iOS 5 Programming Cookbook

第八章

网络，JSON，XML 以及 Twitter

版本 1.0

翻译时间：2012-07-2

DevDiv 翻译： kyelup cloudhsu 耐心摩卡
wangli2003j3 xiebaochun dymx101
DevDiv 校对： laigb kyelup
DevDiv 编辑： BeyondVincent

写在前面

目前，移动开发被广大的开发者们看好，并大量的加入移动领域的开发。

鉴于以下原因：

- 国内的相关中文资料缺乏
- 许多开发者对 E 文很是感冒
- 电子版的文档利于技术传播和交流

DevDiv.com 移动开发论坛特此成立了翻译组，翻译组成员具有丰富的移动开发经验和英语翻译水平。组员们利用业余时间，把一些好的相关英文资料翻译成中文，为广大移动开发者尽一点绵薄之力，希望能对读者有些许作用，在此也感谢组员们的辛勤付出。

关于 DevDiv

DevDiv 已成长为国内最具人气的综合性移动开发社区

更多相关信息请访问 DevDiv 移动开发论坛。

技术支持

首先 DevDiv 翻译组对您能够阅读本文以及关注 DevDiv 表示由衷的感谢。

在您学习和开发过程中，或多或少会遇到一些问题。DevDiv 论坛集结了一流的移动专家，我们很乐意与您一起探讨移动开发。如果您有什么问题和技术需要支持的话，请访问 DevDiv 移动开发论坛 或者发送邮件到 BeyondVincent@DevDiv.com，我们将尽力所能及的帮助你。

关于本文的翻译

感谢 kyelup、cloudhsu、耐心摩卡、wangli2003j3、xiebaochun 和 dymx101 对本文的翻译，同时非常感谢 laigb 和 kyelup 在百忙中抽出时间对翻译初稿的认真校验，指出了文章中的错误。才使本文与读者尽快见面。由于书稿内容多，我们的知识有限，尽管我们进行了细心的检查，但是还是会存在错误，这里恳请广大读者批评指正，并发送邮件至 BeyondVincent@devdiv.com，在此我们表示衷心的感谢。

读者查看下面的帖子可以持续关注章节翻译更新情况

[DevDiv 翻译]iOS 5 Programming Cookbook 翻译各章节汇总

另外，我们也为大家准备了 iOS6 新特征的相关内容，请参考下面的帖子：

iOS6 新特征：参考资料和示例汇总------持续更新中

目录

前言

参考帖子
[DevDiv翻译]iOS 5 Programming Cookbook翻译_前言

第 1 章 基础入门

参考帖子

[DevDiv 翻译]iOS 5 Programming Cookbook 翻译_第一章_基础入门

第 2 章 使用控制器和视图

参考帖子

[DevDiv 翻译]iOS 5 Programming Cookbook 翻译 第二章 使用控制器和视图(上)

[DevDiv 翻译]iOS 5 Programming Cookbook 翻译 第二章 使用控制器和视图(下)

第 3 章　构造和使用 Table View

参考帖子

[DevDiv 翻译]iOS 5 Programming Cookbook 翻译_第三章_构造和使用 Table View

第 4 章 Storyboards

参考帖子

[DevDiv 翻译]iOS 5 Programming Cookbook 翻译_第四章 _Storyboards

第 5 章　并发

参考帖子

[DEVDIV 翻译] iOS 5 Programming Cookbook 翻译_第五章_并发

第 6 章 定位核心与地图

参考帖子

[DEVDIV 翻译] iOS 5 Programming Cookbook 翻译_第六章_核心定位与地图

第 7 章 实现手势识别的功能

参考帖子

[DEVDIV 翻译] iOS 5 Programming Cookbook 翻译_第七章_实现手势识别

第 8 章 网络，JSON，XML 以及 Twitter

8.0. 介绍

iOS 的应用，一般是需要通过网络进行数据的交互的，这样你的应用就成为了联机的应用了。这是件非常有意义的事情。这就好比，你可以通过网络查看一组背景图片，然后可以根据你自己的喜好，随时的给你的 iOS 设备设置背景图片。现在无论是电脑还是 iOS 设备都可以随时的通过网络来更换这些背景图片。这个就是 webservice 的魔力，但是这个 webservice 又必须要建立在网络连接的状态上，然后我们就需要了解一些网络，XML，JSON，以及 twitter 相关的知识啦。

iOS SDK 允许我们向网络发送请求，并且能够很方便的通过 NSURLConnection 这个类解析从网络上传递过来的一些数据，我们可以通过 NSJSONSerializatio 和 NSXML 解析器来进行数据的包装以及解析。然后可以通过这些知识来创建一个 Twitter 的内置连接，通过 Twitter 的开发包。

8.1. 通过 NSURLConnection 实现网络数据下载

8.1.1. 问题

你想通过异步的方式实现利用一个 URL 连接从网上下载一个文件

8.1.2. 方案

通过 NSURLConnection 这个类来创建一个异步的请求

8.1.3. 讨论

NSURLConnection 提供了两种方式来实现连接，一种是同步的另一种是异步的，异步的连接将会创建一个新的线程，这个线程将会来负责下载的动作，同步的连接将会堵塞当前的线程，也就是说会造成当前的主线程堵塞，直到这个同步的线程运行完毕将会继续运行主线程。

许多开发者都会认为同步的连接将会堵塞主线程，其实这种观点是错误的。一个同步的连接从它开始运行时就会堵塞主线程。如果你在主线程中创建一个同步连接，没错，主线程会阻塞。但是如果你并不是从主线程开启的一个同步的连接，它将会类似异步的连接一样。因此这种情况并不会堵塞你的主线程。事实上，同步和异步的主要区别就是运行的时候是否会创建一个新的线程，异步的会创建一个新的，而同步的并不会。

为了能够创建一个异步的请求连接，我们需要做如下的操作。

1.创建一个 NSSring 类型的 URL 连接字符串。

2.把 NSString 类型转化成网络可识别的ＮＳＵＲＬ类型。

3.把我们的ＵＲＬ对象赋值到 NSURLRequest 对象中，如果是多个连接请求，请使用 NSMutableURLRequest.

4.创建一个 NSURLConnection 的连接实例，然后把我们定义好的 URL 对象赋值过去。

我们可以创建一个异步的 URL 连接对象通过 sendAsynchronousRequest:queue:completionHandler 这个方法。这个方法的参数如下.

sendAsynchronousRequest

一个 NSURLRequest 类型的请求，这个我们已经讲过。

Queue

一个操作队列，我们可以很轻松的分配和初始化一个操作队列，然后可以根据我们需求添加到这个方法的参数中。

completionHandler

一个临时堵塞的监听捕获器，当我们异步的连接操作完成之后，无论我们的异步操作是否成功，这个对象都能够接收到如下三个参数。

1.一个 NSURLResopne,这个对象是服务器返回给我们的数据包装对象。

2.NSData，可选的，这个是我们通过 URL 请求返回的数据。

3.NSError 类型的对象，如果请求中有错误发生。

这个 sendAsynchronousRequest:queue:completionHandler: 方法如果是添加在主线程上的将不会被调用。因此如果你想执行一个跟 UI 相关的任务，那么你的主线程就会被堵塞。

讲了那么多，让我们还是来看看代码例子吧，我们将会通过访问苹果网站的主页，然后返回一个 string 类型的数据集合。

```
    /* 1 */
NSString *urlAsString = @"http://www.apple.com";
NSURL *url = [NSURL URLWithString:urlAsString];
NSURLRequest *urlRequest = [NSURLRequest requestWithURL:url];
NSOperationQueue *queue = [[NSOperationQueue alloc] init];
[NSURLConnection
 sendAsynchronousRequest:urlRequest
 queue:queue
 completionHandler:^(NSURLResponse *response,
                     NSData *data,
                     NSError *error) {

    if ([data length] >0    &&
        error == nil){
        NSString *html = [[NSString alloc] initWithData:data
                                               encoding:NSUTF8StringEncoding];
        NSLog(@"HTML = %@", html);
    }
    else if ([data length] == 0 &&
             error == nil){
        NSLog(@"Nothing was downloaded.");
    }
    else if (error != nil){
        NSLog(@"Error happened = %@", error);
    }

 }];
```

代码其实很简单的，如果你想保存从网络上下载的数据到你的硬盘中，那么你要在完成的 block 中将使用 NSData 的适当方法。

```
NSString *urlAsString = @"http://www.apple.com";
  NSURL *url = [NSURL URLWithString:urlAsString];
  NSURLRequest *urlRequest = [NSURLRequest requestWithURL:url];
  NSOperationQueue *queue = [[NSOperationQueue alloc] init];

  [NSURLConnection
    sendAsynchronousRequest:urlRequest
    queue:queue
    completionHandler:^(NSURLResponse *response,
                        NSData *data,
                        NSError *error) {

      if ([data length] >0    &&
            error == nil){

        /* Get the documents directory */
        NSString *documentsDir =
        [NSSearchPathForDirectoriesInDomains(NSDocumentDirectory,
                                             NSUserDomainMask,
                                                  YES) objectAtIndex:0];

        /* Append the file name to the documents directory */
        NSString *filePath = [documentsDir
                              stringByAppendingPathComponent:@"apple.html"];

        /* Write the data to the file */
        [data writeToFile:filePath
                  atomically:YES];

        NSLog(@"Successfully saved the file to %@", filePath);

      }
      else if ([data length] == 0 &&
                  error == nil){
        NSLog(@"Nothing was downloaded.");
      }
      else if (error != nil){
        NSLog(@"Error happened = %@", error);
      }

    }];
```

所有的相关知识点我们都提到了，在 iOS SDK 早期的版本中，url 连接通常是以一种协议的形式存在的，但是现在 SDK 5 以后我们把这点个抽出来作为单独的对象来使用，你再也不用担心要去考虑实现的协议的相关方法

8.1.4. 参考

暂无

8.2. 捕获异步连接中超时的问题

8.2.1. 问题

当时创建一个异步的请求的时候，你想设置一个响应超时的值，来完善你的应用程序

8.2.2. 方案

在你创建一个 URL 请求对象并且准备传递给 NSURLConnection，设置一下你的超时值。

8.2.3. 讨论

当初始化 NSURLRequest 这个对象，并且把这个对象绑定到你的 URL 连接对象的时候，你可以使用 requestWithURL:cachePolicy:timeoutInterval:这个方法来根据你的设计，设置你的访问请求的超时值。

比如，你想最大限度的等待 30 秒来同步连接下载苹果主页的内容，你将创建 URL 请求如下：

```
NSURL *url = [NSURL URLWithString:urlAsString];
NSURLRequest *urlRequest =
[NSURLRequest
 requestWithURL:url
 cachePolicy:NSURLRequestReloadIgnoringLocalAndRemoteCacheData
 timeoutInterval:30.0f];
NSOperationQueue *queue = [[NSOperationQueue alloc] init];
[NSURLConnection
 sendAsynchronousRequest:urlRequest
 queue:queue
 completionHandler:^(NSURLResponse *response,
                     NSData *data,
                     NSError *error) {

   if ([data length] >0    &&
       error == nil){
      NSString *html = [[NSString alloc] initWithData:data
                                             encoding:NSUTF8StringEncoding];
      NSLog(@"HTML = %@", html);
   }
   else if ([data length] == 0 &&
            error == nil){
      NSLog(@"Nothing was downloaded.");
   }
   else if (error != nil){
      NSLog(@"Error happened = %@", error);
   }

 }];
```

当你尝试通过解析返回后的数据，但是在解析的时候超过了 30 秒，那么运行的时候这个访问对象将会给你返回一个超时的错误这个错误你可以通过代码块中的 error 这个参数获得具体的错误信息。

8.2.4. 参考

暂无

8.3. 通过 NSURLConnection 创建一个同步的下载

8.3.1. 问题

你想同步的下载一个 URL 的内容

8.3.2. 方案

通过 NSURLConnection 的 sendSynchronousRequest:returningResponse:error: 方法创建一个同步的网络连接。这个方法将会返回一个 NSData 类型的数据

8.3.3. 讨论

在创建一个同步的网络连接的时候我们需要明白一点，并不是是我们的这个同步连接一定会堵塞我们的主线程，如果这个同步的连接是创建在主线程上的，那么这种情况下是会堵塞我们的主线程的，其他的情况下是不一定会堵塞我们的主线程的。如果你仔细阅读了我们第五章的数据你一定会了解一个 GCD 队列的概念。如果你初始化了一个同步的连接，你其实并不会堵塞我们的主线程的。

让我们通过一个实例来看看同步的连接以及堵塞主线程的问题。这个例子中，我们将偿试取回 Yahoo 主页的内容。

```
- (BOOL)                    application:(UIApplication *)application
   didFinishLaunchingWithOptions:(NSDictionary *)launchOptions{

   NSLog(@"We are here...");

   NSString *urlAsString = @"http://www.yahoo.com";
   NSURL *url = [NSURL URLWithString:urlAsString];

   NSURLRequest *urlRequest = [NSURLRequest requestWithURL:url];

   NSURLResponse *response = nil;
   NSError *error = nil;

   NSLog(@"Firing synchronous url connection...");
   NSData *data = [NSURLConnection sendSynchronousRequest:urlRequest
                                                  returningResponse:&response
                                                              error:&error];

   if ([data length] > 0 &&
         error == nil){
      NSLog(@"%lu bytes of data was returned.", (unsigned long)[data length
   }
   else if ([data length] == 0 &&
               error == nil){
      NSLog(@"No data was returned.");
   }
   else if (error != nil){
NSLog(@"Error happened = %@", error);
  NSLog(@"We are done.");

   self.window = [[UIWindow alloc] initWithFrame:
                        [[UIScreen mainScreen] bounds]];
```

```
    self.window.backgroundColor = [UIColor whiteColor];
    [self.window makeKeyAndVisible];
    return YES;
}
```

如果你添加这段代码之后，然后在模拟器中运行，你将会得到如下的效果。

```
We are here...
Firing synchronous url connection...
194472 bytes of data was returned.
We are done.
```

通过上面的代码我们不难发现，其实我们的主线程是一直堵塞着，如果我们同步访问请求的任务没有完成，那么这个主线程，也就是窗体绘画的动作其实是并没有完成的，当我们的同步请求完成之后，我们主线程才开始绘画我们相应的窗体，也就是说段代码是一行一行执行的。

下面让我们看一下，当我们创建一个同步的连接，并且添加到 GCD 的队列池中区，我们又会发现什么

```
- (BOOL)                    application:(UIApplication *)application
    didFinishLaunchingWithOptions:(NSDictionary *)launchOptions{

    NSLog(@"We are here...");

    NSString *urlAsString = @"http://www.yahoo.com";

    NSLog(@"Firing synchronous url connection...");

    dispatch_queue_t dispatchQueue =
        dispatch_get_global_queue(DISPATCH_QUEUE_PRIORITY_DEFAULT, 0);

    dispatch_async(dispatchQueue, ^(void) {

        NSURL *url = [NSURL URLWithString:urlAsString];
        NSURLRequest *urlRequest = [NSURLRequest requestWithURL:url];
        NSURLResponse *response = nil;
        NSError *error = nil;

        NSData *data = [NSURLConnection sendSynchronousRequest:urlRequest
                                                 returningResponse:&response
                                                             error:&error];

        if ([data length] > 0 &&
                error == nil){
            NSLog(@"%lu bytes of data was returned.", (unsigned long)[data length]);
        }
        else if ([data length] == 0 &&
                    error == nil){
            NSLog(@"No data was returned.");
    }
        else if (error != nil){
            NSLog(@"Error happened = %@", error);
        }
    });
 NSLog(@"We are done.");
```

```
    self.window = [[UIWindow alloc] initWithFrame:
                   [[UIScreen mainScreen] bounds]];

    self.window.backgroundColor = [UIColor whiteColor];
    [self.window makeKeyAndVisible];
    return YES;
}
```

启动模拟器，运行代码，我们会得到如下效果

```
We are here...
Firing synchronous url connection...
We are done.
194326 bytes of data was returned.
```

看了如上的代码是不是感觉特别奇怪,怎么 we are done 还是放在最后面的，怎么现在打印到前面的了，这就是我们要证明的，其实同步的程序并不一定会堵塞我们的主线程的。只要我们的这个同步的线程并不是建立在我们的主线程上的，如果我们把他添加到 GCD 队列池中的话，那么我们的主线程也就没有被堵塞。这样同步的概念实际上也可以看做是某种程度上的异步。

8.3.4. 参考

暂无

8.4. 通过 NSMUtableURLRequest 包装 URL 的请求形式

8.4.1. 问题

你想在传给一个请求连接前能适应一个 URL 的不同 HTTP 头及其设置。

8.4.2. 方案

通过 NSMUtableURLRequest 代替 NSURLRequest 对象。

8.4.3. 讨论

通常我们的 URL 请求可能是一组，并不是一个，而且我们的 URL 请求也是会不停的根据流程在变化的，所以我们要仍然使用 NSURLRequest 这个对象是不能随时的进行一个 URL 的变化的，因此我们的 NSMUtableURLRequest 这个对象，可以用来解决我们的这个问题。

让我们开看看分别使用 NSURLRequest 和 NSMUtableURLRequest 实现的网络请求的写法

```
NSString *urlAsString = @"http://www.apple.com";
NSURL *url = [NSURL URLWithString:urlAsString];
NSMutableURLRequest *urlRequest = [NSMutableURLRequest requestWithURL:url];
[urlRequest setTimeoutInterval:30.0f];
```

让我们再来看一下 NSMutableURLRequest 的写法

```
NSString *urlAsString = @"http://www.apple.com";
NSURL *url = [NSURL URLWithString:urlAsString];
NSMutableURLRequest *urlRequest = [NSMutableURLRequest new];
[urlRequest setTimeoutInterval:30.0f];
[urlRequest setURL:url];
```

通过如上两种不同写法的实例，我们不难发现，NSMutableURLRequest 的创建是可以不用指定一个具体的 URL 的，可以根据我们的需求，然后再设置的。

这样我们就能很容易的根据需求来调整我们的访问请求对像了。

8.4.4. 参考

暂无

8.5. 通过 NSURLConnection 发送一个 HTTP GET 请求

8.5.1. 问题

你想通过 Http 协议向服务器发送一个 Get 的包装请求，并在这个请求中添加了一些请求参数.

8.5.2. 方案

在 URL 规格后面使用问号来补充说明你想做的事。

8.5.3. 讨论

向远程服务器发送一个 GET 请求，然后解析返回的数据。通常一个 GET 请求是添加了一些参数的，这些参数一般是添加在 URL 请求中。

我准备了一个 GET 形式的 webservice 接口，你可以通过 http://pixolity.com/get.php 来进行请求。

如果你通过浏览器打开这个链接，你将会看到如下效果.

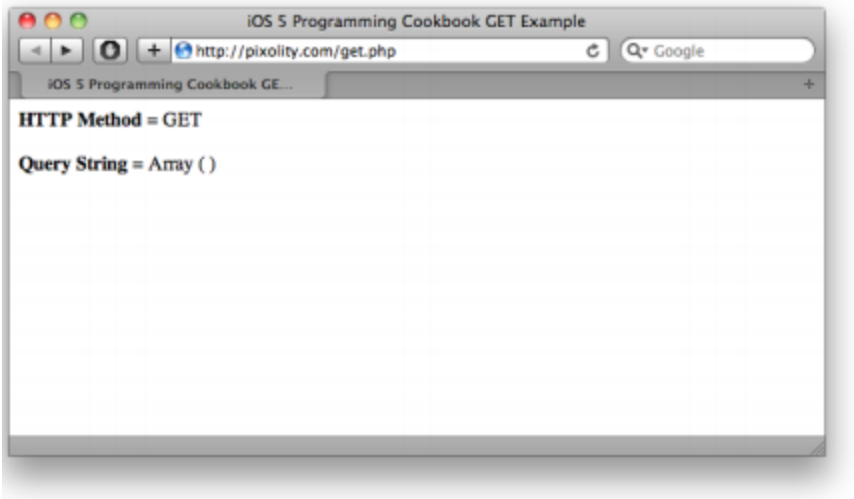

图 8-1 通过浏览器打开的 GET 请求.

如上是我们直接访问这个 GET 请求，我们并没有添加一些参数，然后我们给这个 URL 添加一些请求的参数，再次访问可以看到如下下过。添加参数的 URL 连

接.(http://pixolity.com/get.php?param1=First&param2=Second)

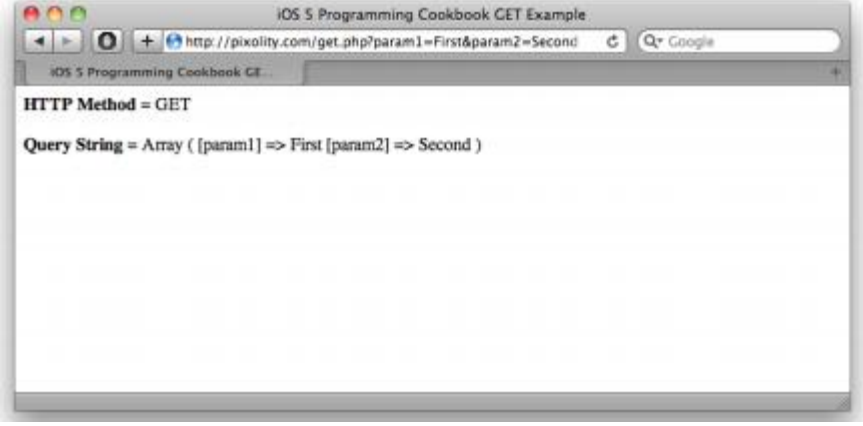

图 8-2 添加请求参数的访问效果

很明显，添加参数之后的 GET 请求，实际上是在服务器上做了一个计算处理的过程的。看如下代码，我们需要如何在我们的代码中修改。

```
/* URL = http://pixolity.com/get.php?param1=First&param2=Second */
NSString *urlAsString = @"http://pixolity.com/get.php";
urlAsString = [urlAsString stringByAppendingString:@"?param1=First"];
urlAsString = [urlAsString stringByAppendingString:@"&param2=Second"];

NSURL *url = [NSURL URLWithString:urlAsString];

NSMutableURLRequest *urlRequest = [NSMutableURLRequest requestWithURL:url];
[urlRequest setTimeoutInterval:30.0f];
[urlRequest setHTTPMethod:@"GET"];

NSOperationQueue *queue = [[NSOperationQueue alloc] init];

[NSURLConnection
 sendAsynchronousRequest:urlRequest
 queue:queue
 completionHandler:^(NSURLResponse *response,
                     NSData *data,
                     NSError *error) {

    if ([data length] >0    &&
        error == nil){
        NSString *html = [[NSString alloc] initWithData:data
                                               encoding:NSUTF8StringEncoding];
        NSLog(@"HTML = %@", html);
    }
    else if ([data length] == 0 &&
             error == nil){
        NSLog(@"Nothing was downloaded.");
    }
    else if (error != nil){
        NSLog(@"Error happened = %@", error);
    }
  }];
```

编译运行我们的程序我们将会得到如下的效果.

```
TML =
<html>
	<head>
		<title>iOS 5 Programming Cookbook GET Example</title>
	</head>
	<body>
<b>HTTP Method</b> = GET<br/><br/><b>Query String</b> = Array
(
	[param1] => First
	[param2] => Second
)
	</body>
</html>
```

唯一一点值得我们需要注意的就是，发送的带参数的 GET 请求，第一个参数前面必须要添加一个“？”，然后每个参数之间再用"&"分开，这样就表示传递了多个参数

8.5.4. 参考

暂无.

8.6. 通过 NSURLConnection 发送一个 POST 表单请求

8.6.1. 问题

你需要通过 HTTP 协议发送一个 POST 请求，这个请求中包含一些参数

8.6.2. 方案

就如同 GET 请求一样，你需要利用 NSURLConnection 来创建一个 POST 请求，不过这个请求的参数不是添加在 URL 请求中。

8.6.3. 讨论

我已经创建了一个 webservice 端， http://pixolity.com/post.php，如果你通过浏览器打开，你将会看到如下效果

图 8-3 通过浏览器打开的 POST 请求.

由于这个 webservice 需要几个参数，而且这些参数又必须要以 POST 的形式包装。那么

我们需要按照如下的代码来实现。

```
NSString *urlAsString = @"http://pixolity.com/post.php";
urlAsString = [urlAsString stringByAppendingString:@"?param1=First"];
urlAsString = [urlAsString stringByAppendingString:@"&param2=Second"];
NSURL *url = [NSURL URLWithString:urlAsString];
NSMutableURLRequest *urlRequest = [NSMutableURLRequest requestWithURL:url];
[urlRequest setTimeoutInterval:30.0f];
[urlRequest setHTTPMethod:@"POST"];
NSString *body = @"bodyParam1=BodyValue1&bodyParam2=BodyValue2";
[urlRequest setHTTPBody:[body dataUsingEncoding:NSUTF8StringEncoding]];
NSOperationQueue *queue = [[NSOperationQueue alloc] init];
[NSURLConnection
 sendAsynchronousRequest:urlRequest
 queue:queue
 completionHandler:^(NSURLResponse *response,
                     NSData *data,
                     NSError *error) {

    if ([data length] >0    &&
          error == nil){
       NSString *html = [[NSString alloc] initWithData:data
                                              encoding:NSUTF8StringEncoding];
       NSLog(@"HTML = %@", html);
    }
    else if ([data length] == 0 &&
                error == nil){
       NSLog(@"Nothing was downloaded.");
    }
    else if (error != nil){
       NSLog(@"Error happened = %@", error);
    }

 }];
```

编译运行，你将会在日志中看到如下内容。

```
HTML =
<html>
      <head>
           <title>iOS 5 Programming Cookbook POST Example</title>
      </head>
      <body>
<b>HTTP Method</b> = POST<br/><br/><b>Query String</b> = Array
(
      [param1] => First
      [param2] => Second
)
<br/><br/><b>Body Parameters</b> = Array
(
      [bodyParam1] => BodyValue1
      [bodyParam2] => BodyValue2
)
      </body>
</html>
```

8.6.4. 参考

暂无

8.7. 通过 NSURLConnection 发生 HTTP DELETE 请求

8.7.1. 问题

你希望通过 HTTP 协议像服务器发送一个删除资源的一个 URL 连接。同时你也可以给这个请求添加参数/.

8.7.2. 方案

就像发送 GET 和 POST 方法一样，我们也是使用 NSURLConnection 来发送请求。我们应该明确的将 URL 设置成 DELETE.

8.7.3. 讨论

我创建了一个 webservice 的 Delete 请求， http://pixolity.com/delete.php, 你可以直接通过浏览器访问，你将看到如下的效果

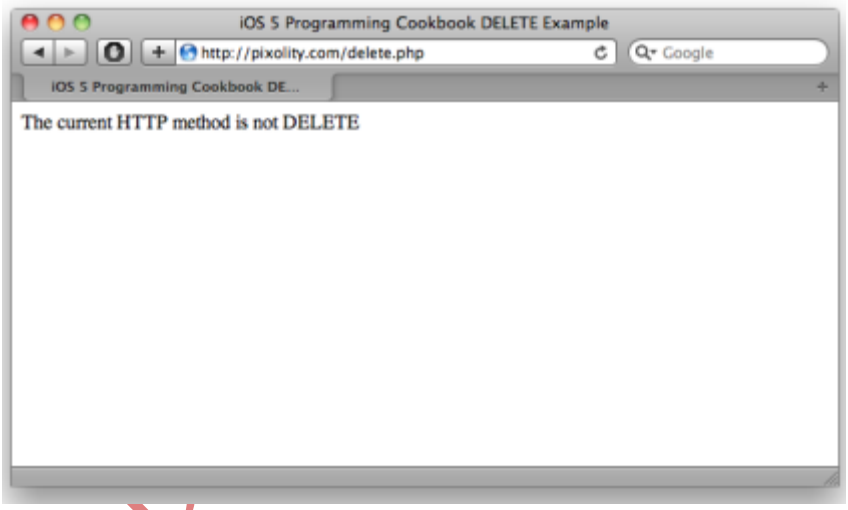

图 8-4 直接通过浏览器访问 DELETE 请求

下面我们看一下用代码怎么来具体实现。

```
NSString *urlAsString = @"http://pixolity.com/delete.php";
urlAsString = [urlAsString stringByAppendingString:@"?param1=First"];
urlAsString = [urlAsString stringByAppendingString:@"&param2=Second"];
NSURL *url = [NSURL URLWithString:urlAsString];
NSMutableURLRequest *urlRequest = [NSMutableURLRequest requestWithURL:url];
[urlRequest setTimeoutInterval:30.0f];
[urlRequest setHTTPMethod:@"DELETE"];
NSString *body = @"bodyParam1=BodyValue1&bodyParam2=BodyValue2";
[urlRequest setHTTPBody:[body dataUsingEncoding:NSUTF8StringEncoding]];
NSOperationQueue *queue = [[NSOperationQueue alloc] init];
[NSURLConnection
  sendAsynchronousRequest:urlRequest
  queue:queue
  completionHandler:^(NSURLResponse *response,
```

```
                                NSData *data,
                                NSError *error) {

    if ([data length] >0    &&
            error == nil){
        NSString *html = [[NSString alloc] initWithData:data
                                                encoding:NSUTF8StringEncoding];
        NSLog(@"HTML = %@", html);
    }
    else if ([data length] == 0 &&
                 error == nil){
        NSLog(@"Nothing was downloaded.");
    }
   else if (error != nil){
        NSLog(@"Error happened = %@", error);
    }

 }];
```

编译运行，我们将会看到如下的效果。

```
HTML =
<html>
      <head>
           <title>iOS 5 Programming Cookbook DELETE Example</title>
      </head>
      <body>
<b>HTTP Method</b> = DELETE<br/><br/><b>Query String</b> = Array
(
      [param1] => First
      [param2] => Second
)
<br/><br/><b>Body Parameters</b> = Array
(
      [bodyParam1] => BodyValue1
      [bodyParam2] => BodyValue2
)
      </body>
</html>
```

8.7.4. 参考

暂无

8.8. 通过 NSURLConnection 发送 HTTP PUT 请求

8.8.1. 问题

你想向服务器发送 HTTP PUT 请求，并且希望在请求的同时添加一些参数.

8.8.2. 方案

就像发送 post,get,delete 请求一样，我们仍然采用 NSURLConnection 实现。我们应该明确的将 URL 设置成 PUT。

8.8.3. 讨论

你可以通过 http://pixolity.com/put.php webservice 请求直接在浏览器中访问，你将会看到如下效果。

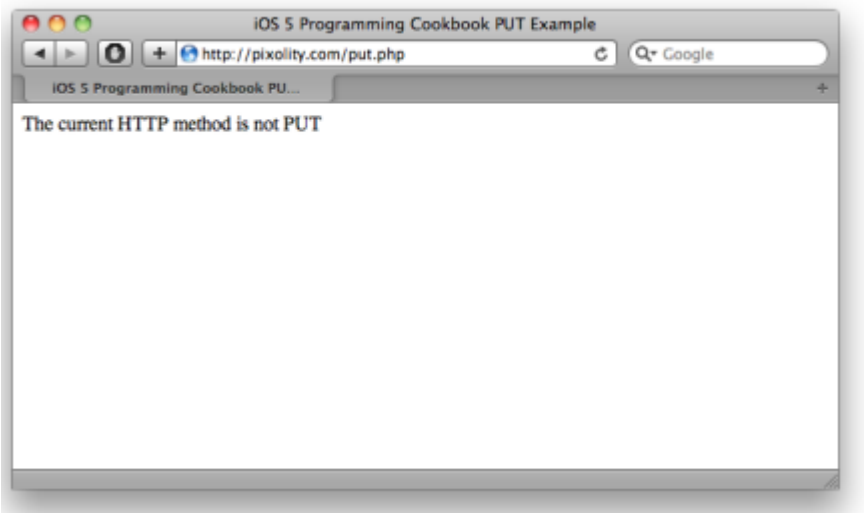

图 8-5 直接通过浏览器访问的 PUT 请求

下面我们看下代码具体怎么操作。

```
NSString *urlAsString = @"http://pixolity.com/put.php";
urlAsString = [urlAsString stringByAppendingString:@"?param1=First"];
urlAsString = [urlAsString stringByAppendingString:@"&param2=Second"];
NSURL *url = [NSURL URLWithString:urlAsString];
NSMutableURLRequest *urlRequest = [NSMutableURLRequest requestWithURL:url];
[urlRequest setTimeoutInterval:30.0f];
[urlRequest setHTTPMethod:@"PUT"];
NSString *body = @"bodyParam1=BodyValue1&bodyParam2=BodyValue2";
[urlRequest setHTTPBody:[body dataUsingEncoding:NSUTF8StringEncoding]];
NSOperationQueue *queue = [[NSOperationQueue alloc] init];
[NSURLConnection
 sendAsynchronousRequest:urlRequest
 queue:queue
 completionHandler:^(NSURLResponse *response,
                     NSData *data,
                     NSError *error) {
 if ([data length] >0    &&
          error == nil){
      NSString *html = [[NSString alloc] initWithData:data
                                              encoding:NSUTF8StringEncoding];
      NSLog(@"HTML = %@", html);
   }
   else if ([data length] == 0 &&
            error == nil){
      NSLog(@"Nothing was downloaded.");
   }
   else if (error != nil){
      NSLog(@"Error happened = %@", error);
   }

 }];
```

编译运行，你会在日志区看到如下的效果。

```
HTML =
```

```
<html>
	<head>
		<title>iOS 5 Programming Cookbook PUT Example</title>
	</head>
	<body>
<b>HTTP Method</b> = PUT<br/><br/><b>Query String</b> = Array
(
	[param1] => First
	[param2] => Second
)
<br/><br/><b>Body Parameters</b> = Array
(
	[bodyParam1] => BodyValue1
	[bodyParam2] => BodyValue2
)
	</body>
</html>
```

8.8.4. 参考

暂无

8.9. 把 Array 和 Dictionaries 序列化成 JSON 对象

8.9.1. 问题

你想把一 Array 和 dictionary 序列化成 JSON 对象。然后方便你在网络中传输。

8.9.2. 方案

通过 NSJSONSerialization 这个类的 ataWithJSONObject:options:error:方法来实现。

8.9.3. 讨论

下面我们将通过代码看看具体的实现过程。

```
NSMutableDictionary *dictionary = [[NSMutableDictionary alloc] init];
[dictionary setValue:@"Anthony"
              forKey:@"First Name"];
[dictionary setValue:@"Robbins"
              forKey:@"Last Name"];
[dictionary setValue:[NSNumber numberWithUnsignedInteger:51]
              forKey:@"Age"];
NSArray *arrayOfAnthonysChildren = [[NSArray alloc]
                                    initWithObjects:
                                    @"Anthony's Son 1",
                                    @"Anthony's Daughter 1",
                                    @"Anthony's Son 2",
                                    @"Anthony's Son 3",
                                    @"Anthony's Daughter 2",
                                    nil];
[dictionary setValue:arrayOfAnthonysChildren
              forKey:@"children"];
```

如上代码我们已经创建了一个 Dictionray 对象，并且组装了一些数据。
同时也创建了一个 NSArray 的对象，组装了一些数据。

```
NSError *error = nil;
NSData *jsonData = [NSJSONSerialization
                    dataWithJSONObject:dictionary
                    options:NSJSONWritingPrettyPrinted
                    error:&error];
if ([jsonData length] > 0 &&
    error == nil){

   NSLog(@"Successfully serialized the dictionary into data = %@", jsonData);

}
else if ([jsonData length] == 0 &&
         error == nil){

   NSLog(@"No data was returned after serialization.");

}
else if (error != nil){

   NSLog(@"An error happened = %@", error);

}
```

如上代码 dataWithJSONObject:options:error:这个方法将会返回一个 NSDate 类型的数据，

因此你可以很简单的使用这个数据把他转化成一个 NString 类型的，然后把他打印在控制台上以便测试查看。如下代码是一个完整的实例

```
NSMutableDictionary *dictionary = [[NSMutableDictionary alloc] init];
[dictionary setValue:@"Anthony"
                forKey:@"First Name"];
[dictionary setValue:@"Robbins"
                forKey:@"Last Name"];
[dictionary setValue:[NSNumber numberWithUnsignedInteger:51]
                forKey:@"Age"];
NSArray *arrayOfAnthonysChildren = [[NSArray alloc]
                                    initWithObjects:
                                    @"Anthony's Son 1",
                                    @"Anthony's Daughter 1",
                                    @"Anthony's Son 2",
                                    @"Anthony's Son 3",
                                    @"Anthony's Daughter 2",
                                    nil];
[dictionary setValue:arrayOfAnthonysChildren
                forKey:@"children"];
NSError *error = nil;
NSData *jsonData = [NSJSONSerialization
                    dataWithJSONObject:dictionary
                    options:NSJSONWritingPrettyPrinted
                    error:&error];
if ([jsonData length] > 0 &&
    error == nil){

   NSLog(@"Successfully serialized the dictionary into data.");
   NSString *jsonString = [[NSString alloc] initWithData:jsonData
                                                 encoding:NSUTF8StringEncoding];
   NSLog(@"JSON String = %@", jsonString);

}
else if ([jsonData length] == 0 &&
         error == nil){

   NSLog(@"No data was returned after serialization.");

}
else if (error != nil){

   NSLog(@"An error happened = %@", error);

}
```

编译运行，你将会看到如下效果，那么我们的操作就已经完成了。

```
Successfully serialized the dictionary into data.
JSON String = {
   "Last Name" : "Robbins",
   "First Name" : "Anthony",
   "children" : [
      "Anthony's Son 1",
      "Anthony's Daughter 1",
      "Anthony's Son 2",
      "Anthony's Son 3",
      "Anthony's Daughter 2"
   ],
   "Age" : 51
}
```

8.9.4. 参考

暂无

8.10. 把 JSON 数据转化成 Arrays 或者 Dictionaries

8.10.1. 问题

你有一个 JSON 的数据对象，希望把这个 JSON 数据解析出来放在数据或者字典里面保存

8.10.2. 方案

通过 NSJSONSerialization 这个类的 JSONObjectWithData:options:error:方法来实现。

8.10.3. 讨论

通过上一个章节我们已经知道如何把数组或者字典转化成 JSON 对象，其实是很简单的一个过程，那么把 JSON 对象转化成数组或者字典也是很简单的一件事情，请看如下代码

```
/* Now try to deserialize the JSON object into a dictionary */
error = nil;
id jsonObject = [NSJSONSerialization
                 JSONObjectWithData:jsonData
                 options:NSJSONReadingAllowFragments
                 error:&error];
if (jsonObject != nil &&
    error == nil){

  NSLog(@"Successfully deserialized...");

  if ([jsonObject isKindOfClass:[NSDictionary class]]){

    NSDictionary *deserializedDictionary = (NSDictionary *)jsonObject;
    NSLog(@"Dersialized JSON Dictionary = %@", deserializedDictionary);

  }
  else if ([jsonObject isKindOfClass:[NSArray class]]){

    NSArray *deserializedArray = (NSArray *)jsonObject;
    NSLog(@"Dersialized JSON Array = %@", deserializedArray);

  }
  else {
    /* Some other object was returned. We don't know how to deal
      with this situation as the deserializer only returns dictionaries
      or arrays */
  }
}
else if (error != nil){
  NSLog(@"An error happened while deserializing the JSON data.");
}
```

如上的代码是一个初略的代码示例。下面我们看一个完整的实例。

```
NSMutableDictionary *dictionary = [[NSMutableDictionary alloc] init];

   [dictionary setValue:@"Anthony"
                 forKey:@"First Name"];

   [dictionary setValue:@"Robbins"
                 forKey:@"Last Name"];

   [dictionary setValue:[NSNumber numberWithUnsignedInteger:51]
                 forKey:@"Age"];

   NSArray *arrayOfAnthonysChildren = [[NSArray alloc]
                                       initWithObjects:
                                       @"Anthony's Son 1",
                                       @"Anthony's Daughter 1",
                                       @"Anthony's Son 2",
                                       @"Anthony's Son 3",
                                       @"Anthony's Daughter 2",
                                       nil];

   [dictionary setValue:arrayOfAnthonysChildren
                 forKey:@"children"];

   NSError *error = nil;
   NSData *jsonData = [NSJSONSerialization
                       dataWithJSONObject:dictionary
                       options:NSJSONWritingPrettyPrinted
                       error:&error];

   if ([jsonData length] > 0 &&
       error == nil){

      NSLog(@"Successfully serialized the dictionary into data.");

      /* Now try to deserialize the JSON object into a dictionary */
      error = nil;
id jsonObject = [NSJSONSerialization
  JSONObjectWithData:jsonData
                       options:NSJSONReadingAllowFragments
                       error:&error];

      if (jsonObject != nil &&
          error == nil){

         NSLog(@"Successfully deserialized...");

         if ([jsonObject isKindOfClass:[NSDictionary class]]){

            NSDictionary *deserializedDictionary = (NSDictionary *)jsonObject;
            NSLog(@"Dersialized JSON Dictionary = %@", deserializedDictionary);

         }
         else if ([jsonObject isKindOfClass:[NSArray class]]){

            NSArray *deserializedArray = (NSArray *)jsonObject;
            NSLog(@"Dersialized JSON Array = %@", deserializedArray);

         }
```

```
        else {
          /* Some other object was returned. We don't know how to deal
            with this situation as the deserializer only returns dictionaries
           or arrays */
        }

    }
    else if (error != nil){
       NSLog(@"An error happened while deserializing the JSON data.");
    }

}
else if ([jsonData length] == 0 &&
             error == nil){

    NSLog(@"No data was returned after serialization.");

}
else if (error != nil){

    NSLog(@"An error happened = %@", error);

}
```

8.10.4. 参考

暂无

8.11. 集成 Twitter 的功能到你的应用中

8.11.1. 问题

你想集成 Twitter 的功能到你的 iOS 应用中去

8.11.2. 方案

使用 Twitter 框架库包

8.11.3. 讨论

为了能够集成 Twitter 的功能到你的应用程序中，我们需要添加 Twitter 的框架包，请参照如下方式来添加

1.点击你的功能图标。
2.选择 target 选项、
3.在右段中选择 Build Phases
4.打开 Link Binary 然后可以看见你的工程目前应用的包。
5.点击“+”号，
6.通过弹出框，然后输入 Twitter，然后找到 Twitter 然后添加进来。
下面我们来看一下代码的操作

```
#import <UIKit/UIKit.h>
#import <Twitter/Twitter.h>
@interface Integrating_Twitter_Functionality_Into_Your_AppsViewController
                : UIViewController
@end
```

然后我们需要添加一个 TWTeetCOmposeViewController 的实例对象，代码如下

```
#import <UIKit/UIKit.h>
#import <Twitter/Twitter.h>
@interface Integrating_Twitter_Functionality_Into_Your_AppsViewController
                : UIViewController
@property (nonatomic, strong) TWTweetComposeViewController *twitterController;
@end
```

然后我们需要对这个变量属性进行 synthesize.

```
#import "Integrating_Twitter_Functionality_Into_Your_AppsViewController.h"
@implementation
   Integrating_Twitter_Functionality_Into_Your_AppsViewController
@synthesize twitterController;
...

- (void)viewDidLoad{
   [super viewDidLoad];

   self.twitterController = [[TWTweetComposeViewController alloc] init];
   [self.twitterController setInitialText:@"Your Tweet Goes Here"];
   [self.navigationController presentModalViewController:self.twitterController
                                                 animated:YES];

}
```

然后编译运行，你将会在你的模拟器中看到如下的效果,也有可能看到 8-7 实例的 setting 项。

图 8-6 添加的一个 Twitter 窗体

图 8-7 Twitter setting 的界面

当我们添加了 Twitter 的认真信息之后，重启我们的应用程序，我们将会仍然看到图 8-6 这个图。

下面我们将会在添加一些其他的信息，代码类似如下

```
- (void)viewDidLoad{
    [super viewDidLoad];
        self.twitterController = [[TWTweetComposeViewController alloc] init];

    NSString *text =
    @"Anthony Robbins at Unleash the Power Within (UPW) in London";

    [self.twitterController setInitialText:text];

    UIImage *anthonyRobbins = [UIImage imageNamed:@"Anthony Robbins.jpg"];
    [self.twitterController addImage:anthonyRobbins];

    NSURL *url = [NSURL URLWithString:@"http://www.tonyrobbins.com"];
    [self.twitterController addURL:url];


    [self.navigationController presentModalViewController:self.twitterController
                                                  animated:YES];

}
```

编译，运行，我们将会看到如下的效果

下面我们来看一下常用的两个方法
TWTweetComposeViewControllerResultCancelled
这个方法是当用户在进入 Twitter 界面的时候,然后点击了取消的事件.

TWTweetComposeViewControllerResultDone
这个方法是进入 Twitter 界面,然后正常的发送了界面中的信息

下面这段代码介绍了具体如何使用

```
- (void)viewDidLoad{
    [super viewDidLoad];

    self.twitterController = [[TWTweetComposeViewController alloc] init];

    __weak
        Integrating_Twitter_Functionality_Into_Your_AppsViewController
            *weakSelf = self;

    [self.twitterController setCompletionHandler:
     ^(TWTweetComposeViewControllerResult result){

         Integrating_Twitter_Functionality_Into_Your_AppsViewController
           *strongSelf = weakSelf;

         switch (result){
            case TWTweetComposeViewControllerResultDone:{
               /* The Tweet was submitted successfully.
                 Will be dismissed automatically */
               break;
            }
            case TWTweetComposeViewControllerResultCancelled:{
               if (strongSelf != nil){
                  [strongSelf.twitterController
                    dismissModalViewControllerAnimated:YES];
               }
               break;
            }
         }

    }];

    NSString *text =
    @"Anthony Robbins at Unleash the Power Within (UPW) in London";

    [self.twitterController setInitialText:text];

    UIImage *anthonyRobbins = [UIImage imageNamed:@"Anthony Robbins.jpg"];
    [self.twitterController addImage:anthonyRobbins];

    NSURL *url = [NSURL URLWithString:@"http://www.tonyrobbins.com"];
    [self.twitterController addURL:url];


    [self.navigationController presentModalViewController:self.twitterController
                                                  animated:YES];

}
```

8.12. 通过 NSXMLParser 来解析 XML

8.12.1. 问题

你想要解析 XML。

8.12.2. 方案

使用 NSXMLParser 类。

8.12.3. 讨论

我们一般是通过实现 NSXMLParser 的协议来进行 XML 内容的解析，首先让我们创建一个简单的 XML 文件，包含如下内容（把他命名为 MyXML.xml，然后添加到你的工程中）：

```
<?xml version="1.0" encoding="UTF-8"?>
<root>

   <person id="1">
      <firstName>Anthony</firstName>
      <lastName>Robbins</lastName>
      <age>51</age>
   </person>

   <person id="2">
      <firstName>Richard</firstName>
      <lastName>Branson</lastName>
      <age>61</age>
   </person>

</root>
```

下面让我们定义一个 NSXMLParser 类型的属性。

```
#import <UIKit/UIKit.h>
@interface Parsing_XML_with_NSXMLParserAppDelegate
                : UIResponder <UIApplicationDelegate, NSXMLParserDelegate>
@property (nonatomic, strong) UIWindow *window;
@property (nonatomic, strong) NSXMLParser *xmlParser;
@end
```

可以看到，我定义了一个 XML parser app delegate，并且尊循 NSXMLParserDelegate 协议，该协议是作为 NSXMLParser 解析 xml 需要用到的代理。下面继续实现我们的 XML 解析：

```
#import "Parsing_XML_with_NSXMLParserAppDelegate.h"
@implementation Parsing_XML_with_NSXMLParserAppDelegate
@synthesize window = _window;
@synthesize xmlParser;
...
```

现在我们从磁盘中读取出 MyXML.xml 文件内容，并传给 NSXMLParser。

```
- (BOOL)                    application:(UIApplication *)application
   didFinishLaunchingWithOptions:(NSDictionary *)launchOptions{

   NSString *xmlFilePath = [[NSBundle mainBundle] pathForResource:@"MyXML"
                                                           ofType:@"xml"];

   NSData *xml = [[NSData alloc] initWithContentsOfFile:xmlFilePath];

   self.xmlParser = [[NSXMLParser alloc] initWithData:xml];
```

```
    self.xmlParser.delegate = self;
    if ([self.xmlParser parse]){
        NSLog(@"The XML is parsed.");
    } else{
        NSLog(@"Failed to parse the XML");
    }

    self.window = [[UIWindow alloc] initWithFrame:
                   [[UIScreen mainScreen] bounds]];

    self.window.backgroundColor = [UIColor whiteColor];
    [self.window makeKeyAndVisible];
    return YES;
}
```

首先把文件内容读取到一个 NSData 实例对象中，然后使用 initWithData:来初始化我们的 XML parser，并把我们从 xml 文件中读取出来的数据传递进去。之后我们可以调用 XML parser 的 parse 方法来开始解析处理。这个方法会阻塞当前线程，直至解析处理结束。如果你需要解析的 XML 文件非常大，强烈建议使用一个全局的 dispatch 队列来进行解析。

为了知道如何来解析 XML 文件，我们需要了解定义在 NSXMLParserDelegate 协议中的代理方法和它们的职责：

parserDidStartDocument:

解析开始的时候调用该方法。

parserDidEndDocument:

解析结束的时候调用该方法。

parser:didStartElement:namespaceURI:qualifiedName:attributes:

在 XML document 中，当解析器在解析的时候遇到了一个新的 element 时会被调用该方法。

parser:didEndElement:namespaceURI:qualifiedName:

当前节点结束之后会调用。

parser:foundCharacters:

当解析器在解析文档内容的时候被调用。

在使用这些协议方法的时候，我们可以为 XML 对象创建一个对象模型。下面我们定义一个对象来代表 XML element，类名叫做 XMLElement，代码如下：

```
#import <Foundation/Foundation.h>

@interface XMLElement : NSObject

@property (nonatomic, strong) NSString *name;
@property (nonatomic, strong) NSString *text;
@property (nonatomic, strong) NSDictionary *attributes;
@property (nonatomic, strong) NSMutableArray *subElements;
@property (nonatomic, weak) XMLElement *parent;

@end
```

下面来实现 XMLElement 类：

```
#import "XMLElement.h"

@implementation XMLElement

@synthesize name;
@synthesize text;
@synthesize attributes;
@synthesize subElements;
@synthesize parent;
- (NSMutableArray *) subElements{
   if (subElements == nil){
      subElements = [[NSMutableArray alloc] init];
   }
   return subElements;
}

@end
```

我只想当访问 subElements 数组的时候，如果该数组是 nil，才进行初始化。因此我把这个属性的内存分配和初始化代码放到了它的 getter 方法中。如果说一个 XML element 没有子 elements，那么我们永远都不会使用到这个属性，因此这里也就不会为那个 element 分配内存和进行初始化工作。这种技术叫做 lazy allocation。

现在我们定义一个 XMLElement 实例，叫做 rootElement。我们的计划是开始解析处理，向下获取 XML 文件内容并使用 delegate 方法进行解析，直至成功解析完整个文件。

```
#import <UIKit/UIKit.h>
@class XMLElement;
@interface Parsing_XML_with_NSXMLParserAppDelegate
                : UIResponder <UIApplicationDelegate, NSXMLParserDelegate>
@property (nonatomic, strong) UIWindow *window;
@property (nonatomic, strong) NSXMLParser *xmlParser;
@property (nonatomic, strong) XMLElement *rootElement;
@property (nonatomic, strong) XMLElement *currentElementPointer;
@end
```

currentElementPointer 代表此刻在 XML 文件结构中，正在解析的 XML element，因此，可以上移或者下移这个结构，来当做我们解析的文件内容。它是一个简单的指针，反之，rootElement 总是指向 XML 文件的 root element。下面我们将添加新的属性：

```
#import "Parsing_XML_with_NSXMLParserAppDelegate.h"
#import "XMLElement.h"
@implementation Parsing_XML_with_NSXMLParserAppDelegate
@synthesize window = _window;
@synthesize xmlParser;
@synthesize rootElement;
@synthesize currentElementPointer;
....
```

然后我们开始解析处理。我们想要关注的第一个方法就是 parserDidStartDocument：方法。在这个方法中，我们进行简单的重置一切：

```
- (void)parserDidStartDocument:(NSXMLParser *)parser{
   self.rootElement = nil;
```

```
    self.currentElementPointer = nil;
}
```

下一个方法就是 parser:didStartElement:namespaceURI:qualifiedName:attributes:方法。在这个方法中，如果 root element 没有被创建，会被创建，并且开始解析一个新的 element，我们会计算它在 XML 结构中的位置，并在当前 element 中添加一个新的 element。

```
- (void)            parser:(NSXMLParser *)parser
          didStartElement:(NSString *)elementName
              namespaceURI:(NSString *)namespaceURI
             qualifiedName:(NSString *)qName
                attributes:(NSDictionary *)attributeDict{

    if (self.rootElement == nil){
        /* We don't have a root element. Create it and point to it */
        self.rootElement = [[XMLElement alloc] init];
        self.currentElementPointer = self.rootElement;
    } else {
        /* Already have root. Create new element and add it as one of
          the subelements of the current element */
        XMLElement *newElement = [[XMLElement alloc] init];
        newElement.parent = self.currentElementPointer;
        [self.currentElementPointer.subElements addObject:newElement];
        self.currentElementPointer = newElement;
    }
    self.currentElementPointer.name = elementName;
    self.currentElementPointer.attributes = attributeDict;

}
```

下一步，就是 parser:foundCharacters: 这个方法了。这个方法将会在解析 element 的时候调用多次，因此我们需要确保已经为多次进入该方法做好了准备。例如，如果一个 element 的文本有 4000 个字符长度，解析器在第一次解析时，最多只能解析 1000 个字符，之后在解析当前 element 时，调用 parser:foundCharacters:方法，每次都是 1000，因此需要 4 次：

```
- (void)            parser:(NSXMLParser *)parser
          foundCharacters:(NSString *)string{

    if ([self.currentElementPointer.text length] > 0){
        self.currentElementPointer.text =
          [self.currentElementPointer.text stringByAppendingString:string];
    } else {
        self.currentElementPointer.text = string;
    }

}
```

下一个需要关注的方法就是 parser:didEndElement:namespaceURI:qualifiedName:方法，当解析至某个 element 尾部时，会调用该方法。在这里，我们只需要把当前 element 指针指向当前 element 的上一级：

```
- (void)            parser:(NSXMLParser *)parser
            didEndElement:(NSString *)elementName
```

```
            namespaceURI:(NSString *)namespaceURI
           qualifiedName:(NSString *)qName{

    self.currentElementPointer = self.currentElementPointer.parent;

}
```

最终，我们需要处理 parserDidEndDocument 这个方法。我们需要 dispose currentElementPointe 属性。

```
- (void)parserDidEndDocument:(NSXMLParser *)parser{
    self.currentElementPointer = nil;
}
```

上面就是所有的实现内容，下面我们开始解析文档。

```
- (BOOL)                    application:(UIApplication *)application
    didFinishLaunchingWithOptions:(NSDictionary *)launchOptions{

    NSString *xmlFilePath = [[NSBundle mainBundle] pathForResource:@"MyXML"
                                                            ofType:@"xml"];

    NSData *xml = [[NSData alloc] initWithContentsOfFile:xmlFilePath];

    self.xmlParser = [[NSXMLParser alloc] initWithData:xml];
    self.xmlParser.delegate = self;
    if ([self.xmlParser parse]){
        NSLog(@"The XML is parsed.");

        /* self.rootElement is now the root element in the XML */

    } else{
        NSLog(@"Failed to parse the XML");
    }

    self.window = [[UIWindow alloc] initWithFrame:
                   [[UIScreen mainScreen] bounds]];

    self.window.backgroundColor = [UIColor whiteColor];
    [self.window makeKeyAndVisible];
    return YES;
}
```

现在你就可以使用 rootElement 属性来遍历 XML 结构了。

8.12.4. 参考

暂无