说明：译者在做app开发时，因为页面的javascript文件比较大导致加载速度很慢，所以想把javascript文件打包在app里，当UIWebView需要加载该脚本时就从app本地读取，但UIWebView并不支持加载本地资源。最后从下文中找到了解决方法，第一次翻译，难免有误，大家多多指教。

iCab Mobile(一款iOS平台的网页浏览器)要实现一个拦截管理器来过滤页面上的广告及其他东西。它有一个简单的基于URL过滤规则的列表（通常由用户维护），当页面包含的资源（图片、js以及css等），文件的URL存在于规则列表中时，资源就不会被加载。

但看一下UIWebView类的API，会发现我们没有办法知道UIWebView正在加载什么资源，更糟的是，当你希望过滤掉某些资源文件的时候，没有方法可以强制UIWebView不去加载这些文件，

拦截器看起来貌似没有可能实现。

当然还是有解决方案的，否则这篇文件就没什么卵用。

正如上面所说，实现拦截器不能靠UIWebView，因为UIWebView没有提供任何有用的API。

对UIWebView的所有请求，要找到一个能中断所有HTTP 请求的切入点，我们需要先了解一下Cocoa的URL Loading System，因为UIWebView是使用URL Loading System从web端取数据的。我们需要的切入点NSURLCache类就是URL Loading System的一部分。虽然目前iOS系统不会在磁盘上缓存任何数据（后面的iOS系统版本或许会有不同），因此在UIWebView开始加载前，NSURLCache管理的缓存数据通常为空，但UIWebView仍然会检测所请求资源文件是否存在于缓存。所以我们需要做的只是继承NSURLCache并重载其方法：

```
- (NSCachedURLResponse*)cachedResponseForRequest:(NSURLRequest*)request
```

UIWebView请求所有资源时都会调用这个方法。因为我们只需要在这个方法里判断请求的URL是否是我们想拦截的。如果是则创建一个没有内容的假response，否则只需调用super方法即可。

如下是实现细节：

1.继承NSURLCache:

FilteredWebCache.h:

```
@interface FilteredWebCache : NSURLCache
{
}
@end
```

子类的主要代码

FilteredWebCache.m:

```
#import "FilteredWebCache.h"
#import "FilterManager.h"
@implementation FilteredWebCache
```

```
- (NSCachedURLResponse*)cachedResponseForRequest:(NSURLRequest*)request
{
    NSURL *url = [request URL];
    BOOL blockURL = [[FilterMgr sharedFilterMgr] shouldBlockURL:url];
    if (blockURL) {
        NSURLResponse *response =
              [[NSURLResponse alloc] initWithURL:url
                                        MIMEType:@"text/plain"
                           expectedContentLength:1
                                textEncodingName:nil];
        NSCachedURLResponse *cachedResponse =
              [[NSCachedURLResponse alloc] initWithResponse:response
                                data:[NSData dataWithBytes:" " length:1]];
        [super storeCachedResponse:cachedResponse forRequest:request];
        [cachedResponse release];
        [response release];
    }
    return [super cachedResponseForRequest:request];
}
@end
```

首先判断URL是否需拦截（判断通过FilterManager类实现，类实现在此不列出）。如果需要，创建一个无内容的响应对象并把它存在cache中。有人可能会认为只需要返回假的响应对象就够了，没必要缓存它。但这样会因响应对象被系统释放而导致app crash。不知道为何为会这样，可能是iOS的bug（Mac OS X 10.5.x也存在同样问题，而10.4.x及更早的系统上没有问题），也可能是URL Loading System内部类之间的依赖所致。所以我们先缓存响应对象。确保所有响应都是真实存在于缓存中，这也iOS希望的，最重要的是不会crash.

更新：因为假的响应是以大于0的大小来初始化的，看起来结缓存它也是必要的。

2.创建新的缓存：

接下来需要创建一个新的缓存并告诉iOS系统使用新的缓存代替默认的，这样当URL Loading System检测资源缓存时才会调用上面的代码。这要在任意UIWebView开始加载页面前做，显然应该放在app启动的时候：

```
NSString *path = .../// the path to the cache file
NSUInteger discCapacity = 10*1024*1024;
NSUInteger memoryCapacity = 512*1024;
FilteredWebCache *cache =
      [[FilteredWebCache alloc] initWithMemoryCapacity: memoryCapacity
                                diskCapacity: discCapacity diskPath:path];
[NSURLCache setSharedURLCache:cache];
[cache release];
```

这里需要提供一个缓存存储路径。缓存文件由NSURLCache对象自动生成，我们无需事先创建文件，但要定义缓存文件所存位置（必须是应用程序“沙盒”内，如“tmp”目录或是“Document”目录）

这就是实现UIWebView基于URL进行请求过滤的所有内容，看起来其实并不复杂

注：如果过滤规则在app运行过程中会改变，你需要从缓存中删除假的响应。NSURLCache提供了删除方法，所以这不是问题。如果过滤规则不会改变，则无需关心