http协议与web编程

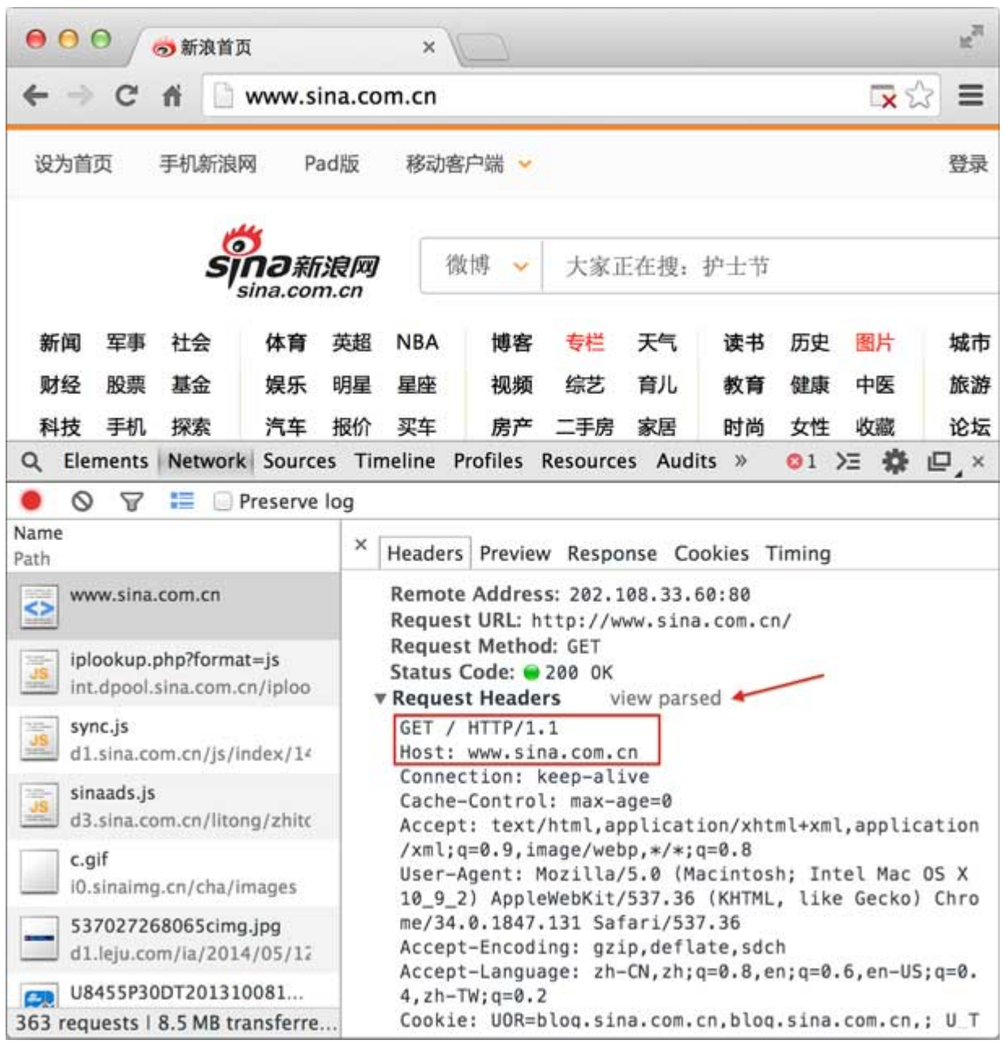
一.http协议介绍

HTTP协议(HyperText Transfer Protocol超文本传输协议)是因特网上应用最为广泛的一种网络传输协议, 所有的WWW文件都必须遵守这个标准.HTTP是一个基于TCP/IP通信协议来传递数据(HTML 文件, 图片文件, 查询结果等)的互联网协议. 在Web应用中, 服务器把网页传给浏览器, 实际上就是把网页的HTML代码发送给浏览器, 让浏览器显示出来; 而浏览器和服务器之间的传输协议是HTTP， 所以HTML是一种用来定义网页的文本, 会HTML就可以编写网页.

1. 使用Google Chrome的开发者工具(F12)来查看服务器和浏览器之间的交互.

Elements显示网页的结构, Network显示浏览器和服务器的通信; 我们点Network确保第一个小红灯亮着,Chrome就会记录所有浏览器和服务器之间的通信.

以请求sina.com.cn为例, 通过代码调试工具看, 请求头与响应头.



最主要的头两行分析如下, 第一行:

1) GET / HTTP/1.1

GET表示一个读取请求, 将从服务器获得网页数据, /表示URL的路径, URL总是以/开头, /就表示首页. 最后的HTTP/1.1指示采用的HTTP协议版本是1.1; 目前HTTP协议的版本就是1.1, 但是大部分服务器也支持1.0版本, 主要区别在于1.1版本允许多个HTTP请求复用一个TCP连接, 以加快传输速度.

2) 从第二行开始, 每一行都类似于Xxx: abcdefg:

Host: www.sina.com.cn (http://www.sina.com.cn)

表示请求的域名是www.sina.com.cn。如果一台服务器有多个网站，服务器就需要通过Host来区分浏览器请求的是哪个网站。(http://www.sina.com.cn%E3%80%82%E5%A6%82%E6%9E%9C%E4%B8%80%E5%8F%B0%E6%9C%8D%E5%8A%A1%E5%99%A8%E6%9C%89%E5%A4%9A%E4%B8%AA%E7%BD%91%E7%AB%99%EF%B

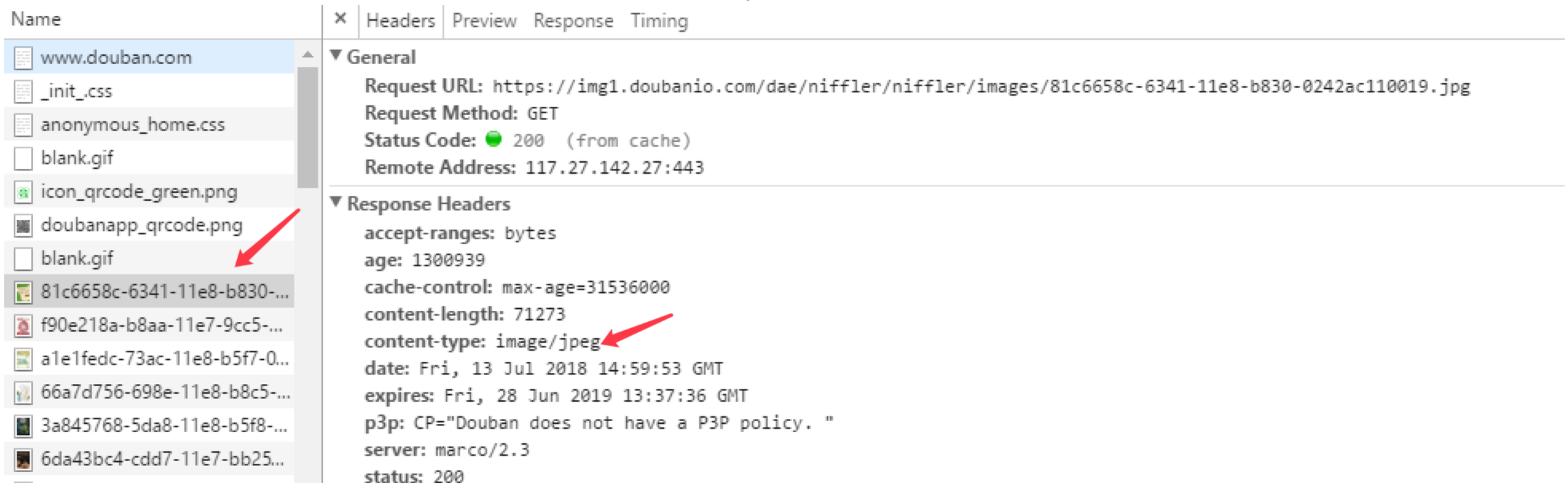
继续往下找到Response Headers，点击view source，显示服务器返回的原始响应数据：

HTTP响应分为Header和Body两部分(Body是可选项), 我们在Network中看到的Header最重要的几行如下:

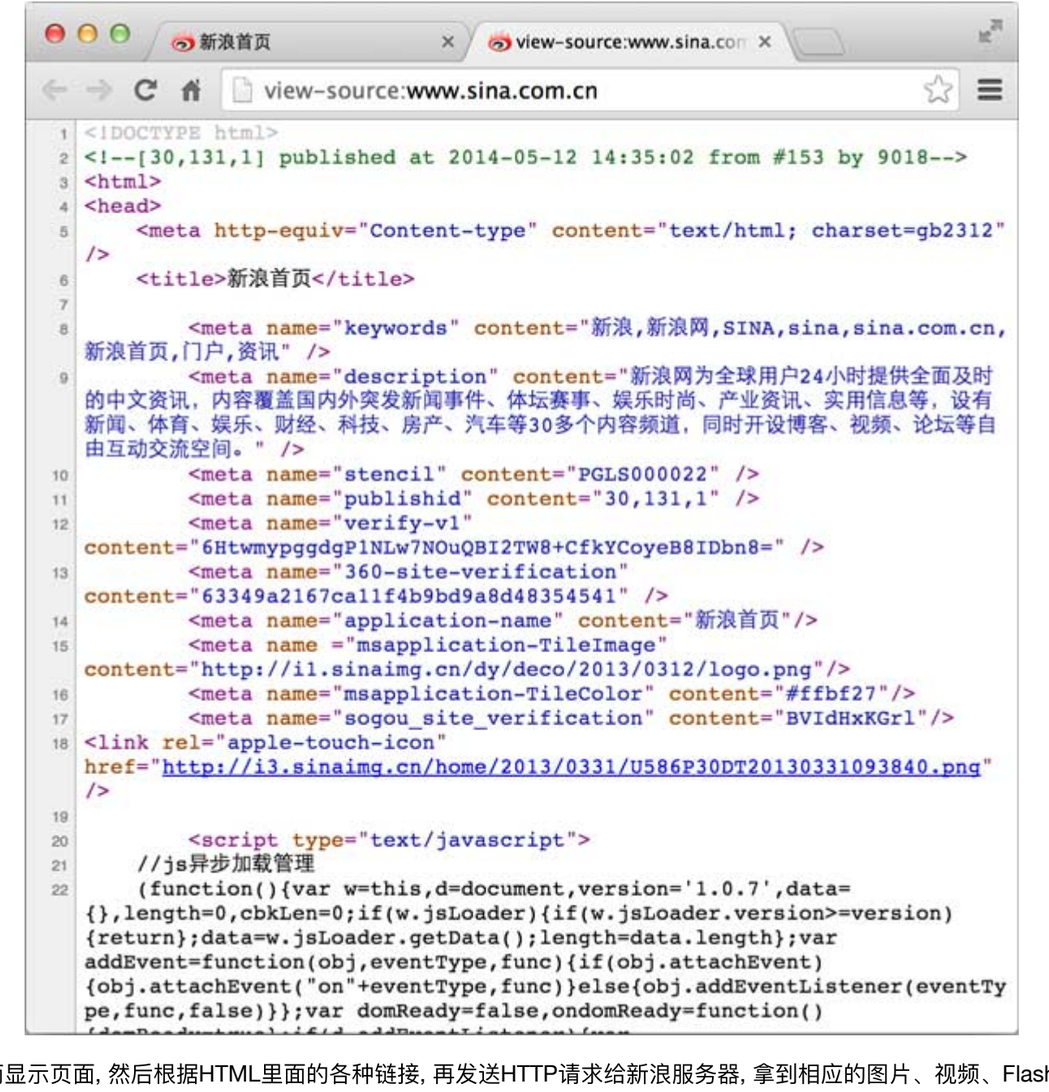
1. 200 OK

   200表示一个成功的响应, 后面的OK是说明; 失败的响应有404 Not Found: 网页不存在; 500 Internal Server Error: 服务器内部出错等等.
2. Content-Type: text/html

   Content-Type指示响应的内容, 这里是text/html表示HTML网页, 请注意浏览器就是依靠Content-Type来判断响应的内容是网页还是图片, 是视频还是音乐.浏览器并不靠URL来判断响应的内容, 所以即使URL是http://example.com/abc.jpg (http://example.com/abc.jpg), 它也不一定就是图片, 我们可以找一个图片看一它的响应头.



HTTP响应的Body就是HTML源码, 我们在菜单栏选择"视图" "开发者", "查看网页源码"就可以在浏览器中直接查看HTML源码:



当浏览器读取到新浪首页的HTML源码后, 它会解析HTML进而显示页面, 然后根据HTML里面的各种链接, 再发送HTTP请求给新浪服务器, 拿到相应的图片、视频、Flash、JavaScript脚本、CSS等各种资源, 最终显示出一个完整的页面. 所以我们在Network下面能看到很多额外的HTTP请求.

二.HTTP请求

跟踪了新浪的首页, 我们来总结一下HTTP请求的流程:

1. 浏览器首先向服务器发送HTTP请求， 请求包括：

   方法: GET还是POST, GET仅请求资源, POST会附带用户数据(如通过表单提交帐号密码).

   路径: /full/url/path

   域名: 由Host头指定Host www.sina.com.cn (http://www.sina.com.cn)

   以及其他相关的Header

   如果是POST, 那么请求还包括一个Body, 包含用户数据.
2. 服务器向浏览器返回HTTP响应, 响应内容包括：

   响应代码: 200表示成功, 3xx表示重定向, 4xx表示客户端发送的请求有错误, 5xx表示服务器端处理时发生了错误.

   响应类型: 由Content-Type指定

   以及其他相关的Header

   通常服务器的HTTP响应会携带内容, 也就是有一个Body, 包含响应的内容, 网页的HTML源码就在Body中.
3. 如果浏览器还需要继续向服务器请求其他资源, 比如图片 就再次发出HTTP请求, 重复步骤1、2.

   Web采用的HTTP协议采用了非常简单的请求-响应模式， 从而大大简化了开发. 当我们编写一个页面时, 我们只需要在HTTP请求中把HTML发送出去, 不需要考虑如何附带图片、视频等. 浏览器如果需要请求图片和视频, 它会发送另一个HTTP请求. 因此一个HTTP请求只处理一个资源, HTTP协议同时具备极强的扩展性, 虽然浏览器请求的是 whttp://www.sina.com.cn/ (http://www.sina.com.cn/) 的首页但是新浪在HTML中可以链入其他服务器的资源，比如 img src=" http://i1.sinaimg.cn/home/2013/1008/U8455P30DT20131008135420.png" (http://i1.sinaimg.cn/home/2013/1008/U8455P30DT20131008135420.png"), 从而将请求压力分散到各个服务器上, 并且一个站点可以链接到其他站点, 无数个站点互相链接起来, 就形成了World Wide Web, 简称WWW.

三.HTTP请求方法

根据HTTP标准, HTTP请求可以使用多种请求方法.

HTTP1.0定义了三种请求方法: GET, POST 和 HEAD方法.

HTTP1.1新增了五种请求方法: OPTIONS, PUT, DELETE, TRACE 和 CONNECT 方法.

序号 方法 描述

1. GET 请求指定的页面信息, 并返回实体主体
2. HEAD 类似于get请求, 只不过返回的响应中没有具体的内容，用于获取报头
3. POST 向指定资源提交数据进行处理请求（例如提交表单或者上传文件）。数据被包含在请求体中。POST请求可能会导致新的资源的建立和/或已有资源的修改。
4. PUT 从客户端向服务器传送的数据取代指定的文档的内容。
5. DELETE 请求服务器删除指定的页面。
6. CONNECT HTTP/1.1协议中预留给能够将连接改为管道方式的代理服务器。
7. OPTIONS 允许客户端查看服务器的性能。
8. TRACE 回显服务器收到的请求，主要用于测试或诊断。

四.HTTP状态码及分类

当浏览者访问一个网页时, 浏览者的浏览器会向网页所在服务器发出请求, 当浏览器接收并显示网页前, 此网页所在的服务器会返回一个包含HTTP状态码的信息头(server header)用以响应浏览器的请求.

HTTP状态码的英文为HTTP Status Code, 下面是常见的HTTP状态码：

1. 200 - 请求成功
2. 301 - 资源（网页等）被永久转移到其它URL
3. 404 - 请求的资源（网页等）不存在
4. 500 - 内部服务器错误

HTTP状态码分类, HTTP状态码由三个十进制数字组成, 第一个十进制数字定义了状态码的类型, HTTP状态码共分为5种类型:

分类 分类描述 1** 服务器收到请求, 需要请求者继续执行操作

2** 操作被成功接收并处理

3** 重定向, 需要进一步的操作以完成请求

4** 客户端错误, 请求包含语法错误或无法完成请求

5** 服务器错误， 服务器在处理请求的过程中发生了错误

例如:

1. 400 Bad Request 语义有误, 当前请求无法被服务器理解, 除非进行修改， 否则客户端不应该重复提交这个请求.
2. 401 认证失败
3. 403 Forbidden 服务器已经理解请求, 但是拒绝执行它.
4. 404 Not Found(请求资源不存在)请求失败, 请求的资源未被在服务器上发现, 没有信息能够告诉用户这个状况到底是暂时的还是永久的; 这个状态码被广泛应用于当服务器不想揭示到底为何请求被拒绝或者没有其他适合的响应可用的情况下, 出现这个错误的最有可能的原因是服务器端已将这个url对应的资源删除.
5. 500 Internal Server Error 服务器遇到了一个未曾预料的状况, 导致了它无法完成对请求的处理. 一般来说, 这个问题都会在服务器端的源代码出现错误时出现。
6. 502 Bad Gateway 作为网关或者代理工作的服务器尝试执行请求时, 从上游服务器接收到无效的响应.
7. 503 Service Unavailable 由于临时的服务器维护或者过载, 服务器当前无法处理请求. 这个状况是临时的, 并且将在一段时间以后恢复. 理500响
8. 504 Gateway Timeout 作为网关或者代理工作的服务器尝试执行请求时, 未能及时从上游服务器(URI标识出的服务器， 例如HTTP、FTP、LDAP)或者辅助服务器(例如DNS)收到响应.
9. 505 HTTP Version Not Supported 服务器不支持或者拒绝支持在请求中使用的HTTP 版本, 这暗示着服务器不能或不愿使用与客户端相同的版本, 响应中应当包含一个描述了为何版本不被支持以及服务器支持哪些协议的实体.

五.Python实现Http请求

通过requests模块请求URL并通过传入headers, 为了让这个请求更是人类用户而不是爬虫, HTTP的请求头是每次向网络服务器发送请求的时候,传递的一组属性和配置信息HTTP定义了十几种请求头信息, 不过大多数都不常用, 下面的为常用的请求头部信息.

所以通过请求头信息就可以区分哪些是真实的用户哪些是爬虫

请求头可以通过 urllib2 或 request 模块自己定义, 虽然网站可能会对HTTP请求头的每个属性进行检查, 但是最重要的可能被检查的参数信息是 User-Agent 所以无论做什么带有http请求的项目, 建议都要修改User-Agent属性, 将其设置成不容易引起怀疑的内容, 如果是一些警觉性比较高的网站, 可能还会检查比如 Accept-Lanuage 这些属性, 请将这些属性妥善配置.

In [4]:
```
#示例代码1.
import requests
url = 'http://www.whatismybrowser.com'
headers = {'User-Agent': 'Mozilla/5.0 (Windows NT 10.0; WOW64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/52.0.2743.116 Safari/537.36'}
response = requests.get(url, headers=headers)
print response.status_code, response.text

#通过以上代码， 我们成功修改了请求头部信息
```

六.什么是restApi?

REST全称是资源的表述性状态转移, 那究竟指的是什么的表述? 其实指的就是资源, 任何事物只要有被引用到的必要, 它就是一个资源; 下面是一些资源的例子: 某用户的手机号码 某用户的个人信息等.

要让一个资源可以被识别, 需要有个唯一标识, 在Web中这个唯一标识就是URI(Uniform Resource Identifier). URI既可以看成是资源的地址, 也可以看成是资源的名称, RESTful API由后台SERVER来提供给前端来调用, 前端调用API向后台发起HTTP请求, 后台响应请求将处理结果反馈给前端, 也就是说RESTful是典型的基于HTTP的协议. 那么RESTful API有哪些设计原则和规范呢?

1. 资源: 资源就是网络上的一个实体, 一段文本, 一张图片或者一首歌曲.
2. 统一接口: RESTful风格的数据操作(create,read,update,delete) 分别对应HTTP方法: GET用来获取资源, POST用来新建资源, PUT用来更新资源, DELETE用来删除资源, 这样就统一了数据操作的接口.
3. URI: 可以用一个URI(统一资源定位符)指向资源, 即每个URI都对应一个特定的资源, 要获取这个资源访问它的URI就可以, 因此URI就成了每一个资源的地址或识别符, 一般的每个资源至少有一个URI与之对应, 最典型的URI就是URL.
4. 无状态: 所谓无状态即所有的资源都可以URI定位, 而且这个定位与其他资源无关, 也不会因为其他资源的变化而变化.

   有状态和无状态的区别, 举个例子说明一下, 例如要查询员工工资的步骤为

   第一步:登录系统

   第二步: 进入查询工资的页面

   第三步: 搜索该员工

   第四步: 点击姓名查看工资

   这样的操作流程就是有状态的, 查询工资的每一个步骤都依赖于前一个步骤, 只要前置操作不成功, 后续操作就无法执行. 如果输入一个URL就可以得到指定员工的工资, 则这种情况就是无状态的, 因为获取工资不依赖于其他资源或状态, 且这种情况下, 员工工资是一个资源, 由一个URL与之对应, 可以通过HTTP中的GET方法得到资源, 这就是典型的RESTful风格.

In [5]:
```
#示例代码: Python请求kubernetes-api, 获取namespace

class k8sApi:
    def __init__(self, ipAddr, Port=8080):
        self.ipAddr = ipAddr
        self.Port = Port
        self.headers = {'Content-Type': 'application/json'}

    def listNamespaces(self, name=None):
        if name:
            url = 'http://%s:%s/api/v1/namespaces/%s'%(self.ipAddr, self.Port, name)
        else:
            url = 'http://%s:%s/api/v1/namespaces' %(self.ipAddr, self.Port)
        print url
        try:
            resp = requests.get(url)
        except:
            raise RequestError("Call K8s API %s Error!!" % url)
        else:
            return resp.status_code, resp.json()

if __name__ == '__main__':
    x = k8sApi('172.16.70.231')
    print x.listNamespaces()
```

In [ ]:
```
#示例代码: Python请求harbor API
import requests
from requests.auth import HTTPBasicAuth

headers = {"Content-Type": "application/json"}
url = "http://172.16.70.203/api/projects"
x = requests.get(url, headers=headers,  auth=HTTPBasicAuth("admin", "Harbor12345"))
print x.status_code
print x.text
```