参考：<http://www.jb51.net/article/86598.htm?pc>

Cookie是由服务器端生成，发送给User-Agent,浏览器会将Cookie的key/value保存到某个目录下的文本文件内，下次请求同一网站时就发送该Cookie给服务器。

**Cookie的诞生**

由于HTTP协议是无状态的，而服务器端的业务必须是要有状态的。Cookie诞生的最初目的是为了存储web中的状态信息，以方便服务器端使用。比如判断用户是否是第一次访问网站。目前最新的规范是RFC 6265，它是一个由浏览器服务器共同协作实现的规范。

**Cookie的处理分为：**

服务器像客户端发送cookie

浏览器将cookie保存

之后每次http请求浏览器都会将cookie发送给服务器端

**服务器端的发送与解析**

**发送cookie**

服务器端像客户端发送Cookie是通过HTTP响应报文实现的，在Set-Cookie中设置需要像客户端发送的cookie，cookie格式如下：

Set-Cookie: "name=value;domain=.domain.com;path=/;expires=Sat, 11 Jun 2016 11:29:42 GMT;HttpOnly;secure"  
其中name=value是必选项，其它都是可选项。Cookie的主要构成如下：

name:一个唯一确定的cookie名称。通常来讲cookie的名称是不区分大小写的。

value:存储在cookie中的字符串值。最好为cookie的name和value进行url编码

domain:cookie对于哪个域是有效的。所有向该域发送的请求中都会包含这个cookie信息。这个值可以包含子域(如：

yq.aliyun.com)，也可以不包含它(如：.aliyun.com，则对于aliyun.com的所有子域都有效).

path: 表示这个cookie影响到的路径，浏览器跟会根据这项配置，像指定域中匹配的路径发送cookie。

expires:失效时间，表示cookie何时应该被删除的时间戳(也就是，何时应该停止向服务器发送这个cookie)。如果不设置这个时间戳，浏览器会在页面关闭时即将删除所有cookie；不过也可以自己设置删除时间。这个值是GMT时间格式，如果客户端和服务器端时间不一致，使用expires就会存在偏差。

max-age: 与expires作用相同，用来告诉浏览器此cookie多久过期（单位是秒），而不是一个固定的时间点。正常情况下，max-age的优先级高于expires。

HttpOnly: 告知浏览器不允许通过脚本document.cookie去更改这个值，同样这个值在document.cookie中也不可见。但在http请求张仍然会携带这个cookie。注意这个值虽然在脚本中不可获取，但仍然在浏览器安装目录中以文件形式存在。这项设置通常在服务器端设置。

secure: 安全标志，指定后，只有在使用SSL链接时候才能发送到服务器，如果是http链接则不会传递该信息。就算设置了secure 属性也并不代表他人不能看到你机器本地保存的 cookie 信息，所以不要把重要信息放cookie就对了服务器端设置

cookie示例如下：

var http = require('http');

var fs = require('fs');

http.createServer(function(req, res) {

  res.setHeader('status', '200 OK');

  res.setHeader('Set-Cookie', 'isVisit=true;domain=.yourdomain.com;path=/;max-age=1000');

  res.write('Hello World');

  res.end();

}).listen(8888);

console.log('running localhost:8888')

直接设置Set-Cookie过于原始，我们可以对cookie的设置过程做如下封装：

var serilize = function(name, val, options) {

  if (!name) {

    throw new Error("coolie must have name");

  }

  var enc = encodeURIComponent;

  var parts = [];

  val = (val !== null && val !== undefined) ? val.toString() : "";

  options = options || {};

  parts.push(enc(name) + "=" + enc(val));

  // domain中必须包含两个点号

  if (options.domain) {

    parts.push("domain=" + options.domain);

  }

  if (options.path) {

    parts.push("path=" + options.path);

  }

  // 如果不设置expires和max-age浏览器会在页面关闭时清空cookie

  if (options.expires) {

    parts.push("expires=" + options.expires.toGMTString());

  }

  if (options.maxAge && typeof options.maxAge === "number") {

    parts.push("max-age=" + options.maxAge);

  }

  if (options.httpOnly) {

    parts.push("HTTPOnly");

  }

  if (options.secure) {

    parts.push("secure");

  }

  return parts.join(";");

}

需要注意的是，如果给cookie设置一个过去的时间，浏览器会立即删除该cookie；此外domain项必须有两个点，因此不能设置为localhost:

|  |  |
| --- | --- |
|  | something that wasn't made clear to me here and totally confused me for a while was that domain names must contain at least two dots (.),hence 'localhost' is invalid and the browser will refuse to set the cookie! |

**服务器端解析cookie**

cookie可以设置不同的域与路径，所以对于同一个name value，在不同域不同路径下是可以重复的，浏览器会按照与当前请求url或页面地址最佳匹配的顺序来排定先后顺序。所以当前端传递到服务器端的cookie有多个重复name value时，我们只需要最匹配的那个，也就是第一个。服务器端解析代码如下：

var parse = function(cstr) {

  if (!cstr) {

    return null;

  }

  var dec = decodeURIComponent;

  var cookies = {};

  var parts = cstr.split(/\s\*;\s\*/g);

  parts.forEach(function(p){

    var pos = p.indexOf('=');

    // name 与value存入cookie之前，必须经过编码

    var name = pos > -1 ? dec(p.substr(0, pos)) : p;

    var val = pos > -1 ? dec(p.substr(pos + 1)) : null;

    //只需要拿到最匹配的那个

    if (!cookies.hasOwnProperty(name)) {

      cookies[name] = val;

    }/\* else if (!cookies[name] instanceof Array) {

      cookies[name] = [cookies[name]].push(val);

    } else {

      cookies[name].push(val);

    }\*/

  });

  return cookies;

}

**客户端的存取**

浏览器将后台传递过来的cookie进行管理，并且允许开发者在JavaScript中使用document.cookie来存取cookie。但是这个接口使用起来非常蹩脚。它会因为使用它的方式不同而表现出不同的行为。

当用来获取属性值时，document.cookie返回当前页面可用的（根据cookie的域、路径、失效时间和安全设置）所有的字符串，字符串的格式如下：

|  |  |
| --- | --- |
|  | "name1=value1;name2=value2;name3=value3"; |

当用来设置值的时候，document.cookie属性可设置为一个新的cookie字符串。这个字符串会被解释并添加到现有的cookie集合中。如：

|  |  |
| --- | --- |
|  | document.cookie = "\_fa=aaaffffasdsf;domain=.dojotoolkit.org;path=/" |

设置document.cookie并不会覆盖cookie，除非设置的name value domain path都与一个已存在cookie重复。

由于cookie的读写非常不方便，我们可以自己封装一些函数来处理cookie，主要是针对cookie的添加、修改、删除。

var cookieUtils = {

    get: function(name){

     var cookieName=encodeURIComponent(name) + "=";

     //只取得最匹配的name，value

     var cookieStart = document.cookie.indexOf(cookieName);

     var cookieValue = null;

     if (cookieStart > -1) {

      // 从cookieStart算起

      var cookieEnd = document.cookie.indexOf(';', cookieStart);

      //从=后面开始

      if (cookieEnd > -1) {

       cookieValue = decodeURIComponent(document.cookie.substring(cookieStart + cookieName.length, cookieEnd));

      } else {

       cookieValue = decodeURIComponent(document.cookie.substring(cookieStart + cookieName.length, document.cookie.length));

      }

     }

     return cookieValue;

    },

    set: function(name, val, options) {

      if (!name) {

        throw new Error("coolie must have name");

      }

      var enc = encodeURIComponent;

      var parts = [];

      val = (val !== null && val !== undefined) ? val.toString() : "";

      options = options || {};

      parts.push(enc(name) + "=" + enc(val));

      // domain中必须包含两个点号

      if (options.domain) {

        parts.push("domain=" + options.domain);

      }

      if (options.path) {

        parts.push("path=" + options.path);

      }

      // 如果不设置expires和max-age浏览器会在页面关闭时清空cookie

      if (options.expires) {

        parts.push("expires=" + options.expires.toGMTString());

      }

      if (options.maxAge && typeof options.maxAge === "number") {

        parts.push("max-age=" + options.maxAge);

      }

      if (options.httpOnly) {

        parts.push("HTTPOnly");

      }

      if (options.secure) {

        parts.push("secure");

      }

      document.cookie = parts.join(";");

    },

    delete: function(name, options) {

     options.expires = new Date(0);// 设置为过去日期

     this.set(name, null, options);

    }

   }

**缓存优点**

通常所说的Web缓存指的是可以自动保存常见http请求副本的http设备。对于前端开发者来说，浏览器充当了重要角色。除此外常见的还有各种各样的代理服务器也可以做缓存。当Web请求到达缓存时，缓存从本地副本中提取这个副本内容而不需要经过服务器。这带来了以下优点：

缓存减少了冗余的数据传输，节省流量

缓存缓解了带宽瓶颈问题。不需要更多的带宽就能更快加载页面

缓存缓解了瞬间拥塞，降低了对原始服务器的要求。

缓存降低了距离延时， 因为从较远的地方加载页面会更慢一些。

**缓存种类**

缓存可以是单个用户专用的，也可以是多个用户共享的。专用缓存被称为私有缓存，共享的缓存被称为公有缓存。

**私有缓存**

私有缓存只针对专有用户，所以不需要很大空间，廉价。Web浏览器中有内建的私有缓存——大多数浏览器都会将常用资源缓存在你的个人电脑的磁盘和内存中。如Chrome浏览器的缓存存放位置就在：C:\Users\Your\_Account\AppData\Local\Google\Chrome\User Data\Default中的Cache文件夹和Media Cache文件夹。

**公有缓存**

公有缓存是特殊的共享代理服务器，被称为缓存代理服务器或代理缓存（反向代理的一种用途）。公有缓存会接受来自多个用户的访问，所以通过它能够更好的减少冗余流量。  
下图中每个客户端都会重复的向服务器访问一个资源（此时还不在私有缓存中），这样它会多次访问服务器，增加服务器压力。而使用共享的公有缓存时，缓存只需要从服务器取一次，以后不用再经过服务器，能够显著减轻服务器压力。

注意，我们讨论的所有关于缓存资源的问题，都仅仅针对GET请求。而对于POST, DELETE, PUT这类行为性操作通常不做任何缓存

**新鲜度限值**

HTTP通过缓存将服务器资源的副本保留一段时间，这段时间称为新鲜度限值。这在一段时间内请求相同资源不会再通过服务器。HTTP协议中Cache-Control和 Expires可以用来设置新鲜度的限值，前者是HTTP1.1中新增的响应头，后者是HTTP1.0中的响应头。二者所做的事时都是相同的，但由于Cache-Control使用的是相对时间，而Expires可能存在客户端与服务器端时间不一样的问题，所以我们更倾向于选择Cache-Control。

**Cache-Control**

下面我们来看看Cache-Control都可以设置哪些属性值：

max-age（单位为s）指定设置缓存最大的有效时间，定义的是时间长短。当浏览器向服务器发送请求后，在max-age这段时间里浏览器就不会再向服务器发送请求了。

<html>

 <head>

  <meta http-equiv="Content-Type" content="text/html; charset=utf-8">

  <meta name="viewport" content="width=device-width, initial-scale=1.0, maximum-scale=1.0, user-scalable=no" />

  <meta http-equiv="X-UA-Compatible" content="IE=EDGE" />

  <title>Web Cache</title>

  <link rel="shortcut icon" href="./shortcut.png">

  <script>

  </script>

 </head>

 <body class="claro">

 <img src="./cache.png">

 </body>

</html>

var http = require('http');

var fs = require('fs');

http.createServer(function(req, res) {

  if (req.url === '/' || req.url === '' || req.url === '/index.html') {

    fs.readFile('./index.html', function(err, file) {

      console.log(req.url)

      //对主文档设置缓存，无效果

      res.setHeader('Cache-Control', "no-cache, max-age=" + 5);

      res.setHeader('Content-Type', 'text/html');

      res.writeHead('200', "OK");

      res.end(file);

    });

  }

  if (req.url === '/cache.png') {

    fs.readFile('./cache.png', function(err, file) {

      res.setHeader('Cache-Control', "max-age=" + 5);//缓存五秒

      res.setHeader('Content-Type', 'images/png');

      res.writeHead('200', "Not Modified");

      res.end(file);

    });

  }

}).listen(8888)

当在5秒内第二次访问页面时，浏览器会直接从缓存中取得资源。

public 指定响应可以在代理缓存中被缓存，于是可以被多用户共享。如果没有明确指定private，则默认为public。

private 响应只能在私有缓存中被缓存，不能放在代理缓存上。对一些用户信息敏感的资源，通常需要设置为private。

no-cache 表示必须先与服务器确认资源是否被更改过（依靠If-None-Match和Etag），然后再决定是否使用本地缓存。

如果上文中关于cache.png的处理改成下面这样，则每次访问页面，浏览器都需要先去服务器端验证资源有没有被更改。

fs.readFile('./cache.png', function(err, file) {

    console.log(req.headers);

    console.log(req.url)

    if (!req.headers['if-none-match']) {

      res.setHeader('Cache-Control', "no-cache, max-age=" + 5);

      res.setHeader('Content-Type', 'images/png');

      res.setHeader('Etag', "ffff");

      res.writeHead('200', "Not Modified");

      res.end(file);

    } else {

      if (req.headers['if-none-match'] === 'ffff') {

        res.writeHead('304', "Not Modified");

        res.end();

      } else {

        res.setHeader('Cache-Control', "max-age=" + 5);

        res.setHeader('Content-Type', 'images/png');

        res.setHeader('Etag', "ffff");

        res.writeHead('200', "Not Modified");

        res.end(file);

      }

    }

  });

no-store 绝对禁止缓存任何资源，也就是说每次用户请求资源时，都会向服务器发送一个请求，每次都会下载完整的资源。通常用于机密性资源。