一.注意作用域

1.避免全局查找

使用全局变量和函数比局部开销要大，因为要涉及作用域链上的查找。

function undateUI(){

var imgs = document.getElementsByTagName(“img”);

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

imgs[i].title = document.title + " image " + i;

}

var msg = document.getElementById("msg");

msg.innerHTML = "update complete"

}

//函数包含了三个对document对象的引用,如果页面上有多个图片,那么for循环中的document引用就会被执行N次,每次都会进行作用域链查找.通过创建一个局部变量可以解决问题.

function undateUI(){

var doc = document;

var imgs = doc.getElementsByTagName(“img”);

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

imgs[i].title = doc.title + " image " + i;

}

var msg = doc.getElementById("msg");

msg.innerHTML = "update complete"

}

2.避免with语句

在性能非常重要地方要避免使用with语句，因为with语句会创建自己的作用域。由于额外的作用域链查找，with语句中执行的代码肯定比外面执行的代码更慢。

二、最小化语句数

1.多个变量的声明

var a =1;

var b=2;

var c =3;

var reg = /d+/;

var str = ‘sfasg’;

//改动

var a =1，

b=2，

c =3，

reg = /d+/，

str = ‘sfasg’;

2.插入迭代值

var name = values[i];

i++;

//优化

var name = values[i++];

3.使用数组和对象字面量形式

三、优化DOM交互

1.最小化现场更新

每一处更改,不管是插入一个字符串还是移除整个片段,都有一个性能惩罚,因为浏览器要重新计算无数尺寸以进行更新.现场更新越多,代码完成执行所花时间越长.

exp:

var list = document.getElementById("MYLIST"),

item,

i;

for(i =0;i<10;i++){

item = document.createElement("li");

list.appendChild(item);

item.appendChild(document.createTextNode("Item " + i))

}

此代码为列表添加了10个项目,添加每个项目时,都有两个现场更新,其中一个是添加li元素,另外一个是给他添加文本节点.这样相当于总共要完成20个现场更新.

要修正这个性能瓶颈,需要减少现场更新的数量.有两种方法,一种是将列表从页面中移除,最后进行更新,最好再将列表插回到同样的位置.这个方法不理想,因为在每次页面更新时候他都会不必要的闪烁.

第二个方法使用文档片段来构建dom结构,接着将其添加到list元素中,这个方法规避了现场更新和页面闪烁的问题.exp:

var list = document.getElementById("MYLIST"),

fragment = document.createDocumentFragment(),

item,

i;

for(i =0;i<10;i++){

item = document.createElement("li");

fragment.appendChild(item);

item.appendChild(document.createTextNode("Item " + i))

}

list.appendChild(fragment);

这个例子中只有一次现场更新,它发生在所有项目创建好之后.文档片段作为一个临时的占位符,放置新创建的项目.然后使用appendChild()将所有项目添加到列表中.

牢记当给appendChild()传入文档片段时,只有片段中的子节点被添加到目标,片段本身不会被添加的.

一旦需要更新DOM,首先考虑使用文档片段来构建DOM结构,然后再将其添加到现存的文档中.

2.使用innerHTML

创建节点方法有三种:createElement(),appendChild()和innerHTML.对于小的DOM更改而言,三者差不多;对于大的DOM修改,使用innerHTML要比使用标准DOM方法创建同样的DOM结构快很多.

原理:当把innerHTML设为某个值的时候,后台会创建一个 HTML解析器,然后使用内部的DOM调用来创建DOM结构,而非基于js的DOM调用.由于内部方法是编译好的而非解释执行的,所以执行快很多.

exp:(基于前面的例子)

var list = document.getElementById("MYLIST"),

html="",

i;

for(i =0;i<10;i++){

html += "<li>Item " + i + "</li>";

}

list.innerHTML = html;

虽然在字符串上拼接上有些性能损失,但这种方式还是要比进行多个DOM操作更快.使用他的关键在于(和其他DOM操作一样)最小化调用它的次数.如以下代码要避免:

var list = document.getElementById("MYLIST"),

html="",

i;

for(i =0;i<10;i++){

list.innerHTML += "<li>Item " + i + "</li>";

}

3.使用事件代理

4.注意HTMLCollection

任何时候要访问HTMLCollection,不管他是一个属性还是一个方法,都是在文档上进行一个查询,这个查询开销很大.最小化访问HTMLCollection的次数可以极大滴改进脚本的性能.

优化HTMLCollection访问最重要的是循环.

exp:

var images = document.getElementsByTagName("img"),i,len;

for(i =0;len=images.length;i<len;i++){

//handle

}

//这里关键在于长度length存入了len变量,而不是每次都去访问HTMLCollection的length属性.当在循环中使用HTMLCollection时候,下一步应该是获取要使用的

项目的引用.如下所示可以避免在循环体内多次调用HTMLCollection:

exp:

var images = document.getElementsByTagName("img"),image,i,len;

for(i =0;len=images.length;i<len;i++){

image = images[i]

//handle

}

这段代码添加了image变量,保存了当前的图像.这之后,在循环内就没有理由再访问images的HTMLCollection了

注意:编写代码时候,一定要知道何时返回HTMLCollection对象,主要是以下场景:

\*进行了对getElementsByTagName调用;

\*获取了元素的childNode属性;

\*获取了元素的attribute属性;

\*访问了特殊的集合,如document.forms,document.images等.