

网易微专业之《前端开发工程师》

学习笔记

开始时间：2016.02.29

《DOM 编程艺术》

基础篇（二）

六、数据通信

HTTP 协议

HTTP 事务



HTTP 报文，分为请求报文和响应报文，它们都是有三部分组成：头行、头部、主体。

请求报文格式：

- 头行：HTTP 方法（get/post） + 主机地址 + HTTP 版本
- 头部：由一些列键值对构成。

GET music.163.com HTTP/1.1
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8 Accept-Encoding: gzip,deflate,sdch Accept-Language: en-US,en;q=0.8,zh-CN;q=0.6,zh;q=0.4 Cache-Control: no-cache Connection: keep-alive Cookie: visited=true; playlist=65117392DNT:1 Host: music.163.com Pragma: no-cache User-Agent: Mozilla/5.0 (Windows NT 6.1; WOW64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/41.0.2272.118 Safari/537.36

响应报文格式

头行：HTTP 协议版本 + HTTP 状态码 + HTTP 状态码描述

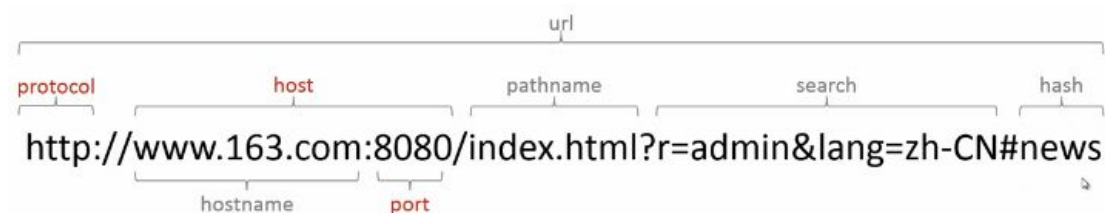
HTTP/1.1 200 OK
Cache-Control: no-cache Cache-Control: no-store Connection: keep-alive Content-Encoding: gzip Content-Language: en-US Content-Type: text/html;charset=utf8 Date: Fri, 17 Apr 2015 01:48:12 GMT Expires: Thu, 01 Jan 1970 00:00:00 GMT Pragma: no-cache Server: nginx Transfer-Encoding: chunked Vary: Accept-Encoding
<!DOCTYPE html><html><head>.....</head><body>.....</body></html>

常用 HTTP 方法

方法	描述	是否包含主体
GET	从服务器获取一份文档	否
POST	向服务器发送需要处理的数据	是
PUT	将请求的主体部分存储在服务器上	是
DELETE	从服务器上删除一份文档	否
HEAD	只从服务器获取文档的首部	否
TRACE	对可能经过代理服务器传送到服务器上去的报文进行追踪	否
OPTIONS	决定可以在服务器上执行哪些方法	否

URL 构成

本笔记由西风潇潇编写，欢迎浏览博客访问更多内容：<http://www.xifengxx.com>



其中，port、pathname、search、hash 是可选部分

HTTP 版本

- HTTP/0.9 1991 年 HTTP 原型，有很多设计缺陷
- HTTP/1.0 很快取代 HTTP/0.9, 成为第一个广泛应用版本
- HTTP/1.0+ 添加持久的 keep-alive 链接，虚拟主机支持，以及代理连接支持，成为非官方的事实标准
- HTTP/1.1 校正结构性缺陷，明确语义，引入重要的性能优化措施，删除不好的特性（当前使用版本）

常见 HTTP 状态码

状态码	描述	原因短语
200	请求成功。一般用于GET和POST方法	OK
301	资源移动。所请求资源自动到新的URL，浏览器自动跳转到新的URL	Moved Permanently
304	未修改。所请求资源未修改，浏览器读取缓存数据	Not Modified
400	请求语法错误，服务器无法理解	Bad Request
404	未找到资源，可以设置个性“404页面”	Not Found
500	服务器内部错误	Internal Server Error

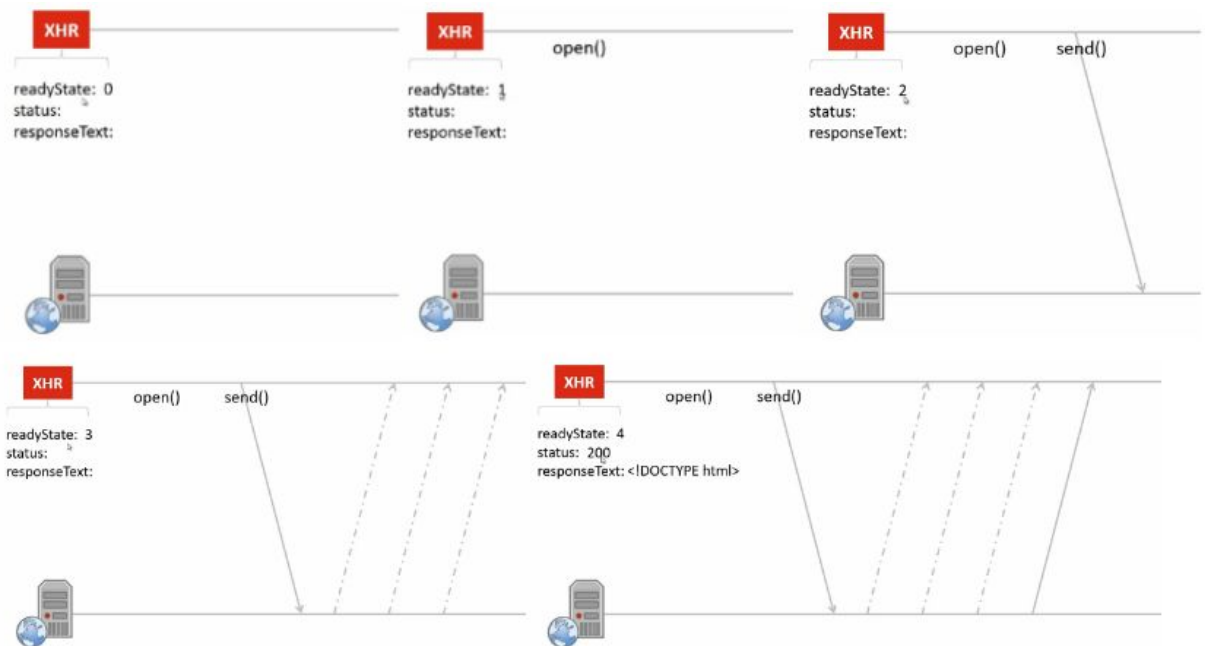
AJAX

Ajax 的概念

AJAX (Asynchronous JavaScript and XML)：异步的 JavaScript 和 XML,即用

JavaScript 去异步的获取 XML 文件作为交换格式，由 Jesse James Garrett 在 2005 年提出。

Ajax 通信流程



Ajax 调用示例

```
var xhr = new XMLHttpRequest(); // 创建XHR对象
xhr.onreadystatechange = function (callback) { // 处理返回数据
    if (xhr.readyState == 4) {
        if ((xhr.status >= 200 && xhr.status < 300) || xhr.status == 304) {
            callback(xhr.responseText);
        } else {
            alert('Request was unsuccessful: ' + xhr.status);
        }
    }
}
xhr.open('get', 'example.json', true);
xhr.setRequestHeader('myHeader', 'myValue'); // 发送请求
xhr.send(null);
```

open 方法:

开启一个请求以备发送，但它不会向服务器端发起正式请求。

```
xhr.open(method, url[ , async = true]);
```

GET	source
POST	./source
DELETE	../source
HEAD	
OPTIONS	
PUT	
CONNECT	
TRACE	
TRACK	

setRequestHeader:

```
xhr.setRequestHeader(header, value);
```

	application/x-www-form-
Content-Type	urlencoded
	multipart/form-data

send 方法:

正式向服务器端发送数据请求。


```
xhr.setRequestHeader('Content-Type'  
| , 'application/x-www-form-urlencoded');
```

```
xhr.send([data = null]);
```

String
FormData

请求参数序列化

```
xhr.open('get'  
  , 'example.json?' + 'name1=value1&name2=value2'  
  , true);
```



```
{  
  name1: value1,  
  name2: value2  
}
```

```
function serialize (data) {  
  if (!data) return '';  
  var pairs = [];  
  for (var name in data) {  
    if (!data.hasOwnProperty(name)) continue;  
    if (typeof data[name] === 'function') continue;  
    var value = data[name].toString();  
    name = encodeURIComponent(name);  
    value = encodeURIComponent(value);  
    pairs.push(name + '=' + value);  
  }  
  return pairs.join('&');  
}
```

GET 请求

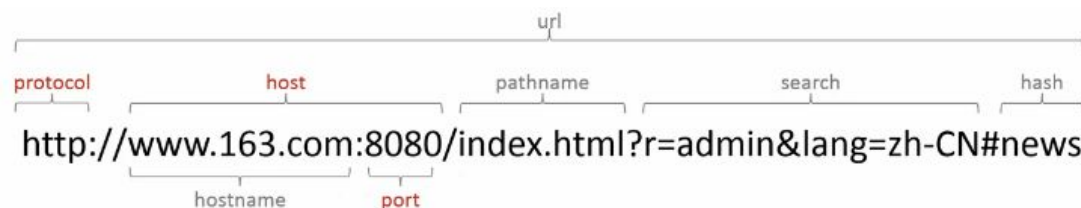
```
var url = 'example.json?' + serialize(formdata);  
xhr.open('get', url, true);  
xhr.send(null);
```

POST 请求

```
xhr.open('post', 'example.json', true);  
xhr.send(serialize(formdata));
```

同源策略

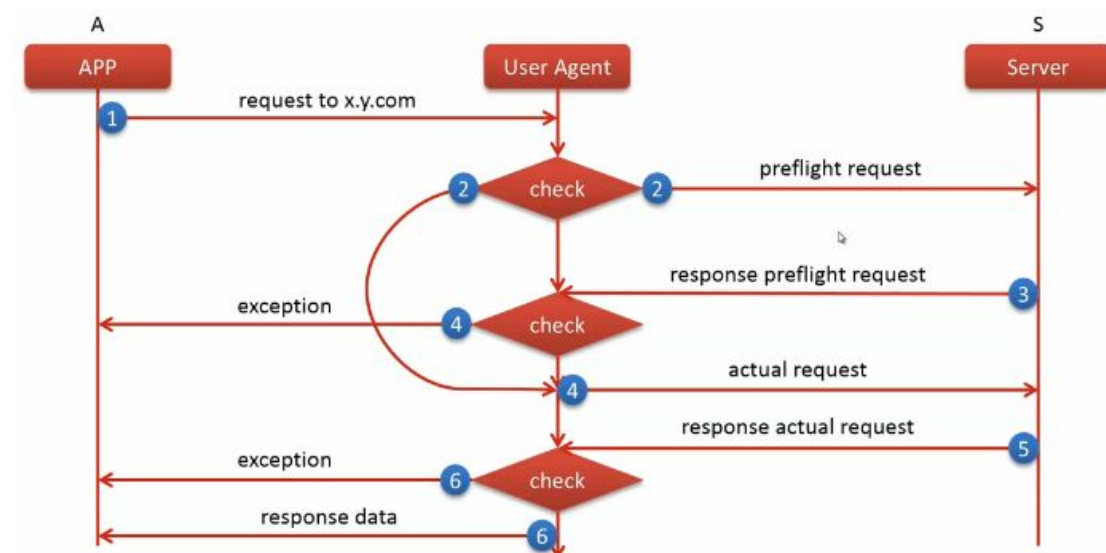
两个页面拥有相同的协议 (Protocol)、端口 (Port)、和主机 (host) 那么这两个页面就是属于同一个源 (Origin)。



跨域资源访问

- 不满足同源策略的资源访问，叫跨域资源访问
- W3C 定义了 **CORS**。
- 现代浏览器已经实现了对 CORS 的支持。

CORS 标准原理（Cross-Origin Resource Sharing）



1. 浏览器捕获到 a.b.com 应用往 x.y.com 的服务器发起的请求
2. 浏览器检查请求情况确定是否需要先做一次预请求来验证 x.y.com 的服务器是否允许发当前请求过去，如果需要发预请求则浏览器发起一个 OPTIONS 的请求到 x.y.com 的服务器验证继续第 3 步，否则直接发送请求到 x.y.com 服务器继续第 5 步
3. 服务器根据浏览器发过来的 header 信息，然后根据服务器端对资源的配置返回资源的实际控制权限配置
4. 浏览器验证预请求返回的信息，判断是否可以将请求发送到 x.y.com 的服务器，如果不行则异常退出，否则继续第 5 步
5. 浏览器发送实际请求至 x.y.com 服务器
6. 服务器返回请求数据及资源控制配置信息至浏览器

本笔记由西风潇潇编写，欢迎浏览博客访问更多内容：<http://www.xifengxx.com>

7. 浏览器验证资源控制信息是否允许当前实际请求的取到数据，如果不允许则异常退出，否则继续第 8 步
8. 浏览器返回 x.y.com 返回的数据至 a.b.com 的应用

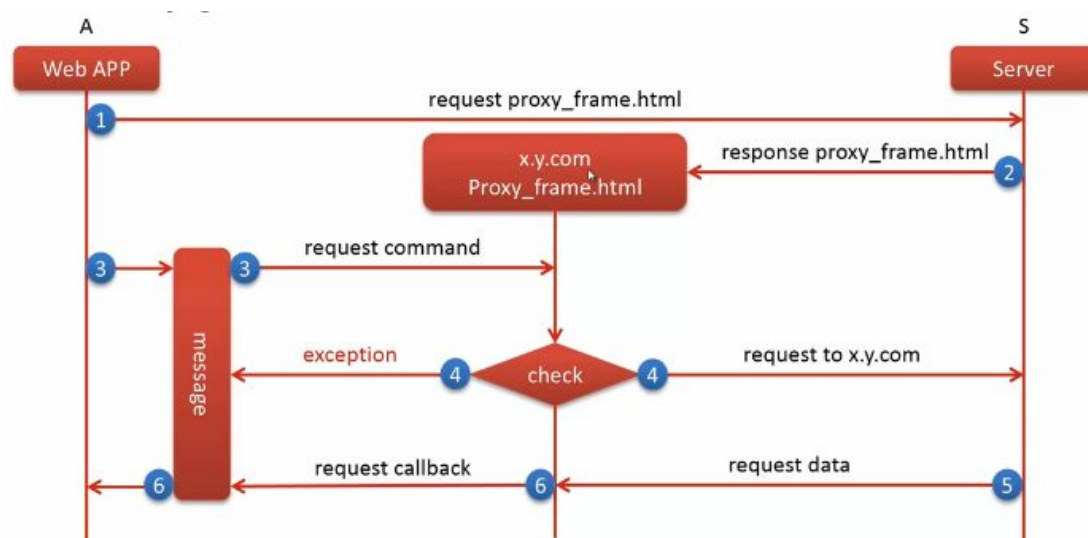
其他跨域技术

- Frame 代理
- JSONP
- Comet
- Web Sockets
- ...

Frame 代理

此模式主要参照 CORS 规范原理，通过在目标服务器放置一个代理文件，然后通过该代理文件来与服务器端进行数据交互，返回数据通过消息通讯返回给上层应用来实现跨域的数据交互。

此方式也支持通过代理文件配置资源可访问的来源。



本笔记由西风潇潇编写，欢迎浏览博客访问更多内容：<http://www.xifengxx.com>

1. 当 a.b.com 的应用要往 x.y.com 的服务器取数据时，首先会用 iframe 载入预先放置在 x.y.com 服务器上的代理文件
2. 服务器端返回做了配置的代理文件
3. 代理文件载入完成后 a.b.com 的应用将要发送的请求指令通过消息通信方式传递给代理文件
4. 代理文件验证 a.b.com 是否在预先配置的白名单中，如果不在则异常返回，否则直接发送请求至 x.y.com 服务器
5. 服务器返回数据至代理文件
6. 代理文件通过消息通讯机制将请求结果返回给 a.b.com 的应用

优点：

- 参照 CORS 标准
- 支持各种请求方法 GET POST PUT DELETE

缺点：

- 需要在目标服务器放置代理文件
- 由于首次发起请求时需要载入代理文件，在载入代理文件之前的所有请求都会存在一定的延时。
- 对于低版本浏览器受限于消息通讯机制的限制，对于并发量大的请求返回时可能存在较大延时。

参考资源：<https://github.com/genify/nej/blob/master/doc/AJAX.md>

JavaScript 跨域与解决方法：

<http://www.cnblogs.com/rainman/archive/2011/02/20/1959325.html>

跨域基本知识：<http://www.cnblogs.com/scottckt/archive/2011/11/12/2246531.html>

跨域资源共享方法：<http://www.cnblogs.com/cat3/archive/2011/06/15/2081559.html>

JavaScript 函数节流：<http://www.alloyteam.com/2012/11/javascript-throttle/>

JSONP

JSON with Padding (填充式 JSON): 利用<script> 可以跨域,请求一段 JavaScript 代码, 然后执行 JavaScript 代码来实现跨域。

案例:



Ajax 请求 GET 方法的封装

方法 get(url, options, callback)

参数

- url {String} 请求资源的 url
- options {Object} 请求的查询参数
- callback {Function} 请求的回调函数, 接收 XMLHttpRequest 对象的 responseText 属性作为参数

返回 void

举例

```
get('/information', {name: 'netease', age: 18}, function (data) {
  console.log(data);
});
```

描述

方法 get(url, options, callback) 是对 Ajax 请求 GET 方法的封装。请写出 get 方法的实现代码。

```
function get(url, options, callback) {
  var xhr = new XMLHttpRequest();
  xhr.onreadystatechange = function(callback) {
    if(xhr.readyState == 4) {
      if((xhr.status >= 200 && xhr.status < 300) || xhr.status == 304) {
        callback(xhr.responseText);
      } else {
        alert('Request was unsuccessful: ' + xhr.status);
      }
    }
  }
}
```

本笔记由西风潇潇编写，欢迎浏览博客访问更多内容：<http://www.xifengxx.com>

```
xhr.open('get', url+'?' +serialize(options), true);
xhr.send(null);
}
function serialize(data) {
    if(!data) return '';
    var pairs = [];
    for(var name in data) {
        if(!data.hasOwnProperty(name)) continue;
        if(typeof data[name] === 'function') continue;
        var value = data[name].toString();
        name = encodeURIComponent(name);
        value = encodeURIComponent(value);
        pairs.push(name + '=' + value);
    }
    return pairs.join('&');
}
```

Ajax 请求 POST 方法的封装

post 函数是对 Ajax 的 POST 请求的封装，语法如下：

post(url, options, callback)

没有返回值，参数说明如下：

url: 请求资源的 url, String 类型

options: 请求的查询参数, Object 类型

callback: 回调函数，接收 XMLHttpRequest 对象的 responseText 属性作为参数，Function 类型

使用示例如下：

```
post('/addUser', {name: 'jerry', age: 1}, function(data) {
    // 处理返回数据
});
```

请写出 post 函数的实现代码，要求浏览器兼容。

```
function post(url, options, callback) {
    //所有现代浏览器均支持 XMLHttpRequest 对象（IE5 和 IE6 使用 ActiveXObject）
    var xhr = window.XMLHttpRequest ? new XMLHttpRequest() : new ActiveXObject(
"Microsoft.XMLHTTP");
    //当请求被发送到服务器时，我们需要执行一些基于响应的任务。每当 readyState 改变时，就会触发 onreadystatechange 事件。
    xhr.onreadystatechange = function () {
        if (xhr.readyState == 4) {
            //http 状态码 200到300是指服务端正常返回;304是告诉客户端取缓存数据
            if ((xhr.status >= 200 && xhr.status < 300) || xhr.status == 304) {
                callback(xhr.responseText);
            } else {
                alert("请求错误: " + xhr.status + " " + xhr.statusText);
            }
        }
    }
}
```

```
    }  
};  
xhr.open('post', url, true);  
//与 get 不同的地方  
xhr.setRequestHeader('Content-Type', 'application/x-www-form-urlencoded');  
xhr.send(serialize(options));  
}  
/**  
 * 序列化对象为查询字符串  
 * @param data {Object} 请求序列化的对象  
 * @return {String}  
 */  
function serialize(data) {  
    if(!data) return '';  
    var pairs = [];  
    for(var name in data) {  
        if(!data.hasOwnProperty(name)) continue;  
        if(typeof data[name] === 'function') continue;  
        var value = data[name].toString();  
        name = encodeURIComponent(name);  
        value = encodeURIComponent(value);  
        pairs.push(name + '=' + value);  
    }  
    return pairs.join("&");  
}
```

七、数据存储

Cookie

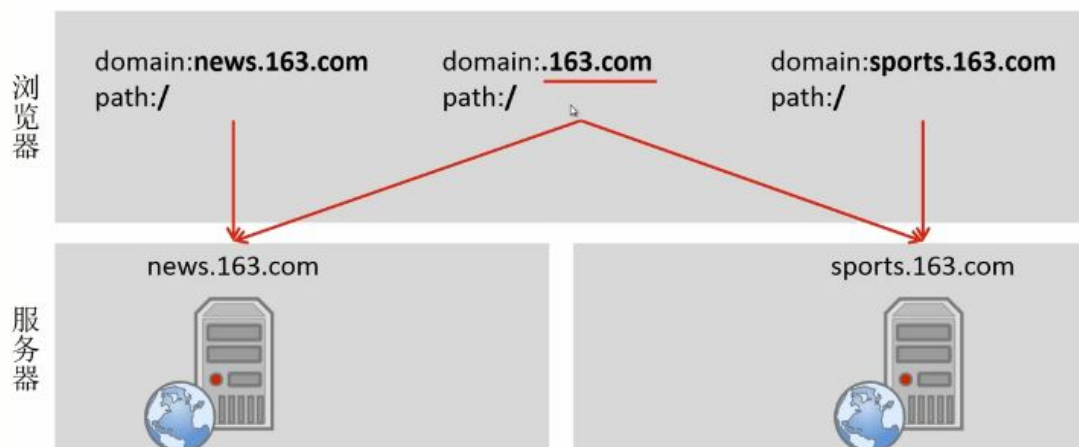
- 浏览器中的 Cookie 是指小型文本文件，通常在 4KB 大小左右。
- Cookie 由键值对构成用，键值对间用；（分号空格）隔开
- 存储在浏览器端，但大部分时候是在服务器端对 Cookie 进行设置，在头文件中 Set-Cookie 来对 Cookie 进行设置。

Cookie 属性

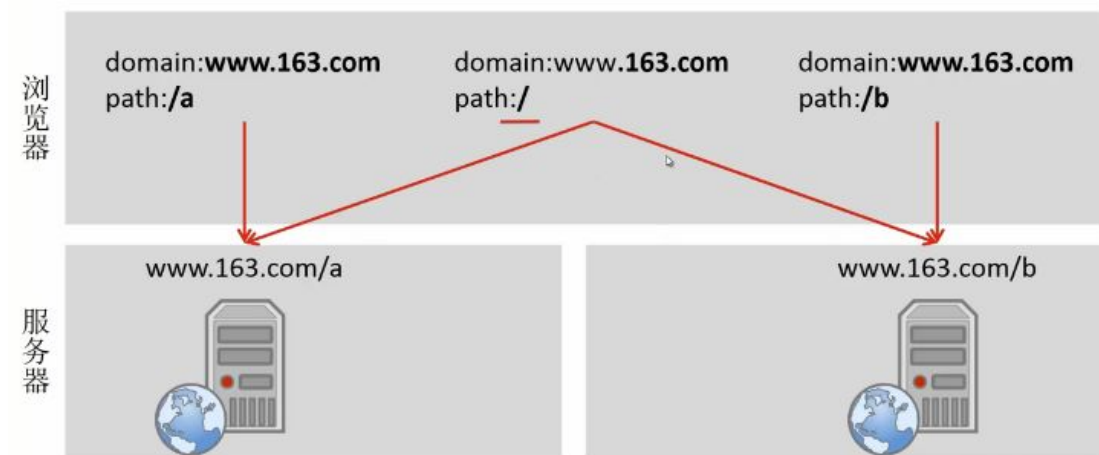
属性名	默认值	作用
Name		名
Value		值
Domain	当前文档域	作用域
Path	当前文档路径	作用路径
Expires/Max-Age	浏览器会话时间	失效时间
Secure	false	https协议时生效

- Name, Value 为必填属性
- Expires(时间戳)、Max-Age(毫秒数值)
- 标识唯一的 Cookie 值：Name + Domain + Path

Cookie 作用域



Cookie 作用路径



Cookie 读取

以下代码，将 Cookie 转化为 JavaScript 对象：

```
function getcookie() {  
    var cookie = {};  
    var all = document.cookie;  
    if (all === '') return cookie;  
    var list = all.split('; ');  
    for (var i = 0, len = list.length; i < len; i++) {  
        var item = list[i];  
        var p = item.indexOf('=');  
        var name = item.substring(0, p);  
        name = decodeURIComponent(name);  
        var value = item.substring(p + 1);  
        value = decodeURIComponent(value);  
        cookie[name] = value;  
    }  
    return cookie;  
}
```

Cookie 设置/修改

1. 为 document.cookie 进行赋值：

```
document.cookie = 'name=value';
```

2. 设置 Cookie 值的封装函数

```
function setCookie(name, value, expires, path, domain, secure) {  
    var cookie = encodeURIComponent(name) + '=' + encodeURIComponent(value);  
    if (expires)  
        cookie += '; expires=' + expires.toGMTString();  
    if (path)  
        cookie += '; path=' + path;  
    if (domain)  
        cookie += '; domain=' + domain;  
    if (secure)  
        cookie += '; secure';  
    document.cookie = cookie;  
}
```



```
cookie += '; secure=' + secure;
document.cookie = cookie;
}
```

删除 Cookie 值

```
function removeCookie(name, path, domain) {
    document.cookie = name + '='
        + '; path=' + path
        + '; domain=' + domain
        + '; max-age=0';
}
```

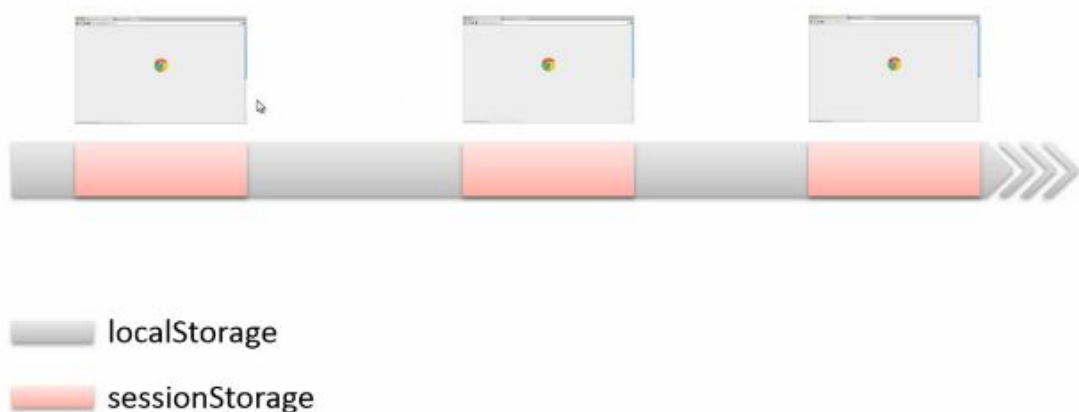
Cookie 缺陷

- 流量代价
- 安全性（明文传递）
- 大小限制

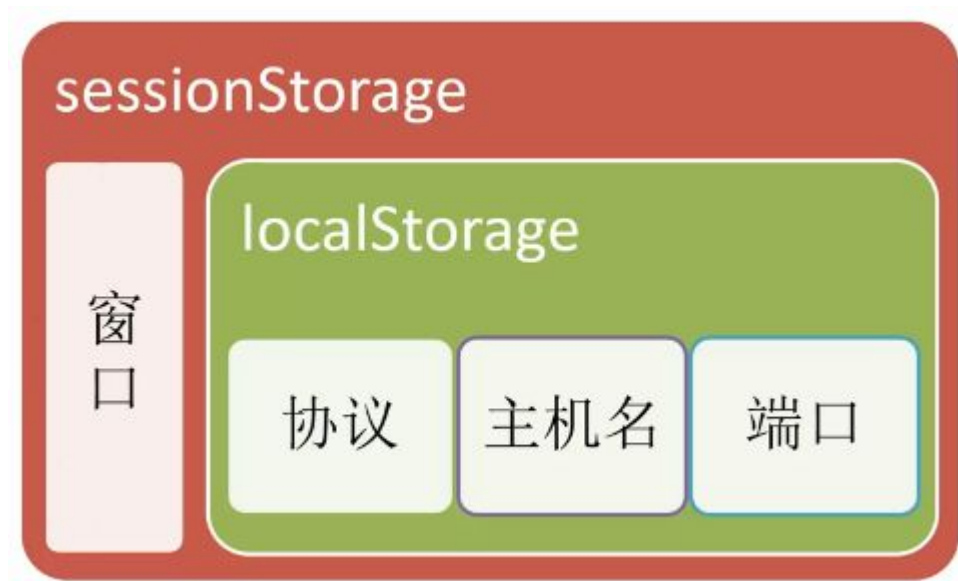
Cookie 作用：一定程度上弥补了 http 无状态协议对交互式 web 应用的影响，它可以使用客户端存储部分信息，维护用户和服务器会话中的状态。

Storage

因为 Cookie 弊端的存在，所以在 HTML5 中提供了 Storage 作为客户端存储的替代方案。根据有效期与作用域的不同，Storage 分为 Local Storage 和 Session Storage，前者在用户不清理的情况下默认时间为永久，后者默认时间则为浏览器。



Storage 作用域



Storage 大小:

不同浏览器对 Storage 实现的不同导致支持大小也不太一样，通常在 5MB 作用。

JS 对象

可以简单的将 Storage 理解为一个对象，其增删查改可以以对象的方式进行。

读取

`localStorage.name`

添加或修改

`localStorage.name = 'Value';`

目前浏览器只支持字符串在 Storage 中的存储。

删除

`delete localStorage.name`

API

W3C 有定义专门的 API 来对 Storage 进行增删查改。

使用 API 操作 Storage 的作用:

可以进行向下兼容的功能，在不支持 Storage 的情况下可以用 Cookie 来代替。

- 获取键值对数量: `localStorage.length`
- 读取:
 - `localStorage.getItem('name')` 获取对应值
 - `localStorage.key(i)` 对应值的索引获取
- 添加/修改: `localStorage.setItem('name', 'value')`
- 删除对应键值: `localStorage.removeItem('name')`
- 删除所有数据: `localStorage.clear()`

八、动画

动画原理：

帧：为动画的最小单位，一个静态的图像。

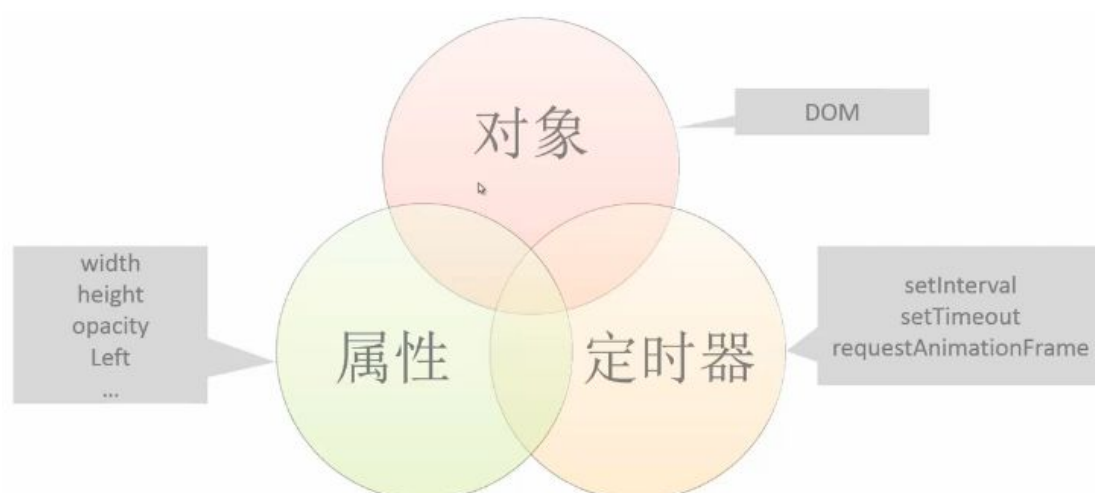
帧频：每秒播放的帧的数量。

一个动画是由很多帧组成的，因为人眼的暂留特性，当图片交替的速度大于每秒 30 帧以上即有动画的感觉。

实现方式

- gif：图像形式存储，容量大，需第三方制图工具制作。
- flash：需要第三方制作工具，不推荐。
- CSS3：实现动画具有局限性
- JavaScript：可实现大部分动画效果

JavaScript 动画三要素



setInterval

本笔记由西风潇潇编写，欢迎浏览博客访问更多内容：<http://www.xifengxx.com>

```
var intervalId = setInterval(func, delay[, param1, param2, ...]);

clearInterval(intervalId);
```

- `func` 为执行改变属性的操作
- `delay` 为触发的时间间隔（毫秒为单位）
- `param1` 为执行时可传入改变属性函数的参数
- `clearInterval` 为清除动画效果

setTimeout

```
var timeoutId = setTimeout(func[, delay, param1, param2, ...]);

clearTimeout(timeoutId);
```

- `func` 为执行改变属性的操作
- `delay` 为触发的时间间隔（毫秒为单位）默认为 0，可选项，不设置的话会立即执行动画。
- `param1` 为执行时可传入改变属性函数的参数

requestAnimationFrame

类似于 `setTimeout` 但是无需设定时间间隔。

此定时器为 HTML5 中的新标准，现代浏览器实现很好，其间隔时间不由用户控制，而是由显示器的刷新频率控制。（市面上的显示器刷新频率为每秒刷新 60 次，因此大概 16.67 毫秒刷新一次，每当浏览器刷新时，便会触发此函数。）

好处：无需设置间隔时间；动画流畅度更高。

```
var requestID = requestAnimationFrame(func);

cancelAnimationFrame(requestID);
```

常见动画

大多数的复杂动画都是有下列的简单动画所组成的。

- 形变，改变元素的宽高
- 位移，改变元素相对位置
- 旋转
- 透明度
- 其他...

动画函数

```
var animation = function(ele, attr, from, to) {
    var distance = Math.abs(to - from);
    var stepLength = distance/100;
    var sign = (to - from)/distance;
    var offset = 0;
    var step = function() {
        var tmpOffset = offset + stepLength;
        if (tmpOffset < distance) {
            ele.style[attr] = from + tmpOffset * sign + 'px';
            offset = tmpOffset;
        } else {
            ele.style[attr] = to + 'px';
            clearInterval(intervalID);
        }
    }
    ele.style[attr] = from + 'px';
    var intervalID = setInterval(step, 10);
}
```

图片轮播

进度条

以下为三种实现方式 (setInterval/setTimeout/requestAnimationFrame)

```
var process = function (prcsswrap, drtn, intrvl, callback) {  
    var width = prcsswrap.clientWidth;  
    var prcss = prcsswrap.getElementsByClassName('prcss')[0];  
    var count = drtn/intrvl;  
    var offset = Math.floor(width/count);  
    var tmpCurrent = CURRENT;  
    var step = function () {  
    }  
    var intervalId = setInterval(step, intrvl);  
};  
  
var step = function () {  
    if (tmpCurrent !== CURRENT) {  
        prcss.style.width = '0px';  
        return;  
    }  
    var des = getNum(prcss.style.width) + offset;  
    if (des < width) {  
        prcss.style.width = getNum(prcss.style.width) + offset + 'px';  
    } else if (des = width) {  
        clearInterval(intervalId);  
        prcss.style.width = '0px';  
        PREV = CURRENT;  
        CURRENT = NEXT;  
        NEXT++;  
        NEXT = NEXT%NUMBER;  
        if (callback)  
            callback();  
    } else {  
        prcss.style.width = width + 'px';  
    }  
}
```

获取对象

修改属性值

触发定时器


```
var process = function (prcsswrap, drtn, intrvl, callback) {
    var width = prcsswrap.clientWidth;
    var prcss = prcsswrap.getElementsByClassName('prcss')[0];
    var count = drtn/intrvl;
    var offset = Math.floor(width/count);
    var tmpCurrent = CURRENT;
    var step = function () {
        if (tmpCurrent !== CURRENT) {
            prcss.style.width = '0px';
            return;
        }
        var des = getNum(prcss.style.width) + offset;
        if (des < width) {
            prcss.style.width = getNum(prcss.style.width) + offset + 'px';
            setTimeout(step, intrvl);
        } else if (des = width) {
            prcss.style.width = '0px';
            PREV = CURRENT;
            CURRENT = NEXT;
            NEXT++;
            NEXT = NEXT%NUMBER;
            if (callback)
                callback();
        } else {
            prcss.style.width = width + 'px';
        }
    }
    var timeoutId = setTimeout(step, intrvl);
};
```

```
var process = function (prcsswrap, drtn, intrvl, callback) {
    var width = prcsswrap.clientWidth;
    var prcss = prcsswrap.getElementsByClassName('prcss')[0];
    var count = drtn/intrvl;
    var offset = Math.floor(width/count);
    var tmpCurrent = CURRENT;
    var step = function () {
        if (tmpCurrent !== CURRENT) {
            prcss.style.width = '0px';
            return;
        }
        var des = getNum(prcss.style.width) + offset;
        if (des < width) {
            prcss.style.width = getNum(prcss.style.width) + offset + 'px';
            requestAnimationFrame(step);
        } else if (des = width) {
            prcss.style.width = '0px';
            PREV = CURRENT;
            CURRENT = NEXT;
            NEXT++;
            NEXT = NEXT%NUMBER;
            if (callback)
                callback();
        } else {
            prcss.style.width = width + 'px';
        }
    }
    var requestId = requestAnimationFrame(step);
};
```

左右移动

本笔记由西风潇潇编写，欢迎浏览博客访问更多内容：<http://www.xifengxx.com>

```
var animation = function (ele, from, to, callback) {
    var distance = Math.abs(to - from);
    var cover = 0;
    var symbol = (to - from)/distance;
    var stepLength = Math.floor((distance*STEP)/SPEED);
    var step = function () {
        var des = cover + stepLength;
        if (des < distance) {
            cover += stepLength;
            ele.style.left = getNum(ele.style.left) + stepLength*symbol + 'px';
        } else {
            clearInterval(intervalId);
            ele.style.left = to + 'px';
            if (callback)
                callback();
        }
    }
    var intervalId = setInterval(step, STEP);
}
```

参考：DEMO 源码