# 一. 什么是爬虫?

**请求**网站并**提取**数据的自动化程序。

# 二. 爬虫的基本流程

- 发起请求

  > 通过HTTP库目标站点发起请求，即发送一个Request，请求可以包含额外的headers等信息，等待服务器响应。

- 获取响应内容

  > 如果服务器能够正常响应，会得到一个Response，Response的内容便是所要获取的页面内容，类型可能有HTML，Json字符串，二进制数据（如图片视频）等类型。

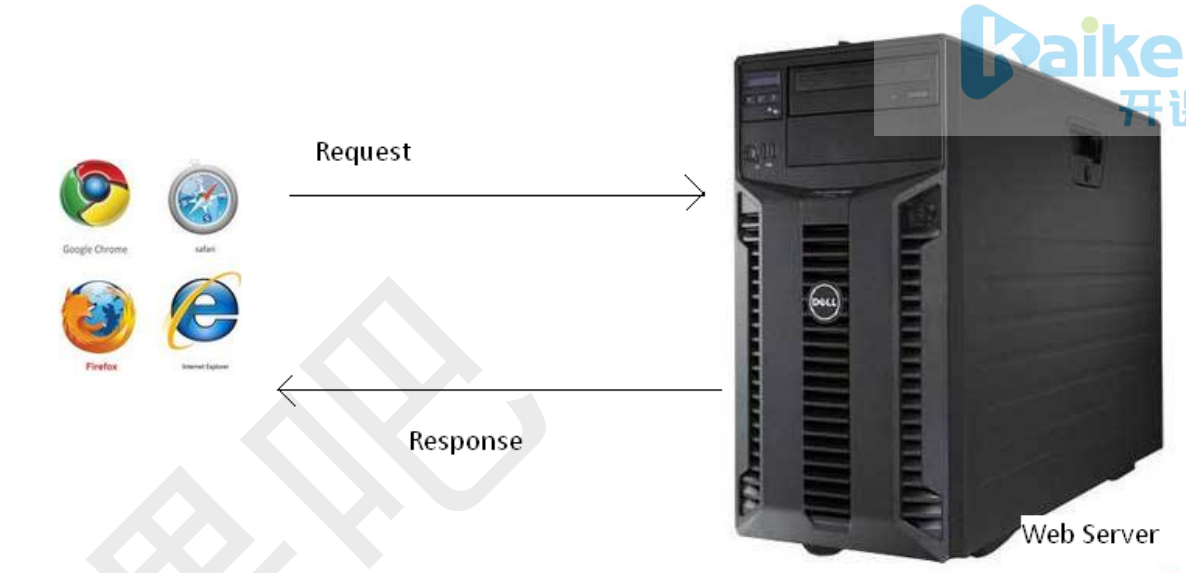
- 解析内容

  > 得到的内容可能是HTML，可以用正则表达式，网页解析库进行解析。可能是Json，可以直接转为Json对象解析，可能是二进制数据，可以做保存或者进一步的处理。

- 保存数据

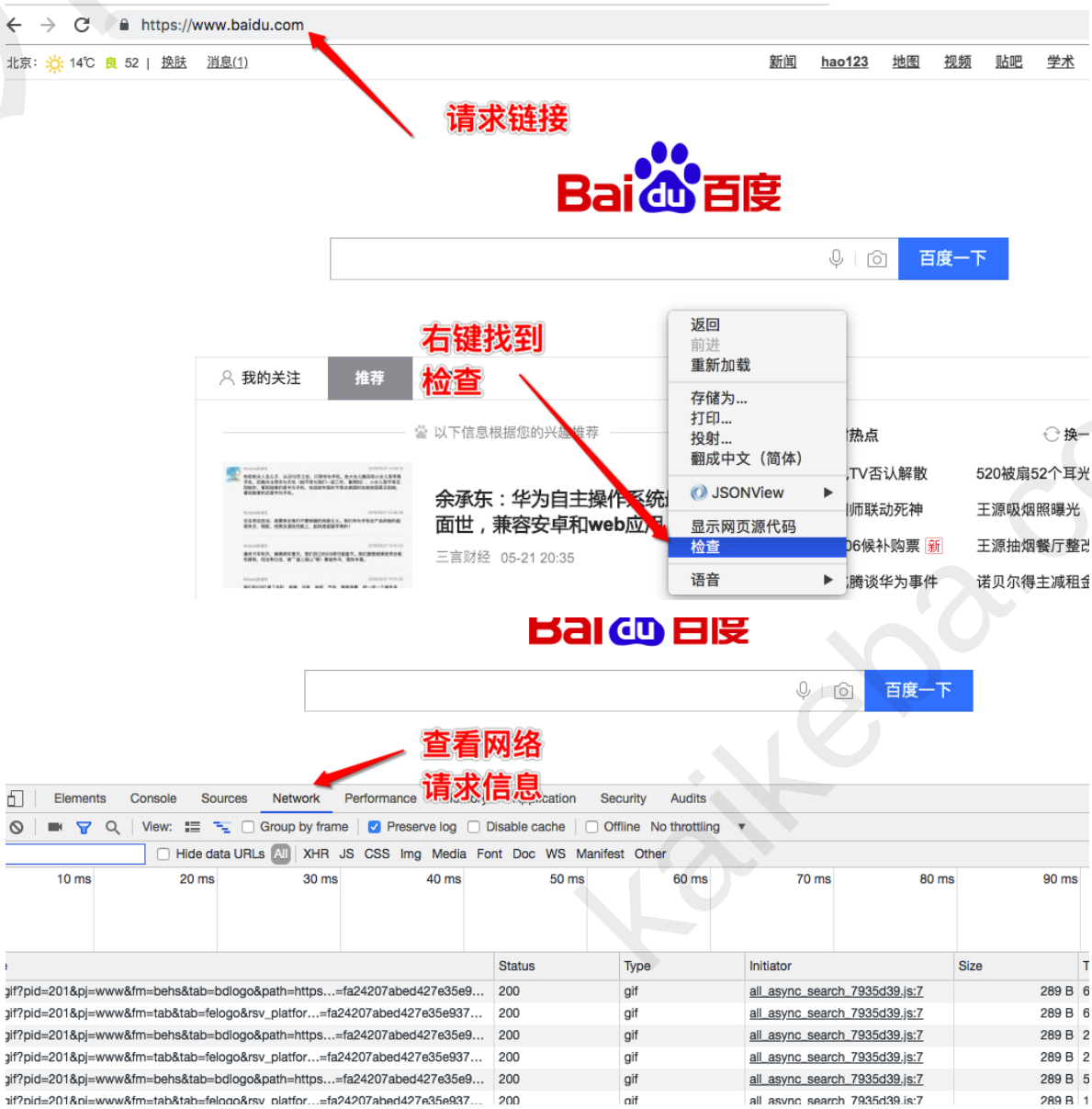
  > 保存形式多样，可以存为文本，也可以保存至数据库，或者保存特定格式的文件。

# 三. Request和Response




- 浏览器发送消息给给该网址所在的服务器，这个过程叫做HTTP Request。
- 服务器收到浏览器发送的消息后，能够根据浏览器发送消息的内容，做响应处理，然后把消息回传给浏览器。这个过程叫做HTTP Response。
- 浏览器收到服务器的Response信息后，会对信息进行响应处理，然后展示。

查看浏览器的请求和响应：



# 1. Request

**Request**是请求,在浏览器输入地址,回车,就是一个请求。

- 请求方式

  > 主要有get、post两种类型，另外还有HEAD、PUT、DELETE、OPTIONS等。

- 请求的URL

  > URL 全称统一资源定位符，如一个网页的文档、一张图片、一个视频等都可以用URL来确定。

- 请求头

  > 包含请求时的头部信息，如User-Agent、host、Cookies等信息

- 请求体

  > 请求时额外携带的数据如表单提交时的表单数据。

# 2. Response

**Response**是响应,服务器根据请求,返回数据到浏览器显示,就是一个响应。

- 响应状态

  > 有多种响应状态，如200代表成功、301跳转、404找不到页面、502服务器错误。

- 响应头

  > 如内容类型、内容长度、服务器信息、设置Cookie等等。

- 响应体

  > 最主要的部分，包含了请求资源的内容，如网页HTML、图片二进制数据等。

例如:

```
# 导入网络请求模块（该模块需要使用pip install requests 安装）
import requests

# 创建请求头
headers = {
    "User-Agent":'Mozilla/5.0 (Macintosh; Intel Mac OS X 10_13_6)
AppleWebKit/537.36 (KHTML, like Gecko) Chrome/74.0.3729.131 Safari/537.36'
}
# 发送网络请求
response = requests.get(url='https://www.baidu.com',headers = headers)

# 获取请求的内容
print(response.content.decode('utf-8'))
print(response.text)

# 获取响应头
print(response.headers)

# 状态码
print(response.status_code)
```

# 四. 能抓取什么样的数据?

- 网页文本

  > 如HTML文档、Json格式的文本等。

- 图片

  > 获取到的是二进制文件，保存为图片格式。

- 视频

  > 获取到的是二进制文件，保存为视频格式。

例如： 获取图片

```
import requests

# 创建请求头
headers = {
    "User-Agent":'Mozilla/5.0 (Macintosh; Intel Mac OS X 10_13_6)
AppleWebKit/537.36 (KHTML, like Gecko) Chrome/74.0.3729.131 Safari/537.36'
}
# 发送网络请求
response = requests.get(url='https://www.baidu.com/img/bd_logo1.png',headers =
headers)

# 将获取的二进制内容进行保存
with open('./baidu.png','wb') as f:
    f.write(response.content)
```

# 五. 网页的解析方式

- 直接处理

  > 例如： 请求回来的就是一段字符串，我们可以简单处理后直接保存

- Json解析

  > 例如： Ajax请求回来的一般都是json格式的数据，我们就需要从json中拿出我们想要的数据。

- 正则表达式
- BeautifulSoup
- PyQuery
- XPath

解析库的选择可以根据爬取的网页具体情况酌情选择。

# 六. 前端初体验(了解)

## 1. HTML

**超级文本标记语言（HyperText Markup Language）**是标准通用标记语言下的一个应用，也是一种规范和标准，它通过标记符号来标记要显示的网页中的各个部分。网页文件本身是一种文本文件，通过在文本文件中添加标记符，可以告诉浏览器如何显示其中的内容（如：文字如何处理，画面如何安排，图片如何显示等）。

## 1.1 HTML元素

**1.根元素**

| | |
|---|---|
| `<doctype>` | **定义文档类型。** |
| `<html>` | 定义 HTML 文档。 |

**2.元数据元素**

| | |
|---|---|
| `<head>` | **定义关于文档的信息。** |
| `<meta>` | 定义关于 HTML 文档的元数据。 |
| `<link>` | 定义文档与外部资源之间的关系，一般用于引入样式表。 |
| `<base>` | 定义页面上所有链接的默认地址或默认目标。 |
| `<title>` | 定义文档标题。 |
| `<style>` | 定义文档的样式信息。 |

**3.脚本元素**

| | |
|---|---|
| `<script>` | **定义客户端脚本。** |
| `<noscript>` | 定义当浏览器不支持脚本的时候所显示的内容 |

**4.块元素**

| | |
|---|---|
| `<body>` | **定义文档的主体。** |
| `<h1>、<h2>...<h6>` | 定义文档标题。 |
| `<p>` | 定义文档段落。 |
| `<blockquote>` | 定义块引用。 |
| `<ul>、<ul>、<dl>` | 定义列表。 |
| `<table>` | 定义表格。 |

**5.列表元素**

无序列表

| | |
|---|---|
| `<ul>` | **定义无序的列表。** |
| `<li>` | 定义列表项。 |

有序列表

| | |
|---|---|
| `<ol>` | **定义有序的列表。** |
| `<li>` | 定义列表项。 |

定义列表

| | |
|---|---|
| `<dl>` | **定义定义列表。** |
| `<dt>` | 定义定义术语。 |
| `<dd>` | 定义定义描述。 |

**6.表格元素**

| | |
|---|---|
| `<table>` | **定义表格。** |
| `<thead>` | 定义表格的页眉。 |
| `<tbody>` | 定义表格的主体。 |
| `<tfoot>` | 定义表格的页脚。 |
| `<th>` | 定义表格的表头行。 |
| `<tr>` | 定义表格的行。 |
| `<td>` | 定义表格单元。 |

**7.文本元素**

文本格式化元素

| | |
|---|---|
| `<em>` | **定义着重文字。** |
| `<strong>` | 定义加重语气。 |
| `<sup>` | 定义上标字。 |
| `<sub>` | 定义下标字。 |
| `<ins>` | 定义插入字。 |
| `<del>` | 定义删除字。 |
| `<b>` | 定义粗体文本。 |
| `<i>` | 定义斜体文本。 |
| `<big>` | 定义大号字。 |
| `<small>` | 定义小号字。 |

**8.链接与图像**

| | |
|---|---|
| `<a>` | **定义超链接** |
| `<img>` | 定义图像。 |
| `<map>` | 定义图像地图。 |
| `<area>` | 定义图像地图中的可点击区域。 |

**9.`<div>` 和 `<span>`**

| | |
|---|---|
| `<div>` | **定义文档中的分区或节（division/section）。** |
| `<span>` | 定义 span，用来组合文档中的行内元素。 |

**10.表单元素**

| | |
|---|---|
| `<form>` | **定义供用户输入的表单。** |
| `<input>` | 定义输入域。 |
| `<textarea>` | 定义文本域 (一个多行的输入控件)。 |
| `<lable>` | 定义一个控制的标签。 |
| `<select>` | 定义一个选择列表。 |
| `<option>` | 定义下拉列表中的选项。 |
| `<optgroup>` | 定义选项组。 |
| `<button>` | 定义一个按钮。 |
| `<fieldset>` | 定义域。 |
| `<legend>` | 定义域的标题。 |

# 2. CSS

## 2.1 CSS简介

CSS是一种定义样式结构如字体、颜色、位置等的语言，用于描述网页上的信息格式化和现实的方式。CSS样式可以直接存储于HTML网页或者单独的样式单文件。

## 2.2 css类型

内联方式

样式定义在单个的 HTML元素中

内部样式表

样式定义在 HTML 页的头元素中

外部样式表

将样式定义在一个外部的 CSS 文件中（.css 文件）


# 3. JavaScript

JavaScript是一种属于网络的脚本语言,已经被广泛用于Web应用开发,常用来为网页添加各式各样的动态功能,为用户提供更流畅美观的浏览效果。通常JavaScript脚本是通过嵌入在HTML中来实现自身的功能的。

## JavaScript语法

1. js代码能直接嵌入网页的任何地方，通常放在`<head></head>`中，但会要求用户把所有js代码都下载解析执行以后在呈现内容；
2. 可以把代码单独的放在一个.js文件中，同时，多个页面可以引用同一份.js文件；
3. 页面中可以多次编写`<script></script>`,但是浏览器会按照顺序执行这些JavaScript代码；
4. JavaScript中每个语句以； 结尾 每个语句块以{.....} 注释用//或/* */
5. 所有类型都是定义变量都是用 var， 如果一个变量没有通过var申明就被使用，那么该变量就自动被申明为全局变量；

hello.html

```
<html>
    <head>
        <title>js</title>
        <script type="text/javascript">
            console.log("Hello javascript")
        </script>
    </head>
    <body>
    </body>
</html>
```

外部样式表

test.js

```
alert('Hello JS')
```

test.html

```
<html>

    <head>
        <title>js</title>
        <script type="text/javascript" src='test.js'></script>
    </head>
    <body>
    </body>
</html>
```

# 七. Xpath的基本使用

## 7.1 简介

**XPath** 是一门在 XML 文档中查找信息的语言。 使用路径表达式来选取 XML 文档中的节点或者节点集。

**注意**： xpath速度比较快，是爬虫在网页定位中的较优选择，但是很多网页前端代码混乱难以定位。

安装：`pip install lxml`

## 7.2 Xpath语法

xpath的更多语法

```
'''
表达式        描述
nodename     选取此节点的所有子节点。
/            从根节点选取。
//           从匹配选择的当前节点选择文档中的节点，而不考虑它们的位置。
.            选取当前节点。
..           选取当前节点的父节点。
@            选取属性。
'''

'''
通配符       描述
*            匹配任何元素节点。
@*           匹配任何属性节点。
node()       匹配任何类型的节点。
'''
```

## 7.3 基础使用

```
# 导入lxml的etree库
from lxml import etree

data_str = """
        <div>
            <ul>
                 <li class="item-0"><a href="link1.html">first item</a></li>
                 <li class="item-1"><a href="link2.html">second item</a></li>
```

```
                <li class="item-inactive"><a href="link3.html">third item</a>
</li>
                <li class="item-1"><a href="link4.html">fourth item</a></li>
                <li class="item-0"><a href="link5.html">fifth item</a>
            </ul>
         </div>
        """
# 注意： 该数据中缺少了一个li标签的闭合标签

# 利用etree.HTML可以将字符串或者bytes转化为Element python对象，这个对象具有Xpath的方法
html = etree.HTML(data_str)
# print(html)

# etree.tostring(html)可以自动修正HTML代码，补全了缺胳膊少腿的标签
# 使用为了观察修改以后的html样子，根据修改后的HTML去写Xpath
result = etree.tostring(html)
print(result.decode("utf-8"))

# 获取class =item-1 的 a标签的href属性
result = html.xpath('//li[@class="item-1"]/a/@href')
print(result)
```

## 7.4 Xpath练习

```
import requests
from lxml import etree
tiebaName = 'lol'
base_url = 'https://tieba.baidu.com/f?kw='+tiebaName+'&ie=utf-8&pn={}'
headers = {
    'User-Agent': 'Mozilla/5.0 (Macintosh; Intel Mac OS X 10_13_6)
AppleWebKit/537.36 (KHTML, like Gecko) Chrome/74.0.3729.131 Safari/537.36'
}

# 构建请求连接
url_list = [] # 存储连接的列表
for i in range(1): # 初始为第一页
    url_list.append(base_url.format(i*50))

# 发送网络请求并得到相应内容
response = requests.get(url=url_list[0],headers = headers)

result_str = response.content.decode('utf-8')
# 由于爬取的内容被注释了，所以将注释去掉
result_str = result_str.replace('<!--','').replace('-->','')

html = etree.HTML(result_str)

# 获取每一页的连接
links = html.xpath('//li[@class=" j_thread_list
clearfix"]//div[@class="threadlist_title pull_left j_th_tit
"]//a[@rel="noreferrer"]/@href')
# 拼接成完整的连接
links = ['https://tieba.baidu.com/{}'.format(i) for i in links]

# 获取文字
```

```
texts = html.xpath('//li[@class=" j_thread_list
clearfix"]//div[@class="threadlist_title pull_left j_th_tit
"]//a[@rel="noreferrer"]/text()')
```