设备则会 crash。因为是库，所以要考虑到兼容不同版本，于是进一步修改为：

```
    + (void)dismiss
    {
    #if __IPHONE_OS_VERSION_MAX_ALLOWED > __IPHONE_4_3
        if ([sharedDialog respondsToSelector:@selector(presentingViewController)])
        {
            UIViewController *theViewController = [sharedDialog
presentingViewController];
            [theViewController dismissModalViewControllerAnimated:YES];
        }
        else
        {
            UIViewController *theViewController = [sharedDialog
parentViewController];
            [theViewController dismissModalViewControllerAnimated:YES];
        }
    #else
        UIViewController *theViewController = [sharedDialog parentViewController];
        [theViewController dismissModalViewControllerAnimated:YES];
    #endif
    }
```

还有上面那个 Console 的错误提示，解决方法是，在 ASIAuthenticationDialog.m 中找到 -(void)show 这个方法，并把最后一行代码

```
    [[self presentingController] presentModalViewController:self animated:YES];
```

修改为：

```
    UIViewController *theController = [self presentingController];

    #if __IPHONE_OS_VERSION_MAX_ALLOWED > __IPHONE_4_3
        SEL theSelector =
NSSelectorFromString(@"presentModalViewController:animated:");
        NSInvocation *anInvocation = [NSInvocation
```

```
invocationWithMethodSignature:[[theController class]
instanceMethodSignatureForSelector:theSelector]];

        [anInvocation setSelector:theSelector];
        [anInvocation setTarget:theController];

        BOOL                    anim = YES;
        UIViewController    *val = self;
        [anInvocation setArgument:&val atIndex:2];
        [anInvocation setArgument:&anim atIndex:3];

        [anInvocation performSelector:@selector(invoke) withObject:nil afterDelay:1];
    #else

        [theController presentModalViewController:self animated:YES];
    #endif
```

这下就可以正常运行了哟， 我的问题也解决了。关于 ASIHttpRequest 的其它方面，到目前为止还没发现问题。