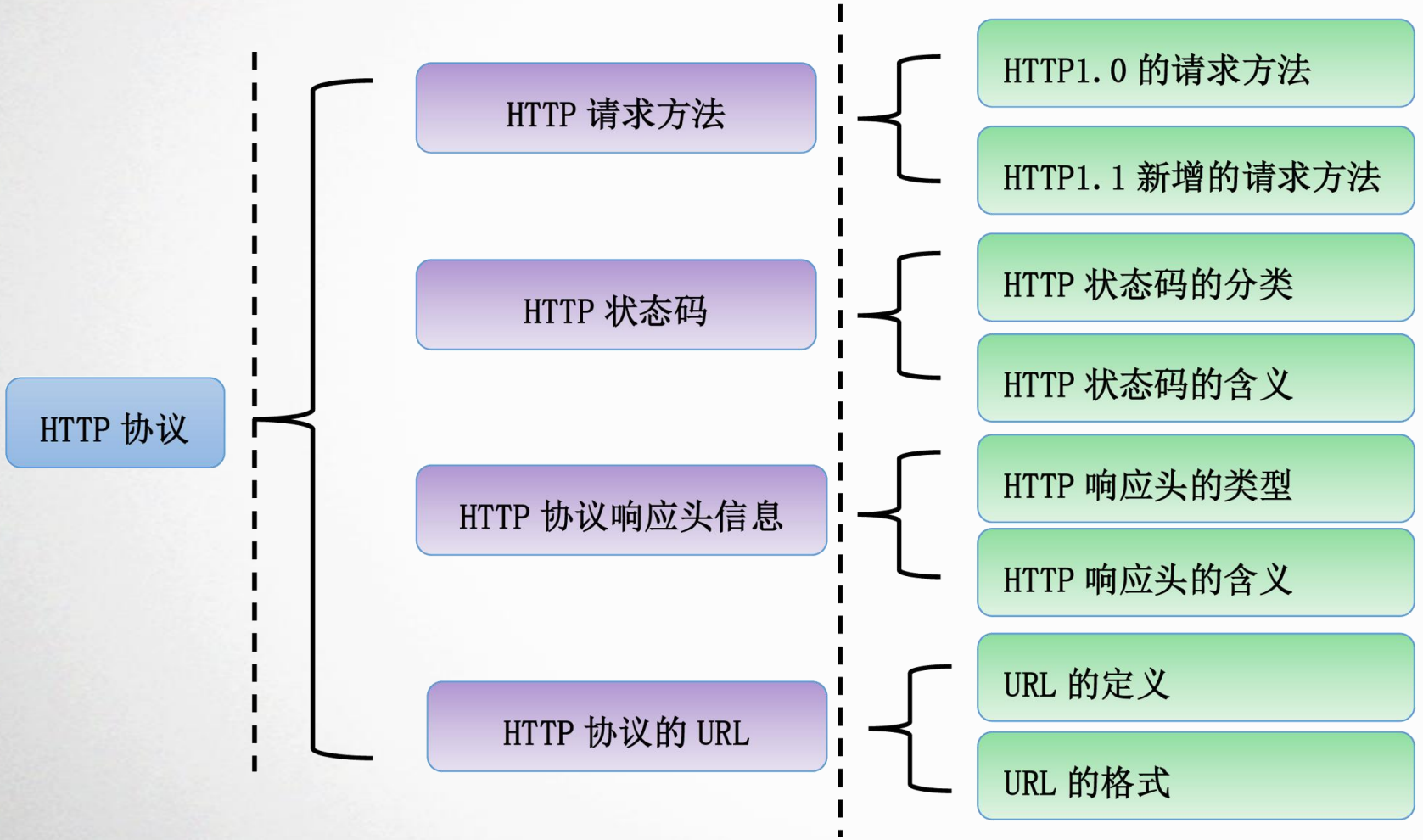
培训教育
Training Services

CISP-PTE

Web 安全基础(1) – HTTP协议

主讲：

HTTP协议



知识体　　知识域　　知识子域

GET POST HEAD

HTTP请求方法

HTTP请求方法

- 通过本知识域，我们会：
  - HTTP1.0的请求方法
    - 了解HTTP1.0三种请求方法，GET, POST 和 HEAD
    - 掌握GET请求的标准格式
    - 掌握POST请求提交表单，上传文件的方法
    - 了解HEAD请求与GET请求的区别
  - HTTP1.1新增的请求方法
    - 了解HTTP1.1新增的五种请求方法：OPTIONS，PUT，DELETE，TRACE，CONNECT方法的基本概念
    - 掌握HTTP1.1新增的五种请求的基本方法和产生的请求结果

HTTP协议 - 请求

- http请求由三部分组成，分别是：请求行、消息报头、请求正文
- 请求行以一个方法符号开头，以空格分开，后面跟着请求的URI和协议的版本，格式如下：
  - Method Request-URI HTTP-Version (CR)(LF)
  - 其中 Method表示请求方法(GET,POST,HEAD等)
  - Request-URI是一个统一资源标识符
  - HTTP-Version表示请求的HTTP协议版本
  - (CR)(LF)表示回车和换行（除了作为结尾的(CR)(LF)外，不允许出现单独的(CR)或(LF)字符）。
  - 例：GET /get.php?arg1=value1 HTTP/1.1

HTTP协议 – 请求（续）

- HTTP消息请求报头允许客户端向服务器端传递请求的附加信息以及客户端自身的信息。
- 常用的报头：
- 常用的请求报头
  Accept，Accept-Charset，Accept-Encoding，Accept-Language，Authorization，Host（发送请求时，该报头域是必需的），User-Agent
- 每个类型请求头结束后，会跟上(CR)(LF)
- 消息报头和请求正文会隔一行

HTTP请求 - GET

- GET请求格式:
  - <访问路径>[?<arg1>=<value1>[&<arg2>=<value2>....]]

    例子：http://site1.com/get.php?arg1=value1
- Server端可以根据参数名获取值:
  - PHP例子:

```
<?php
echo $_GET['arg1'];
?>
```

结果就是输出arg1相应的值

```
$ curl "http://192.168.0.105/get.php?arg1=value1"
value1
```

HTTP请求 - GET

- GET请求，是可以把数据放在URL中来传递，也可以不包含任何数据，HTTP请求只有请求头，没有请求数据。
- 请求头中可以不包含Content-Length，例子：

```
GET /get.php?arg1=value1 HTTP/1.1
Host: site1.com
Connection: close
User-Agent: Paw/2.2.5 (Macintosh; OS X/10.12.2) GCDHTTPRequest
```

- GET请求可以只有请求的路径：
  - http://www.gooann.com
- GET请求也可以带需要传递的数据，在访问路径之后带问号（?）+参数=值的方式发送。
  - http://site1.com/get.php?arg1=value1

HTTP请求 - GET

- 在服务端使用相应的方法（函数）来获取相应的值，比如PHP可以用如下方法：

```
<?php
echo $_GET['arg1'];
?>
```

- 结果是输出arg1相应的值。

HTTP请求 – POST

- POST请求是包含数据
- 请求数据的格式，可以在HTTP头中定义。格式一般会有：
  - 表单格式：application/x-www-form-urlencoded
  - 混合格式：multipart/form-data
  - JSON格式：application/json
  - XML格式：text/xml
  - 文本：text/plain
- 表单格式：
  - 与get方式类似，是把所有提交数据放在数据区域。

HTTP请求 – POST（续）

- POST方式也可以像GET方式在URL带参数，但一般不会这么去使用。
- 表单方式与GET方式类似，只是把数据放在头文件下面的请求正文区域。请求如下：

```
POST /post-form.php HTTP/1.1
Content-Type: application/x-www-form-urlencoded
Host: 192.168.0.105
Connection: close
User-Agent: Paw/2.2.5 (Macintosh; OS X/10.12.2) GCDHTTPRequest
Content-Length: 11

arg1=value1
```

HTTP请求 – POST（续）

- 混合模式
  - 有文件上传时常用的方法。可以接受同时提交不同类型的数据
    - 表单中，可以把类型更改为file就可以上传文件

    <input type="file" name="file" id="file" />
  - 类型后面一般会跟boundary来告知数据区域分隔符
  - 每个数据都可以单独说明数据类型
  - 获取文件时，可以使用相应参数：
    - PHP代码如下：

    $_FILES["file"]["name"] - 文件名
    $_FILES["file"]["type"] - 类型
    $_FILES["file"]["size"] - 文件大小
    $_FILES["file"]["tmp_name"] - 临时文件路径

HTTP请求 – POST（续）

- 混合模式一般是用来传输文件。后面会跟boundary=__xxxx__来进行每个参数的分割。

```
POST /upload_file.php HTTP/1.1
Content-Type: multipart/form-data; boundary=------WebKitFormBoundaryRTP6hG23yFYrExfg
Host: 192.168.0.105
Connection: close
User-Agent: Paw/2.2.5 (Macintosh; OS X/10.12.2) GCDHTTPRequest
Content-Length: 101

------WebKitFormBoundaryRTP6hG23yFYrExfg
Content-Disposition: form-data; name="arg1"

value1
------WebKitFormBoundaryRTP6hG23yFYrExfg
Content-Disposition: form-data; name="file"; filename="python.txt"
Content-Type: text/plain

多线程
xxxxx
------WebKitFormBoundaryRTP6hG23yFYrExfg
Content-Disposition: form-data; name="file2"; filename="mails.zip"
Content-Type: application/zip

xxxxx
------WebKitFormBoundaryRTP6hG23yFYrExfg--
```

HTTP请求 – POST（续）

- 文本模式
  - 常见类型为json,xml,plain
  - 这种类型的数据，需要服务端代码自行解析
    - PHP代码为例：

    file_get_contents("php://input"); - 可以获取数据区域文本
    要是接受的是json，使用相关方法（函数）来解析
    echo json_decode(file_get_contents("php://input"),true)['arg1'];
    可以根据type（类型）来解开数据。

HTTP请求 – POST（续）

- 文本模式，也可以按照文件来接收，
  - 使用file_get_contents（“php://input”）; 可以免去读取文件里的内容。
  - file_get_contents（“php://input”）; 模式不能接收multipart/form-data模式。
- 下面代码就是PHP写的，当类型为json时，根据json解析

```
<?php
if( $_SERVER['CONTENT_TYPE'] == 'application/json')
{
   echo json_decode(file_get_contents("php://input"),true)['arg1'];
}
?>
```

  - 这时请求数据是：{“arg1”：“value1”}
  - 返回值是：value1

HTTP请求 – HEAD

- HEAD请求就是返回只有头部数据，数据部分不返回内容
  - 返回的内容基本上与GET，POST的返回头一致

GET模式的返回头：

```
HTTP/1.1 200 OK
Cache-Control: private
Content-Type: text/html; charset=utf-8
Server: Microsoft-IIS/7.5
X-AspNet-Version: 2.0.50727
X-Powered-By: ASP.NET
X-UA-Compatible: IE=EmulateIE7
Date: Sun, 20 Aug 2017 07:11:17 GMT
Connection: close
Content-Length: 45511
```

HEAD模式的返回头：

```
HTTP/1.1 200 OK
Cache-Control: private
Content-Length: 45511
Content-Type: text/html; charset=utf-8
Server: Microsoft-IIS/7.5
X-AspNet-Version: 2.0.50727
X-Powered-By: ASP.NET
X-UA-Compatible: IE=EmulateIE7
Date: Sun, 20 Aug 2017 07:11:51 GMT
Connection: close
```

HTTP请求 – OPTIONS

- OPTIONS请求，默认情况下会返回允许的请求类型
  - http://www.microsoft.com/zh-cn/ 会返回：
    - Access-Control-Allow-Headers: Origin, X-Requested-With, Content-Type, Accept
    - Access-Control-Allow-Methods: GET, POST, PUT, DELETE, OPTIONS
    - Access-Control-Allow-Credentials: true
  - 一般需要跨域的时候需要设置OPTIONS（跨域后面来讲）
  - 使用脚本让浏览器跨域进行请求的时候，会检测OPTIONS，对方服务器是否允许跨域。允许的情况下，才会真正去进行相应请求
    - 当你用浏览器访问 site1.com, 某些脚本操作会提交到site2.com而且附带非浏览器默认的头信息，这个时候浏览器不会直接发出POST请求，先发出OPTIONS请求来判断是否允许发送POST。当对方回复允许了，才能发送POST请求。

HTTP请求 – PUT DELETE

- PUT：在特定目录里上传指定文件，文件名在url中设置。
- DELETE：删除特定目录里的文件，文件名在url中设置。
- 在NGINX中，可以添加如下配置来允许PUT，DELETE

```
location /upload/ {
        dav_methods  PUT DELETE;
        root   /usr/share/nginx/html;
}
```

PUT / DELETE使用方法

- PUT - 发送包

```
PUT /upload/Untitled HTTP/1.1
Content-Type: application/octet-stream
Host: 192.168.1.64
Connection: close
User-Agent: Paw/2.2.5 (Macintosh; OS X/10.12.6) GCDHTTPRequest
Content-Length: 284

<data>....
```

PUT/DELETE使用方法

- PUT – 返回包

  HTTP/1.1 201 Created
  Server: nginx/1.12.1
  Date: Fri, 01 Sep 2017 13:40:03 GMT
  Content-Length: 0
  Location: http://192.168.1.64/upload/Untitled
  Connection: close

DELETE使用方法

- DETELE – 发送包

  DELETE /upload/Untitled HTTP/1.1

  Host: 192.168.1.64

  Connection: close

  User-Agent: Paw/2.2.5 (Macintosh; OS X/10.12.6) GCDHTTPRequest

DELETE使用方法

- DELETE – 返回包

  HTTP/1.1 204 No Content

  Server: nginx/1.12.1

  Date: Fri, 01 Sep 2017 13:51:07 GMT

  Connection: close

HTTP请求 – TRACE ,CONNECT

- HTTP TRACE是让我们的web服务器端将客户端的所有请求信息返回给客户端的方法，该方法多见于debug的需求。
- CONNECT是在特定应用走HTTP协议时会用到
  - proxy可能会用到
  - 某些使用http协议，需要长链接的程序（SSL就使用connect）

HTTP状态码

HTTP状态码

- 通过本知识域，我们会：
  - HTTP状态码的分类
    - 了解HTTP状态码的规范？
    - 了解HTTP状态码的作用？
    - 掌握常见的HTTP状态码？
  - HTTP状态码的含义
    - 了解HTTP状态码2**,3**,4**,5** 代表的含义？
    - 掌握用计算机语言获取HTTP状态码的方法？

HTTP状态码的分类

- HTTP状态码的作用是：Web服务器用来告诉客户端，发生了什么事。
- HTTP状态码被分为五大类， 目前我们使用的HTTP协议版本是1.1，支持以下的状态码。

	已定义范围	分类
1XX	100-101	信息提示
2XX	200-206	成功
3XX	300-307	重定向
4XX	400-417	客户端错误
5XX	500-505	服务端错误

状态码的分类（续）

- **常见的状态码**
  - 200 OK 服务器成功处理了请求（这个是我们见到最多的）
  - 301/302 Moved Permanently（重定向）请求的URL已移走。Response中应该包含一个Location URL, 说明资源现在所处的位置
  - 304 Not Modified（未修改）客户的缓存资源是最新的， 要客户端使用缓存
  - 404 Not Found 未找到资源
  - 501 Internal Server Error服务器遇到一个错误，使其无法对请求提供服务

状态码的含义 - 1XX

状态码	状态消息	含义
100	Continue(继续)	收到了请求的起始部分，客户端应该继续请求
101	Switching Protocols（切换协议）	服务器正根据客户端的指示将协议切换成Update Header列出的协议

状态码的含义 - 2XX

状态码	状态消息	含义
200	OK	服务器成功处理了请求（这个是我们见到最多的）
201	Created（已创建）	对于那些要服务器创建对象的请求来说，资源已创建完毕。
202	Accepted（已接受）	请求已接受， 但服务器尚未处理
203	Non-Authoritative Information（非权威信息）	服务器已将事务成功处理，只是实体Header包含的信息不是来自原始服务器，而是来自资源的副本。
204	No Content(没有内容)	Response中包含一些Header和一个状态行， 但不包括实体的主题内容（没有response body）
205	Reset Content(重置内容)	另一个主要用于浏览器的代码。意思是浏览器应该重置当前页面上所有的HTML表单。
206	Partial Content（部分内容）	部分请求成功

状态码的含义 - 3XX

状态码	状态消息	含义
300	Multiple Choices（多项选择）	客户端请求了实际指向多个资源的URL。这个代码是和一个选项列表一起返回的，然后用户就可以选择他希望的选项了
301	Moved Permanently（永久移除)	请求的URL已移走。Response中应该包含一个Location URL，说明资源现在所处的位置
302	Found（已找到）	与状态码301类似。但这里的移除是临时的。 客户端会使用Location中给出的URL，重新发送新的HTTP request
303	See Other（参见其他）	类似302
304	Not Modified（未修改）	客户的缓存资源是最新的， 要客户端使用缓存
305	Use Proxy（使用代理）	必须通过代理访问资源， 代理的地址在Response 的Location中
306	未使用	这个状态码当前没使用
307	Temporary Redirect（临时重定向	类似302

状态码的含义 – 4XX

状态码	状态消息	含义
400	Bad Request（坏请求）	告诉客户端，它发送了一个错误的请求。
401	Unauthorized（未授权）	需要客户端对自己认证
402	Payment Required（要求付款）	这个状态还没被使用， 保留给将来用
403	Forbidden（禁止）	请求被服务器拒绝了
404	Not Found（未找到）	未找到资源
405	Method Not Allowed（不允许使用的方法）	不支持该Request的方法。
406	Not Acceptable（无法接受）	
407	Proxy Authentication Required(要求进行代理认证)	与状态码401类似， 用于需要进行认证的代理服务器
408	Request Timeout（请求超时）	如果客户端完成请求时花费的时间太长， 服务器可以回送这个状态码并关闭连接
409	Conflict（冲突）	发出的请求在资源上造成了一些冲突
410	Gone（消失了）	服务器曾经有这个资源，现在没有了， 与状态码404类似
411	Length Required（要求长度指示）	服务器要求在Request中包含Content-Length。
412	Precondition Failed（先决条件失败）	
413	Request Entity Too Large（请求实体太大）	客户端发送的实体主体部分比服务器能够或者希望处理的要大
414	Request URI Too Long（请求URI太长）	客户端发送的请求所携带的URL超过了服务器能够或者希望处理的长度
415	Unsupported Media Type（不支持的媒体类型）	服务器无法理解或不支持客户端所发送的实体的内容类型
416	Requested Range Not Satisfiable（所请求的范围未得到满足）	
417	Expectation Failed（无法满足期望）	

状态码的含义 – 5XX

状态码	状态消息	含义
500	Internal Server Error(内部服务器错误)	服务器遇到一个错误，使其无法为请求提供服务
501	Not Implemented（未实现）	客户端发起的请求超出服务器的能力范围(比如，使用了服务器不支持的请求方法)时，使用此状态码。
502	Bad Gateway（网关故障）	代理使用的服务器遇到了上游的无效响应
503	Service Unavailable（未提供此服务）	服务器目前无法为请求提供服务，但过一段时间就可以恢复服务
504	Gateway Timeout（网关超时）	与状态吗408类似， 但是响应来自网关或代理，此网关或代理在等待另一台服务器的响应时出现了超时
505	HTTP Version Not Supported（不支持的HTTP版本）	服务器收到的请求使用了它不支持的HTTP协议版本。有些服务器不支持HTTP早期的HTTP协议版本，也不支持太高的协议版本

用计算机语言获取HTTP状态码的方法

- 我们可以使用linux里的curl
  - curl -I -m 10 -o /dev/null -s -w "%{http_code}\n" <URL>
- 我们也可以使用Python等语言：

```
import urllib2
urllib2.urlopen('<URL>').code
```

Refer Location

HTTP协议响应头信息

HTTP协议响应头信息

- 通过本知识域，我们会：
  - HTTP响应头的含义
    - 了解常见的HTTP响应头
    - 掌握HTTP响应头的作用
  - HTTP响应头的类型
    - 了解HTTP响应头的名称
    - 掌握HTTP响应头的格式

常用标准响应头字段

- Access：服务器支持哪些请求方法（如GET、POST等）。
- Content-Encoding：文档的编码（Encode）方法。
- Content-Length：表示内容长度。
- Content-Type：表示后面的文档属于什么MIME类型。
- Date：当前的GMT时间。
- Expires：应该在什么时候认为文档已经过期，从而不再缓存它
- Last-Modified：文档的最后改动时间。
- Location：表示客户应当到哪里去提取文档。

常用标准响应头字段

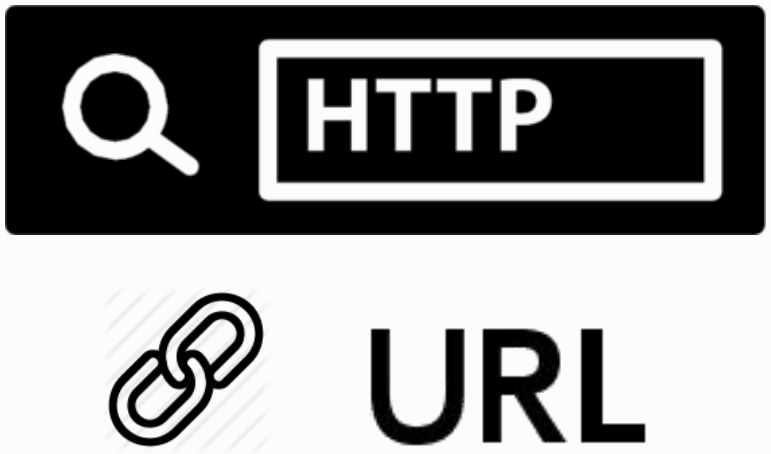
- Refresh：表示浏览器应该在多少时间之后刷新文档，以秒计。
- Server：服务器名字。
- Set-Cookie：设置和页面关联的Cookie。
- WWW-Authenticate：标识访问请求实体的身份验证方案

- 更多的，请参考RFC2612
- 参考文档： http://www.jianshu.com/p/6e86903d74f7

响应头格式

HTTP/1.1 302 Found
Date: Sun, 20 Aug 2017 13:38:54 GMT
Server: Apache/2.4.6 (CentOS) OpenSSL/1.0.1e-fips PHP/5.4.16
Location: https://www.aqzhi.com/
Content-Length: 206
Connection: close
Content-Type: text/html; charset=iso-8859-1

<!DOCTYPE HTML PUBLIC "-//IETF//DTD HTML 2.0//EN"><html><head><title>302 Found</title></head><body><h1>Found</h1> The document has moved here. </body></html>



HTTP协议中的URL

HTTP协议中的URL

- 通过本知识域，我们会：
  - URL的基本构成
    - 了解URL的基本概念
    - 了解URL的结构
    - 掌握URL的编码格式

什么是URL

- URL是统一资源定位符，是互联网上标准资源的地址
- URL包含
  - 协议
  - 用户名:密码
  - 主机 - 子域名.域名.顶级域名（或IP）
  - 端口号
  - 目录/文件名.文件后缀
  - 参数=值
  - 标志
- 格式:
  - 协议://用户名:密码@子域名.域名.顶级域名:端口号/目录/文件名.文件后缀?参数=值#标志
- 相对URL:
  - /目录/文件名.文件后缀?参数=值#标志

URL编码格式

- 只有字母和数字[0-9a-zA-Z]、一些特殊符号“$-_.+!* ‘(),”[不包括双引号]、以及某些保留字，才可以不经过编码直接用于URL。
- 编码格式为16进制，每两个16进制前加百分号（%）
  - 例：“你好”的utf-8码为: \xe4\xbd\xa0\xe5\xa5\xbd
  - “你好”的URL – utf-8格式编码为： %E4%BD%A0%E5%A5%BD

URL同源策略

- URL格式中，协议，主机，端口三部分相同，才能算是同源。
- 浏览器设置里，默认情况下只有同源的内容才能相互操作。
  - 打开site1.com
  - 创建iframe，打开site2.com
  - site1.com的js访问site2.com的内容
  - 这个时候会报错，不是同源不能进行相关操作。