必须当前一个js文件下载并执行完才能下载下一个，当这些文件下载时浏览器会一片空白，现在大部分浏览器都是如此，后来浏览器允许并行js文件下载，但依然会阻塞其他的代码，所以就有了JavaScript第一条定律：将脚本放在body底部

1. ：尽量少的JavaScript文件，每个http请求都会造成额外的负担，所以下100kb比下四个25kb的要快，可以将多个文件整合为一个文件

任何带有defer属性的script标签在DOM加载完成之前都不会被执行，可以被放在文档的任意位置，但只有IE和Firefox

var script = document.createElement ("script");

script.type = "text/javascript";

script.src = "file1.js";

document.getElementsByTagName\_r("head")[0].appendChild(script);

这个技术重点是添加到文档中后立即开始下载，且无论在何处开始下载都不会阻塞其他代码，但有些脚本文件没下载完会带来一些问题，所以其他浏览器实现：

var script = document.createElement(‘script’);

script.type = “text/javascript”;

script.onload = function () {

callback();

}

script.src = “flie.js”;

document.getElementsByTagName(‘head’)[0].appendChild(‘script’);

但IE支持另一种实现方式，IE有一个readystatechange事件，script有五个属性值，当脚本下载完后可能为loaded或complete属性，所以：

var script = document.createElement ("script")

script.type = "text/javascript";

//Internet Explorer

script.onreadystatechange = function(){

if (script.readyState == "loaded" || script.readyState == "complete"){

script.onreadystatechange = null;

alert("Script loaded.");

}

};

script.src = "file1.js";

document.getElementsByTagName\_r("head")[0].appendChild(script);

封装一个IE和其他浏览器都能用的函数：

function loadScript(url, callback){

var script = document.createElement ("script")

script.type = "text/javascript";

if (script.readyState){ //IE

script.onreadystatechange = function(){

if (script.readyState == "loaded" || script.readyState == "complete"){

script.onreadystatechange = null;

callback();

}

};

} else { //Others

script.onload = function(){

callback();

};

}

script.src = url;

document.getElementsByTagName\_r("head")[0].appendChild(script);

}

第一个参数是”file.js”，第二个是当JavaScript接收完成时触发的回调函数

可以在页面中动态加载很多js文件，但浏览器不保证文件的加载顺序，只有Firefox和Opera能保证按照你给出的顺序执行，

可以将下载文件操作串联在一起保证它们的顺序

loadScript("file1.js", function(){

loadScript("file2.js", function(){

loadScript("file3.js", function(){

alert("All files are loaded!");

});});});

虽然此方法可行，但有些麻烦，如果顺序很重要，那么更好的办法是按照顺序放在一个大文件里

XMLHttpRequest Script Injection XHR脚本注入

先创建一个XHR对象，然后下载JavaScript文件，再用一个动态script元素将JavaScript代码注入页面

var xhr = new XMLHttpRequest();

xhr.open("get", "file1.js", true);

xhr.onreadystatechange = function(){

if (xhr.readyState == 4){

if (xhr.status >= 200 && xhr.status < 300 || xhr.status == 304){

var script = document.createElement ("script");

script.type = "text/javascript";

script.text = xhr.responseText;

document.body.appendChild(script);

}

}

};

xhr.send(null);

JavaScript文件必须与页面放在一个域内，不能从CDNs(内容投递网络)因此大型网页通常不用此方法

推荐的非阻塞模式

<script type="text/javascript" src="loader.js"></script>

<script type="text/javascript">

loadScript("the-rest.js", function(){

Application.init();

});

</script>

将此代码放到</body>之前，第一不会影响页面其他部分显示，其次，当第二部分的js文件下载完时所有的dom都已经准备好

另一个是将loadScript()函数直接嵌入在页面中省一次http请求

<script type="text/javascript">

function loadScript(url, callback){

var script = document.createElement ("script")

script.type = "text/javascript";

if (script.readyState){ //IE

script.onreadystatechange = function(){

if (script.readyState == "loaded" ||

script.readyState == "complete"){

script.onreadystatechange = null;

callback();

}

};

} else { //Others

script.onload = function(){

callback();

};

}

script.src = url;

document.getElementsByTagName\_r("head")[0].appendChild(script);

}

loadScript("the-rest.js", function(){

Application.init();

});

</script>

总结

每次遇到JavaScript标签都得停下来等它下载并执行，再处理其它部分

1. 将script标签放在body底部</body>上方
2. 尽可能少的script标签，不论外部脚本还是内联代码
3. 有几种方法可以非阻塞方式下载JavaScript

——为script添加defer属性(IE和Firefox3.5以上版本)

——动态创建script标签，用它下载并执行代码

——用XHR对象下载代码，并注入到页面中

第二章 数据访问

数据存储位置关系到访问速度，在JavaScript中有四种基本的数据位置：

直接量：直接量仅仅代表自己，不存储于特定位置，JavaScript中直接量有number，string，boolean，null，undefined，function，数组，正则表达式，对象

变量：开发人员用var关键字创建用于存储数据

数组项：具有数字索引，用于储存一个JavaScript数组对象

对象成员：具有字符串索引，用于存储一个JavaScript对象

尽量使用直接量和局部变量，为此有几种模式来优化：

1. 管理作用域：函数执行找变量，从作用域链顶端开始向下搜索，

正是搜索过程影响了性能

1. 标示符识别性能：作用域链上不同深度的标示符影响了速度，总是局部变量最快，全局变量总是最慢的

function initUI(){

var doc = document,

bd = doc.body,

links = doc.getElementsByTagName\_r("a"),

i = 0,

len = links.length;

while(i < len){

update(links[i++]);

}

doc.getElementById("go-btn").onclick = function(){

start();

};

bd.className = "active";

}

首先将document引入局部变量doc中，现在访问全局document是一次而不是三次，虽然一个不会有太大影响但多了还是很可观

对象成员：处理对象成员的方式和变量相似，当xxx被调用时，

先从对象实例找，没有找对象原型，可以使用hasOwnProperty()

和in来确定一个对象是否有特定名称的实例成员，只不过in会找原型上有没有，在原型上搜索比在实例本身中搜索要慢，原型链越深越耗时间，

嵌套成员：成员嵌套越深消耗时间越长，location.herf快于window.

location.herf更快与window.location.herf.toString()，尽量少除非必须访问对象

function hasEitherClass(element, className1, className2){

return element.className == className1 || element.className == className2;

}可以写成：

function hasEitherClass(element, className1, className2){

var currentClassName = element.className;

return currentClassName == className1 || currentClassName == className2;

}

局部变量访问速度要比对象快得多，在一个函数中要多次用到同一个对象属性最好存到局部变量中，再处理嵌套成员时特别重要

，除非对象成员值会变，否则一个函数中对象成员不要超过一次搜索

总结

数据存储位置关系到访问速度

1.在JavaScript中数据存储位置可以对代码整体性能产生影响，直接量和局部变量访问最快，数组项和对象成员需要更长的时间

2.局部变量访问比域外变量快，因为它们位于作用域链顶端，全局变量总是最慢的

3.避免使用with语句，小心使用try-catch因为catch字句会造成同样效果

4.嵌套对象会造成巨大性能影响，尽量少用，属性在原型链中的位置越深，访问它的速度越慢

5.可以通过这种方式来提高代码性能：将经常使用的数组项，对象成员和域外变量存入局部变量中。

1. DOM编程

对DOM操作代价高昂，本章讨论三类问题：  
访问和修改DOM元素

修改DOM元素的样式，从而导致重绘和重新排版

通过DOM事件处理用户响应

1. 浏览器世界中的DOM：

一种独立于语言用来操作html和xml的一类标准编程接口，DOM为JavaScript语言中的重要组成部分

DOM访问和修改

function innerHTMLLoop() {

for (var count = 0; count < 15000; count++) {

document.getElementById('here').innerHTML += 'a';

}

}

在每次循环单元中都对DOM元素访问两次，一次读取内容，一次写入它

更有效率的方法，在循环结束时一次性写入

function innerHTMLLoop2() {

var content = '';

for (var count = 0; count < 15000; count++) {

content += 'a';

}

document.getElementById('here').innerHTML += content;

}

InnerHTML与DOM方法比较

InnerHTML是快于DOM方法，但除了基于webkit的浏览器如Safari和Chrome，所以两者无太大分别，看你的用户或你的编程习惯

节点克隆

使用DOM更新页面内容也可以克隆已有元素，而不是创建新的

但提升并不多

HTML集合

 document.getElementsByName()

 document.getElementsByClassName()

 document.getElementsByTagName\_r()

是一种数组对象，有length属性，可以通过数字索引访问属性，HTML集合会随着文档更新而更新，这就是效率低的原因，每次更新都会重复查询文档的操作

昂贵的集合

var alldivs = document.getElementsByTagName\_r('div');

for (var i = 0; i < alldivs.length; i++) {

document.body.appendChild(document.createElement('div'))

}

看似简单倍增div元素，其实是死循环，length属性会实时更新，

不但逻辑错误，速度也会很慢，第四章将要讨论不建议用数组length属性作为循环判段标准，集合的length更慢

将一个HTML集合拷贝给一个数组：

function toArray(coll) {

for (var i = 0, a = [], len = coll.length; i < len; i++) {

a[i] = coll[i];

}

return a;

}

var coll = document.getElementsByTagName\_r('div');

var ar = toArray(coll);

//slower

function loopCollection() {

for (var count = 0; count < coll.length; count++) {

}

}

// faster

function loopCopiedArray() {

for (var count = 0; count < arr.length; count++) {

}

}

将集合的length属性缓存到变量中可以优化因集合实时更新反复查询文档造成的性能损失

访问集合元素时用局部变量

// slow

function collectionGlobal() {

var coll = document.getElementsByTagName\_r('div'),

len = coll.length,

name = '';

for (var count = 0; count < len; count++) {

name = document.getElementsByTagName('div')[count].nodeName;

name = document.getElementsByTagName('div')[count].nodeType;

name = document.getElementsByTagName('div')[count].tagName;

}

return name;

};

// faster

function collectionLocal() {

var coll = document.getElementsByTagName('div'),

len = coll.length,

name = '';

for (var count = 0; count < len; count++) {

name = coll[count].nodeName;

name = coll[count].nodeType;

name = coll[count].tagName;

}

return name;

};

// fastest

function collectionNodesLocal() {

var coll = document.getElementsByTagName\_r('div'),

len = coll.length,

name = '',

el = null;

for (var count = 0; count < len; count++) {

el = coll[count];

name = el.nodeName;

name = el.nodeType;

name = el.tagName;

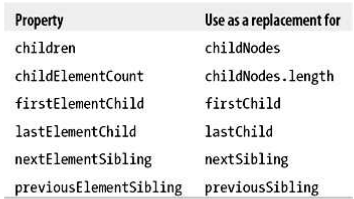
}

return name;

};

元素节点

在许多情况下只有元素节点需要被访问，像过滤非元素节点这些检查和过滤都是非必须的DOM操作



选择器API

querySelectorAll()不返回HTML集合

var errs = document.querySelectorAll('div.warning, div.notice');联合查询也很方便

重绘和重排版

当浏览器下载完所有页面标记，JavaScript，css和图片之后，它解析文件并创建两个内部结构，一颗DOM树表示页面结构，一颗渲染树表示如何显示DOM结构，如果改变元素的几何属性和位置，那么将重绘和重排版，重绘和重排版是负担很重的操作，

重排版耗时比重绘要多

重排版会发生什么

会发生重排版的时候

添加或删除可见的DOM元素，元素位置改变，元素尺寸改变，内容改变从而引起元素大小的改变，最初的页面渲染，浏览器窗口改变尺寸

**1、尽量减少HTTP请求个数——须权衡**

合并图片（如css sprites，内置图片使用数据）、合并CSS、[JS](http://www.18sucai.com/js" \t "http://www.18sucai.com/article/_blank)，这一点很重要，但是要考虑合并后的文件体积。

**4、避免空的src和href**

留意具有这两个属性的标签如link，script，img，iframe等；

**6、把CSS放到顶部**

实现页面有秩序地加载，这对于拥有较多内容的页面和网速较慢的用户来说更为重要，同时，HTML规范清楚指出样式表要放包含在页面的<head />区域内；

**7、把JS放到底部**

HTTP/1.1 规范建议，浏览器每个主机名的并行下载内容不超过两个，而问题在于脚本阻止了页面的平行下载，即便是主机名不相同

**11、精简CSS和JS**

目的就是减少下载的文件体积，可考虑压缩[工具](http://www.18sucai.com/Toolbox/index.aspx" \t "http://www.18sucai.com/article/_blank)JSMin和Y[UI](http://www.18sucai.com/article/193" \t "http://www.18sucai.com/article/_blank) Compressor。

**34、保持单个内容小于25K**

因为iPhone不能缓存大于25K的文件。注意这里指的是解压缩后的大小。由于单纯gizp压缩可能达不要求，因此精简文件就显得十分重 要。

1. 有几种方法可以非阻塞方式下载JavaScript

——为script添加defer属性(IE和Firefox3.5以上版本)

——动态创建script标签，用它下载并执行代码

——用XHR对象下载代码，并注入到页面中