潜力无限的JavaScript

一、浏览器渲染与JavaScript执行原理

周志明 @ 开发二部
2013年4月

潜力无限

- 越来越多的应用使用JavaScrip来完成
  - ✓ Window 8后支持原生程序开发。
  - ✓ HTML5下的程序与游戏体验。
  - ✓ Node.js支持下的服务端应用。
- 钱力无限
  - ✓ 不受OS限制，原生态无污染的天然跨平台语言。
  - ✓ 会JavaScript的人很多，精JavaScript的人极少。
  - ✓ Web时代的银弹，随着Web流行而发展。

不是《零基础学会JavaScript》

不是《21天精通JavaScript语言》

不是《论前端开发人员的修养》

只讨论一些JavaScript令人又爱又恨的特性

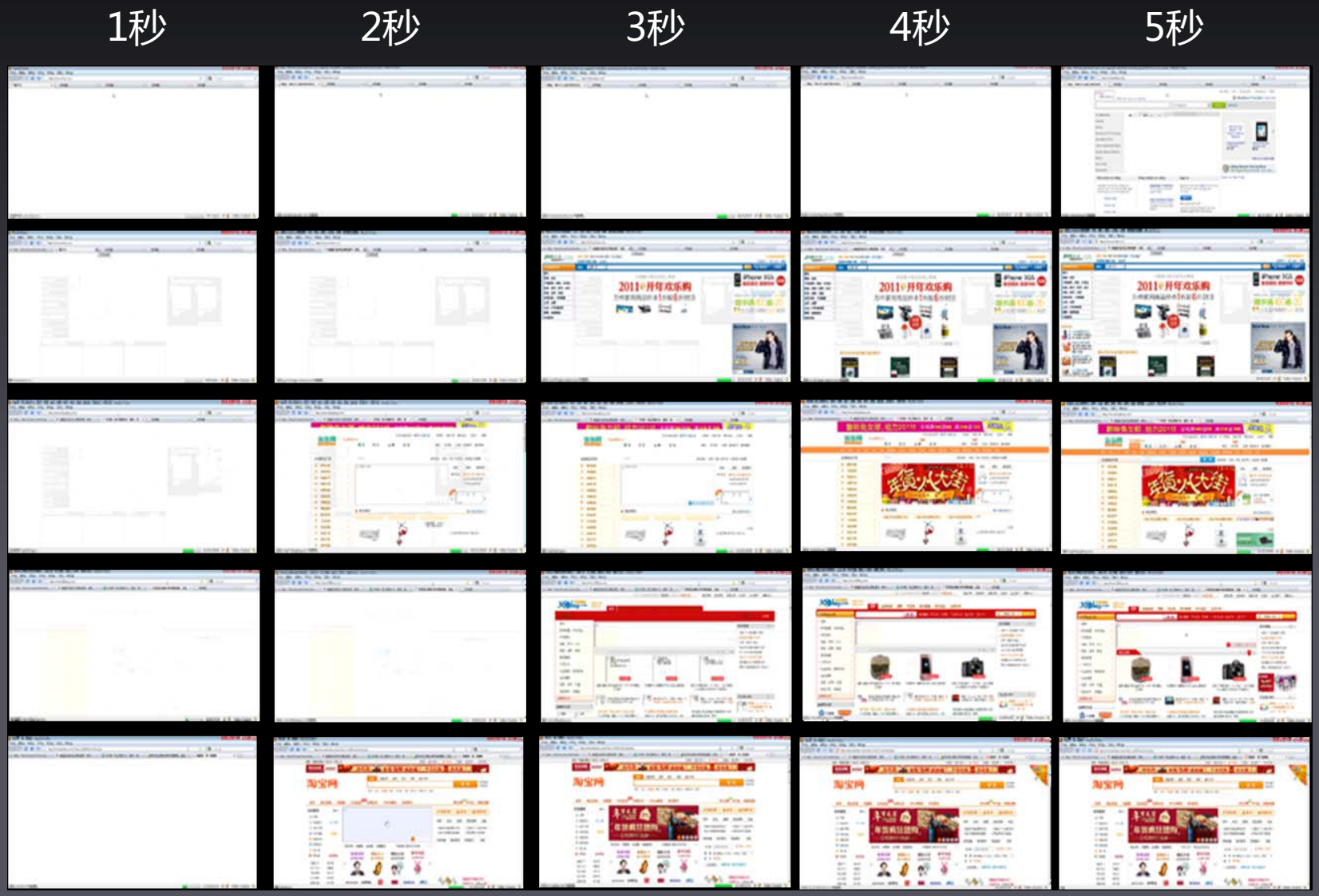
目录

1. 浏览器渲染与JavaScript执行原理
   - ✓ 浏览器渲染和执行引擎、阻塞、异步、定时器、事件流
2. JavaScript进阶基础知识
   - ✓ 执行上下文、变量对象、this指针、作用域链
3. 面向对象的JavaScript
   - ✓ 加载、执行、函数对象、原型继承、闭包、柯里化
4. 性能与JavaScript
   - ✓ 性能陷阱、“看起来快些”的技巧、HTTP头优化、问题排查工具

用户和程序员的小小小愿望

我们目标是没有  于白屏！

网站渲染速度



阻塞渲染

产生白屏，是浏览器的UI渲染的过程被阻塞了。

问题：那到底是什么阻塞它？

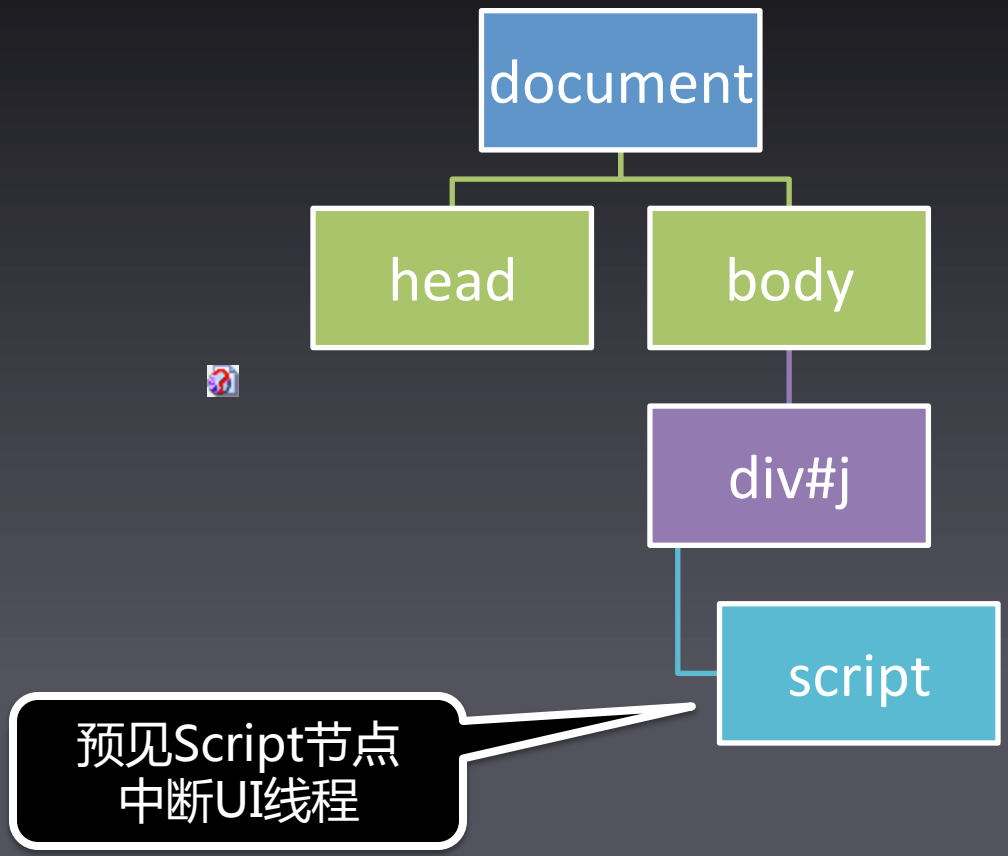
一、浏览器渲染过程

浏览器渲染过程

- 渲染是以流式进行的，不需要得到完整的数据再渲染，如HTML文件下载多少就渲染多少。
- 绝大部分HTML外部的资源都不会阻塞渲染，如CSS、图片、Flash等等，没有下载完图片就留个位置空在那里。
- 绝大部分的HTML元素都是渲染出DOM立刻显示。
- HTML解析过程是从前往后，不可逆（注1）的过程。但是会出现reflow和repaint。

浏览器渲染过程

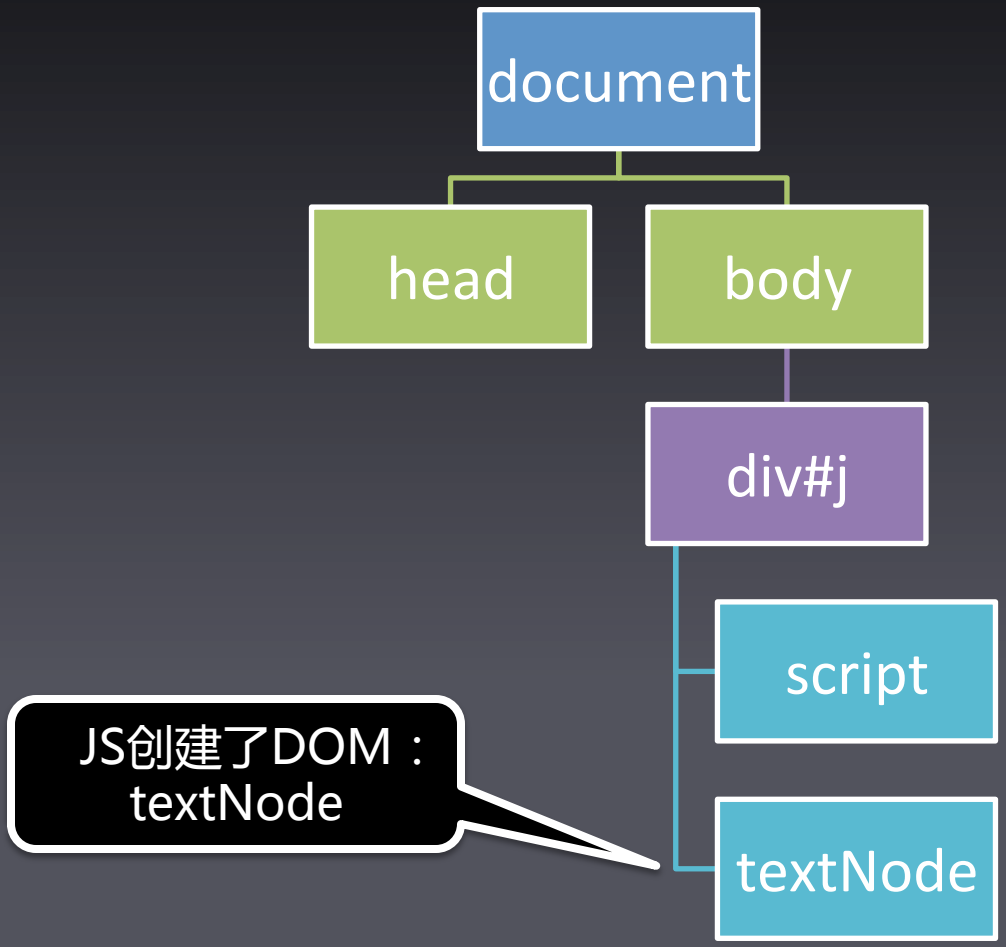
```
<!DOCTYPE HTML>
<html lang="zh">
  <head>head</head>
  <body>
    <divid= “j">
      <script>
        $.write('helloworld');
      </script>
    </div>
    ...
  </body>
</html>
```



浏览器暂停渲染HTML将script交由js引擎编译执行

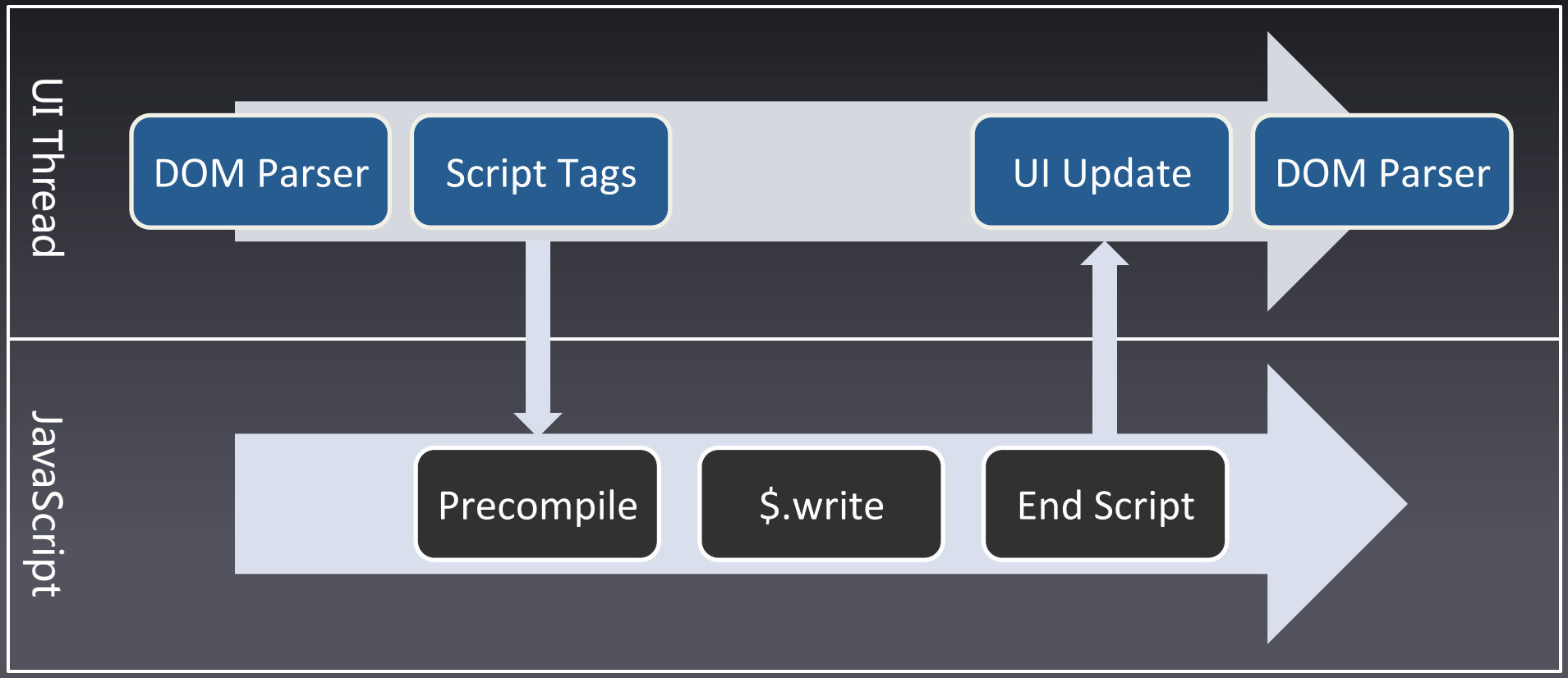
浏览器渲染过程

```
<!DOCTYPE HTML>
<html lang="zh">
  <head>head</head>
  <body>
    <divid= “j">
      <script>
        $.write('helloworld');
      </script>
    </div>
    ...
  </body>
</html>
```

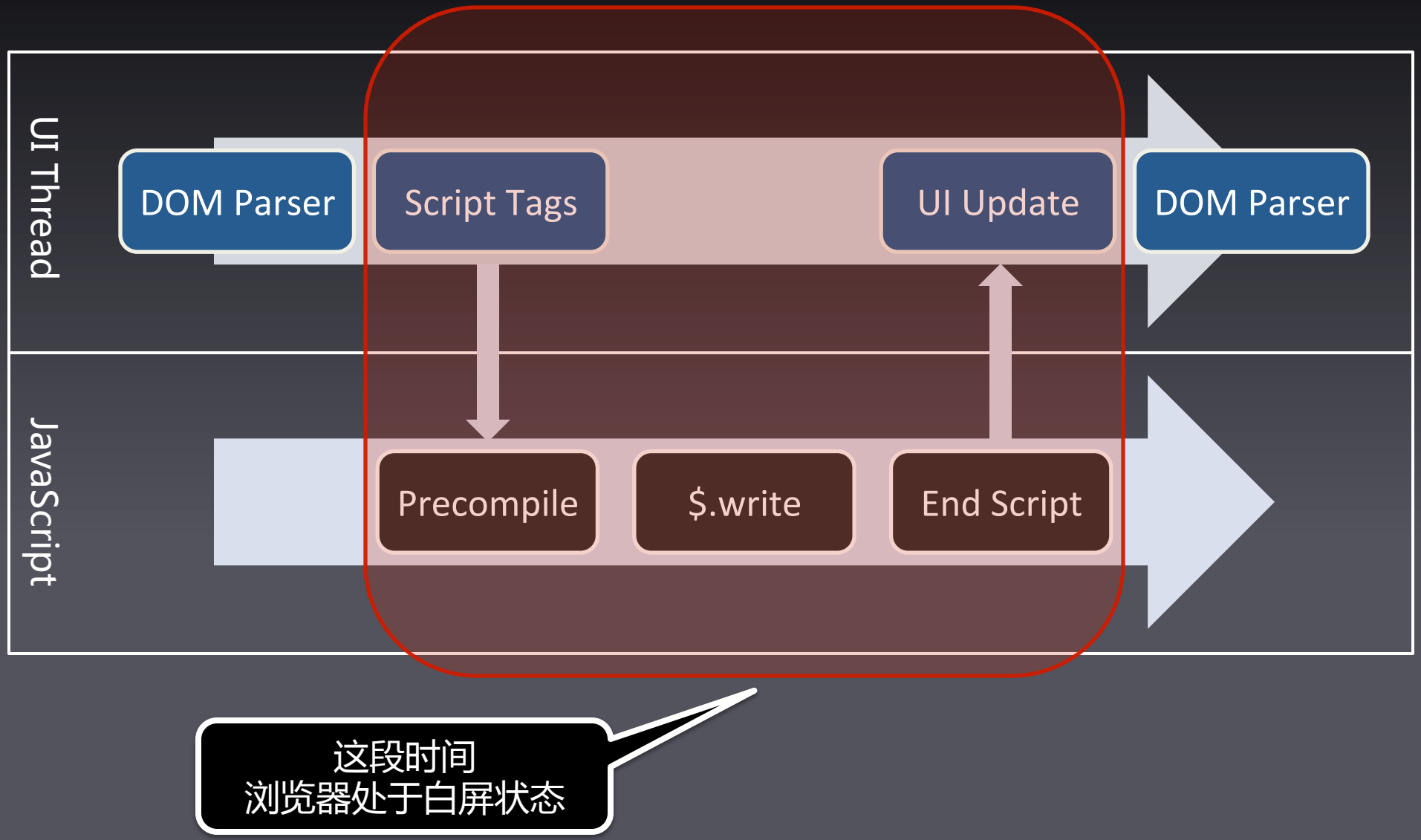


JS引擎执行结束，产生textNode，继续HTML渲染

浏览器渲染过程


UI Thread
DOM Parser
Script Tags
UI Update
DOM Parser
JavaScript
Precompile
$.write
End Script

浏览器渲染过程


UI Thread
DOM Parser
Script Tags
UI Update
DOM Parser
JavaScript
Precompile
$.write
End Script
这段时间
浏览器处于白屏状态

浏览器渲染过程

```
<!DOCTYPE HTML>
<html lang="zh">
  <head>head</head>
  <body>
    <divid= “j">
      <script>
        setTimeout(function(){  $.write(  ‘helloworld<br/>’  );  }, 0);
      </script>
    </div>
    …
  </body>
</html>
```

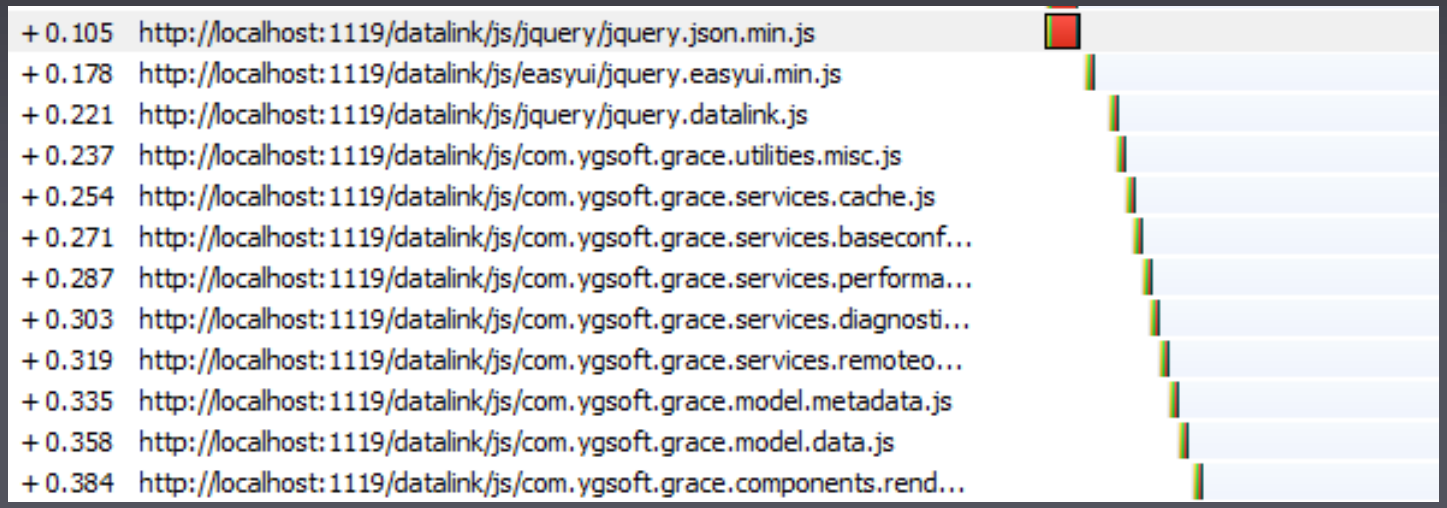
DEMO示例

结论

因为JavaScript能够改变UI的能力，
所以它必须阻塞UI渲染线程。

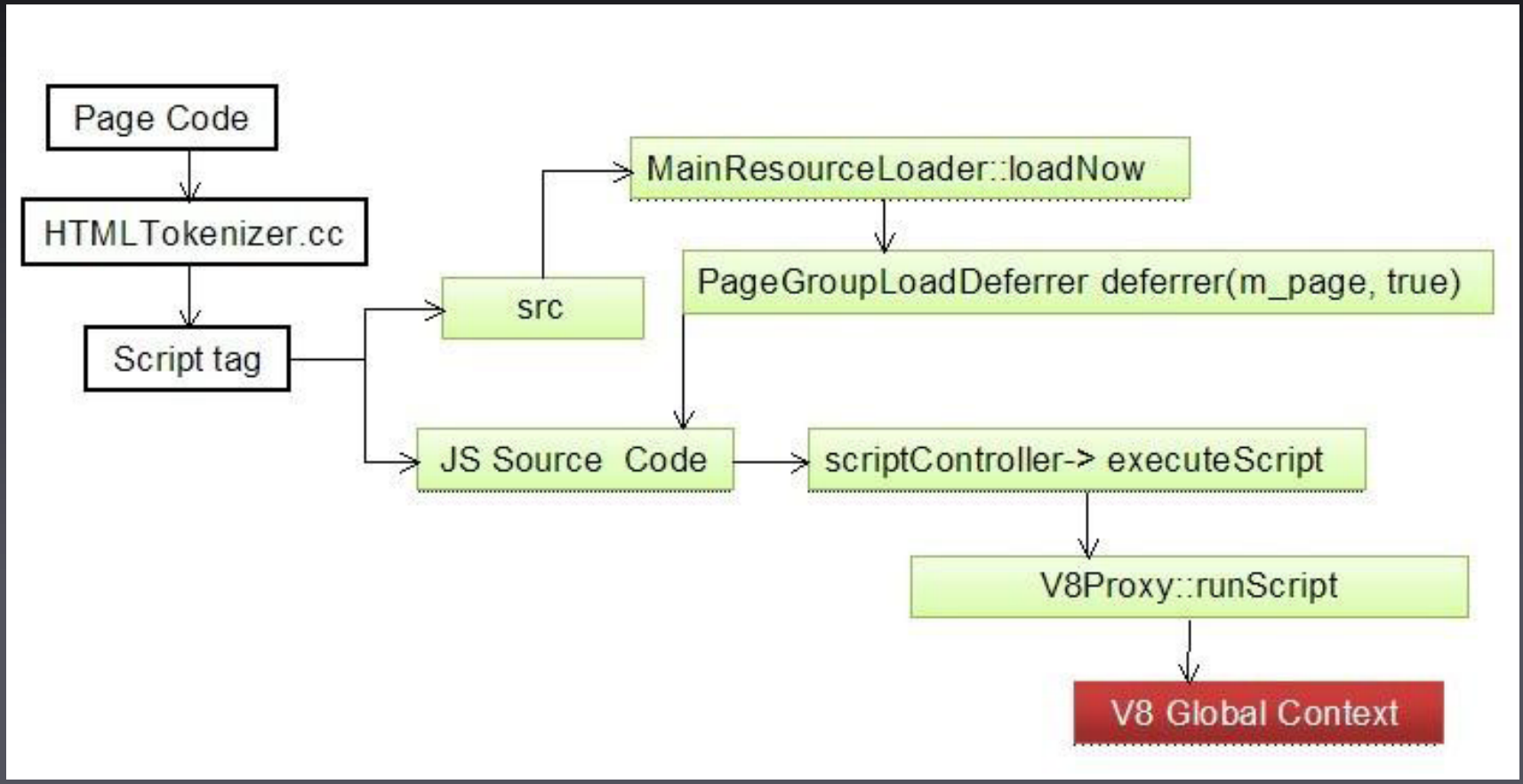
更进一步

- 如果Script脚本是魔鬼，那JS文件就是大魔王！
  - ✓ 需要下载，需要串行（注1）下载！
  - ✓ 需要解析。
  - ✓ 需要执行，需要串行执行！



二、浏览器脚本下载过程

Chromium内核解析流程



JS脚本伤不起

- 在所有下载线程中！
  - ✓ 为什么CSS文件加载不会阻塞页面？
  - ✓ 为什么Images加载不会阻塞页面？
  - ✓ 为什么Flash加载不会阻塞页面？
  - ✓ 为什么ActiveX加载不会阻塞页面？
  - ✓ 为什么Ajax还有同步和异步之分？
  - ✓ 为什么JavaScript文件就会阻塞页面？！

能回避UI阻塞吗？

- Put Scripts at the Bottom
  - ✓ http://developer.yahoo.com/performance/rules.html#js_bottom
  - ✓ 如果脚本的下载+解析+执行的时间太久，UI队列没有得到执行，页面会出现空白
  - ✓ Yahoo！建议将所有的脚本都放在</body>之前，让UI队列优先执行和显示。
- Minimize HTTP Requests
  - ✓ http://developer.yahoo.com/performance/rules.html#num_http
  - ✓ 页面脚本过多的情况下，通过combo和compress减少请求数
- 问题？
  - ✓ 没有真正的回避下载阻塞，在</body>之前存在一个较大的脚本文件需要下载和执行，UI在ready之后，需要较长时间等待脚本的下载和执行，在脚本ready之前，UI是出于无事件响应状态的。

能回避UI阻塞吗？

- Defer属性
  - ✓ HTML4标准为<script>标签定义了defer属性，以此声明告诉浏览器内容中不包含document.write之类破坏DOM的脚本（注2）。
  - ✓ 浏览器会延迟（无阻塞）下载脚本，并按<script>脚本顺序串行执行。
  - ✓ 在HTML流渲染完毕之后，onload事件触发之前执行。
- 实现/支持情况
  - ✓ IE4.0
  - ✓ Firefox3.5

能回避UI阻塞吗？

- Async属性
  - ✓ HTML5标准为<script>标签定义了async属性。
  - ✓ 与defer属性相同的是脚本会无阻塞加载。
  - ✓ 与defer属性不同的是脚本在加载完了立即执行。
  - ✓ 不保证按照<script>标签顺序执行。
- 实现/支持情况
  - ✓ Firefox3.6
  - ✓ Opera10.5、 Safari
  - ✓ Chrome
  - ✓ IE9.0

能回避UI阻塞吗？

- 不依赖浏览器的解决方案
  - ✓ Dynamic Script DOM
  - ✓ XHR Inject
  - ✓ XHR Eval
  - ✓ Script in Iframe

Google Analytics代码

```
<script type="text/javascript">

  var _gaq = _gaq || [];
  _gaq.push(['_setAccount', 'UA-XXXXX-X']);
  _gaq.push(['_trackPageview']);

  (function() {
    var ga = document.createElement('script'); ga.type = 'text/javascript'; ga.async = true;
    ga.src = ('https:' == document.location.protocol ? 'https://ssl' : 'http://www') + '.google-analytics.com/ga.js';
    var s = document.getElementsByTagName('script')[0]; s.parentNode.insertBefore(ga, s);
  })();

</script>
```

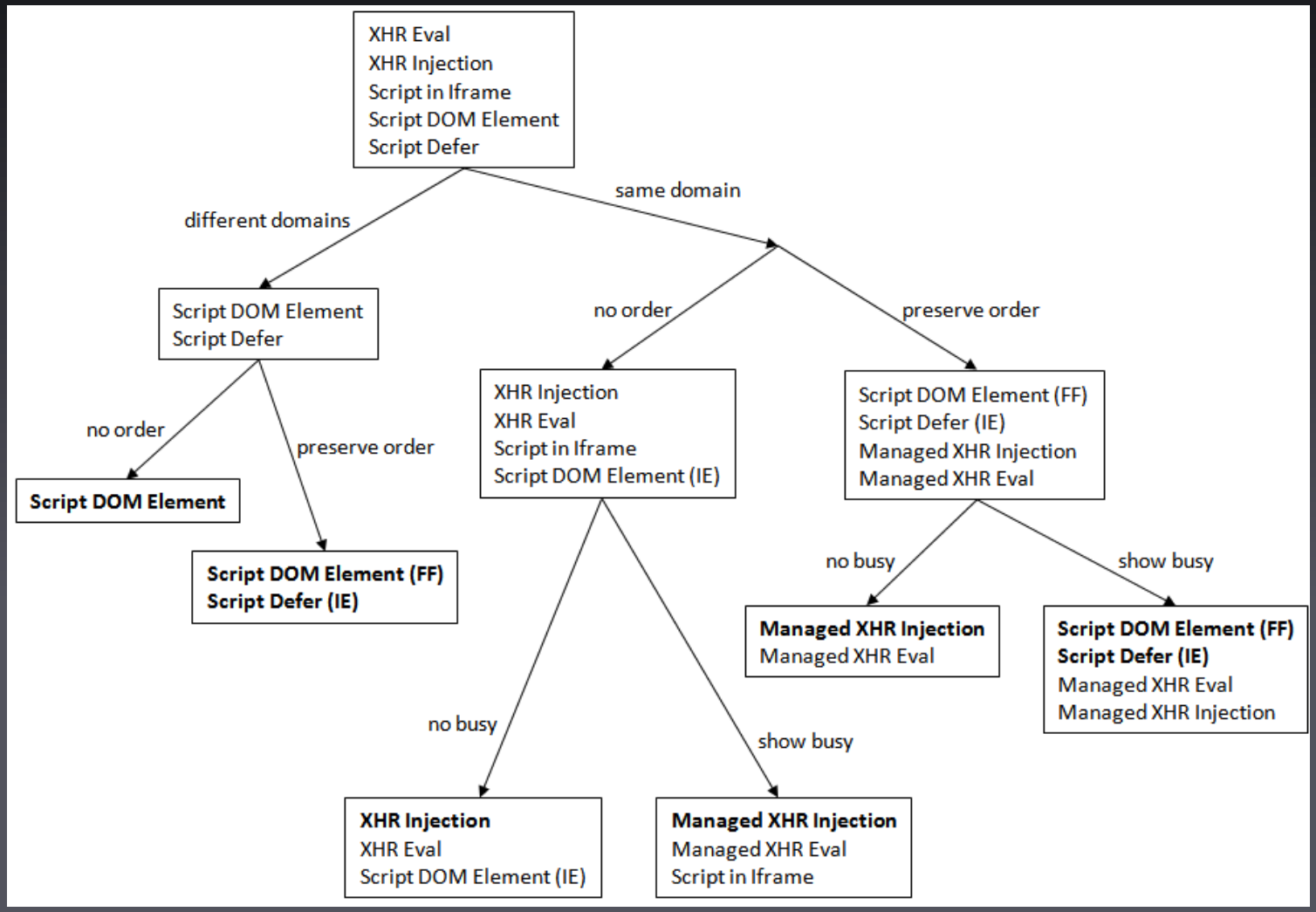
能回避UI阻塞吗？

- 还有问题
  - ✓ 并行、异步下载JS脚本也需要保证顺序、同源策略、CDN、缓存等因素的影响。
    - 没有完美的方案，但是上述每一个问题都可以解决。
  - ✓ 没有一种所有浏览器通用的解决方案。
    - 有，使用LABjs
    - 更好的方式，服务端Combo脚本
    - 最好的方式，以上两种混用。很多大型系统，如远光的ECP平台，IBM的RTC都使用这种方式

LABjs的执行过程

- 对于Firefox/Opera，采用动态Script DOM element可以完美地实现并行下载和顺序执行。
- 对于Safari/Chrome，无法保证顺序执行，但是LABjs通过插入一个<script type=“text/cache” src=“#”>来实现。IE/Safari/Chrome浏览器会下载文件到缓存并触发onload事件，但不会执行。
- 在需要的文件下载完成之后，再次通过插入正确的。type=“text/javascript”而且监听onload来完成对于顺序执行的控制
- LABjs判断script文件是否同域文件，优先选择XHR Injection实现并行下载，顺序执行。
- Text/cache严重依赖浏览器的非标准特性。

LABjs的执行过程


XHR Eval
XHR Injection
Script in Iframe
Script DOM Element
Script Defer
different domains
same domain
Script DOM Element
Script Defer
no order
preserve order
XHR Injection
XHR Eval
Script in Iframe
Script DOM Element (IE)
Script DOM Element (FF)
Script Defer (IE)
Managed XHR Injection
Managed XHR Eval
no order
preserve order
Script DOM Element
Script DOM Element (FF)
Script Defer (IE)
no busy
show busy
Managed XHR Injection
Managed XHR Eval
Script DOM Element (FF)
Script Defer (IE)
Managed XHR Eval
Managed XHR Injection
no busy
show busy
XHR Injection
XHR Eval
Script DOM Element (IE)
Managed XHR Injection
Managed XHR Eval
Script in Iframe

能回避UI阻塞吗？

- 还有问题
  - ✓ 并行、异步下载JS脚本也需要保证顺序、同源策略、CDN、缓存等因素的影响。
    - 上述每一个问题都可以解决
  - ✓ 没有一种所有浏览器通用的解决方案。
    - 有，使用LABjs
    - 更好的方式，服务端Combo脚本
    - 最好的方式，以上两种混用。很多大型系统，如远光的ECP平台，IBM的RTC都使用这种方式

ECP平台的Combo优化

```
var services = [
    "js/jquery/jquery.json.min.js",                          // 第三方包：jQuery JSON Library v2.2
    "js/easyui/jquery.easyui.min.js",                        // 第三方包：jQuery EasyUI 1.2.3
    "js/datepicker/jquery.ui.core.js",                        // 第三方包：jQueryUI core
    "js/datepicker/jquery.ui.datepicker.js",                  // 第三方包：jQueryUI datepicker
    "js/jqgrid/grid.locale-cn.js",                           // 第三方包：jqGrid 国际化文件
    "js/jqgrid/jquery.jqGrid.src.js",                        // 第三方包：jQuery jqGrid
    "js/jquery/jquery.datalink.js",                          // 第三方包：jQuery Data Link plugin v1
    "js/dialog/dialog.src.js",                               // 第三方包：JQuery Zebra Dialog 1.1
    "js/autocomplete/jquery.autocomplete.js",                // 第三方包：jQuery Autocomplete plugin
    "js/com.ygsoft.grace.utilities.misc.js",                 // 平台工具
    "js/com.ygsoft.grace.services.performance.js",           // 平台服务：性能管理服务
    "js/com.ygsoft.grace.services.cache.js",                 // 平台服务：缓存服务
    "js/com.ygsoft.grace.services.remoteobject.js",          // 平台服务：远程访问服务
      "js/com.ygsoft.grace.services.baseconfig.js",            // 平台服务：基础信息服务
    "js/com.ygsoft.grace.services.diagnostic.js",            // 平台服务：诊断及日志服务
    "js/com.ygsoft.grace.components.action.js",               // 平台控件：组件行为
    "js/com.ygsoft.grace.model.filter.js",                  // 平台控件：过滤
    "js/com.ygsoft.grace.model.exportExcel.js",                  // 平台控件：导出EXCEL
    "js/com.ygsoft.grace.model.controlConfig.js",            // 平台控件：控件全局默认配置模型
    "js/com.ygsoft.grace.model.metadata.js",                 // 平台控件：元数据模型
    "js/com.ygsoft.grace.model.data.js",                     // 平台控件：数据模型
    "js/com.ygsoft.grace.model.display.js",                  // 平台控件：ID名称转换模型
    "js/com.ygsoft.grace.components.event.js",              // 平台控件：graceUI事件机制
    "js/com.ygsoft.grace.components.graceUI.js",            // 平台控件：渲染组件
    "js/graceUI/com.ygsoft.grace.components.fltx.js",                // 平台控件：注册分类体控件
    "js/graceUI/com.ygsoft.grace.components.sjxm.js",                // 平台控件：注册数据项目控件
    "js/graceUI/com.ygsoft.grace.components.baseUI.js",              // 平台控件：注册基础控件
    "js/graceUI/com.ygsoft.grace.components.combobox.js",            // 平台控件：注册combobox控件
    "js/graceUI/com.ygsoft.grace.components.datebox.js",             // 平台控件：注册datebox控件
    "js/graceUI/com.ygsoft.grace.components.dxSelect.js",            // 平台控件：注册dxSelect控件
    "js/graceUI/com.ygsoft.grace.components.dxTree.js",              // 平台控件：注册dxTree控件
    "js/graceUI/com.ygsoft.grace.components.jqGrid.js",              // 平台控件：注册jqGrid控件
    "js/graceUI/com.ygsoft.grace.components.queryPanel.js",          // 平台控件：注册查询面板控件
    "js/graceUI/com.ygsoft.grace.components.exportExcel.js",          // 平台控件：注册导出excel按钮
    "js/graceUI/com.ygsoft.grace.components.tree.js",               // 平台控件：注册树控件
    "js/graceUI/com.ygsoft.grace.components.uniSelector.js",          // 平台控件：注册统一选择器控件
    "js/com.ygsoft.grace.components.window.js",              // 平台控件：组件渲染
    "js/com.ygsoft.grace.components.render.js",              // 平台控件：组件渲染（这句加载完成后，页
    "js/com.ygsoft.grace.components.jqGridAddon.js",         // 平台工具：对jqGrid表格的扩展功能
    "js/com.ygsoft.grace.services.messagequeue.js"           // 平台服务：消息队列
];
```

+ 0.144		0.067	2649	167920	GET	200	http://localhost:9080/grace/CompressorServlet?fileType=js&hashCode=626791390

ECP平台的Compress优化

```
/**↓
* 远光Gris平台3.0版本 http://www.ygsoft.com/↓
* Grace [Gris Application Cloud-Computing Extension]↓
* ↓
* 平台性能管理服务↓
* 高性能与高可调试性是互相矛盾的，平台由性能管理器统一确定各个细节的运作模式，以在性能与调试之间获得平衡。↓
* 性能管理器内置了product、debug、local三种性能配置，可以在页面加载时通过参数进行修改，这三种配置见代码注释。↓
*↓
* 使用方式：↓
*  转换性能模式：example.html?mode=debug↓
*  获取当前性能模式：$.getPerformance().scriptCompact↓
*↓
* 变更版本：↓
* zhouzhiming@ygsoft.com 2011-8-9 创建↓
*/↓
(function ($) {↓
    var localMode = location.protocol.substr(0, 4) == "http" ? false : true;↓
↓
    // 性能对象↓
    function ProformanceOptions(options
        // 检查协议，如果是http[s]://url
        this["localMode"] = localMode↓
        if (options) {↓
            for (var item in options) {
                this[item] = options[it
            }↓
        }↓
```

```
(function($){var localMode=location.protocol.substr(0,4)=="http"?false:true;function
ProformanceOptions(options){this["localMode"]=localMode;if(options){for(var item in
options){this[item]=options[item];}}}ProformanceOptions.prototype={constructor:ProformanceOptions,scriptC
ompact:true,scriptInline:true,scriptCache:true,scriptAsynLoading:true,modelCache:true,breakPoint:false,fi
reBugLite:false,localMode:false,gzipEnable:true};var repository={product:new
ProformanceOptions(),debug:new
ProformanceOptions({scriptCompact:false,scriptAsynLoading:false,scriptInline:false,scriptCache:false,brea
kPoint:true,fireBugLite:true}),local:new
ProformanceOptions({scriptCompact:false,scriptInline:false,scriptCache:false,breakPoint:true,fireBugLite:
false,localMode:true})};var debugMode=$.getUtils().getArgment("mode");if(debugMode){var
args=debugMode.split("|");var mode=args[0].toLocaleLowerCase();var
current=repository[mode]?repository[mode]:repository[localMode?"local":"product"];if(args.length>1){try{v
ar adjectObj=eval("("+args[1]+")");$.extend(current,adjectObj);}catch(e){}}}else{var
current=repository[localMode?"local":"product"];}$.extend({getPerformance:function(){return
current;}});})(jQuery);←
```

能回避UI阻塞吗？

- 还有问题
  - ✓ JS脚本下载可以比较完美的解决，但更大的挑战还在后面。
- 执行阻塞
  - ✓ 只能解决脚本下载阻塞UI，不能解决脚本执行阻塞UI的问题！
  - ✓ JavaScript向异步进军！

三、向异步进军

异步，先看一些观点

- 认为“异步编程 = 方法 + 回调函数”。
- 认为“浏览器是单线程执行的”。
- 使用过setTimeout或者setInterval模拟多线程。
- 试过用JavaScript实现sleep()方法。

异步，先看一些观点

- 认为“异步编程 = 方法 + 回调函数”。
- 认为“浏览器是单线程执行的”。
- 使用过setTimeout或者setInterval模拟多线程。
- 试过用JavaScript实现sleep()方法。

异步例子：冒泡排序

```
var compare = function (x, y) {
  return x - y;
}
var swap = function (a, i, j) {
  var t = a[i]; a[i] = a[j]; a[j] = t;
}
var bubbleSort = function (array) {
  for (var x = 0; x < array.length; x++) {
    for (var y = 0; y < array.length - x; y++)
    {
      if (compare(array[y], array[y + 1]) > 0)
    {
        swap(array, y, y + 1);
      }
    }
  }
}
```

需求变更

为了让用户能看清楚排序过程，

请修改为每隔100ms进行一次排序交换。

异步例子：冒泡排序

```
var compare = function (x, y, callback) {
  setTimeout(10, function () {
    callback(x - y);
  });
}

var swap = function (a, i, j, callback) {
  var t = a[i]; a[i] = a[j]; a[j] = t;
  repaint(a);

  setTimeout(20, callback);
}

var outerLoop = function (array, x,
    callback) {
  if (x < array) {
    innerLoop(array, x, 0, function () {
      outerLoop(array, x + 1, callback);
    });
  } else {
  callback();
  }
}
```

```
var innerLoop = function (array, x, y, callback) {
  if (y < array.length - x) {
    compare(array[y], array[y + 1], function (r) {
      if (r > 0) {
        swap(array, y, y + 1, function () {
          innerLoop(array, x, y + 1, callback);
        });
      } else {
        innerLoop(array, x, y + 1, callback);
      }
    });
  } else {
    callback();
  }
}

outerLoop(array, 0, function () {
  console.log("done!");
}
);
```

异步例子：冒泡排序

```
var compare = function (x, y, callback) {
  setTimeout(10, function () {
    callback(x - y);
  });
}

var swap = function (a, i, j, callback) {
  var t = a[i]; a[i] = a[j]; a[j] = t;
  repaint(a);

  setTimeout(20, callback);
}

var outerLoop = function (array, x,
    callback) {
  if (x < array) {
    innerLoop(array, x, 0, function () {
      outerLoop(array, x + 1, callback);
    });
  } else {
  callback();
  }
}
```

```
var innerLoop = function (array, x, y, callback) {
  if (y < array.length - x) {
    compare(array[y], array[y + 1], function (r) {
      if (r > 0)
        swap(array, y, y + 1, function () {
          innerLoop(array, x, y + 1, callback);
        });
      } else {
        innerLoop(array, x, y + 1, callback);
      }
    });
  } else {
    callback();
  }
}

outerLoop(array, 0, function () {
  console.log("done!");
}
);
```

异步例子：冒泡排序

```
var compare = function (x, y) {
  return x - y;
}
var swap = function (a, i, j) {
  Thread.sleep(100);    // 我就是想要Java语言的这句话而已……
  var t = a[i]; a[i] = a[j]; a[j] = t;
}
var bubbleSort = function (array) {
  for (var x = 0; x < array.length; x++) {
    for (var y = 0; y < array.length - x; y++) {
      if (compare(array[y], array[y + 1]) > 0) {
        swap(array, y, y + 1);
      }
    }
  }}
```

JSCEX解决方案

提出问题：异步编程有难度

- 破坏代码局部性
  - ✓ 程序员习惯线性地表达算法。
  - ✓ 异步代码将逻辑拆分地支离破碎。
- 难以
  - ✓ 应用于需要保持顺序的场景。
  - ✓ 异步操作之间的协作及组合。
  - ✓ 处理异常及取消。

分析问题：问题的根源

- JavaScript是单线程的编程语言。
  - ✓ 不能创建线程，开展并行任务。
  - ✓ 不能对（当前）线程操作 。
- 没有多线程就算了，还动不动阻塞掉UI渲染。
- 设计JavaScript执行阻塞UI渲染的人上辈子是折翼的天使，伤不起。

解决问题

- 抱怨JavaScript没有多线程的程序员不是好程序员。
- JavaScript是单线程 ≠ 浏览器是单线程
  - ✓ JavaScript执行线程
  - ✓ UI渲染线程
  - ✓ 资源下载线程（JavaScript, CSS, Image, Object）
  - ✓ Ajax线程
  - ✓ Web Worker线程
  - ✓ 使用setTimeout“模拟多线程”
- 没有一揽子方案，根据具体场景来解决问题。

解决问题

- 抱怨JavaScript没有多线程的程序员不是好程序员。
- JavaScript是单线程 ≠ 浏览器是单线程
  - ✓ JavaScript执行线程
  - ✓ UI渲染线程
  - ✓ 资源下载线程（JavaScript, CSS, Image, Object）
  - ✓ Ajax线程
  - ✓ Web Worker线程
  - ✓ 使用setTimeout“模拟多线程”

可以利用的异步线程

- 没有一揽子方案，根据具体场景来解决问题。

使用异步典型场景举例之一

请求数据

三种主流的请求方式

- Beacons（信标）
- XMLHttpRequest（XHR）
- Dynamic script tag insertion 动态脚本注入
  （JSON-P，JSON with Padding）

Beacons

- 最古老的Ajax技术（在IE3之前就出现了），使用资源下载线程实现异步的代表。
- 原理：使用JS创建一个新Image对象，并把src设置为服务器脚本的URL，然后监听image的load事件来获知服务器响应。
- 优劣势：
  - ✓ 优点：异步，简单、性能消耗小，不受同源策略影响。
  - ✓ 缺点：强制异步，URL长度受限，且无法发送POST数据，只能返回有限的状态，无法返回数据。

Beacons

```
var url= '/BeaconsServlet';
var params= ['step=2', 'time=1238027314'];
var beacon = new Image();
beacon.src = url+ '?'+ params.join('&');

//使用信标处理服务器返回状态
beacon.onload= function() {
  if(this.width== 1){
    //成功
  }  else if(this.width== 2){
    //失败
  }
}
```

this.width是图片宽度
使用图片的高宽来返回数据

XMLHttpRequest

- IE4出现的XHR请求是最常见的Ajax技术，XHR对象现代Ajax和Comet技术（也称为反向Ajax）的基础。
- XHR可以指明使用哪种HTTP Method
  - ✓ GET：没有HTTP Body，参数直接在URL中，数据包长度小，HTTP协议没有明确限制GET方法参数长度，但是浏览器有限制（exp：IE为2083字节）
  - ✓ POST：有HTTP Body，参数在Body中，能传输大数据量，能享受GZIP的好处，（相对）安全一些。

XMLHttpRequest

- XHR可以监听readyState状态
  - ✓ readyState== 0 //尚未加载
  - ✓ readyState== 1 //正在加载
  - ✓ readyState== 2 //加载完毕
  - ✓ readyState== 3 //正在处理
  - ✓ readyState== 4 //处理完毕

有这种状态，就可以使用Stream的方式与服务端交互，由此产生了Comet技术。

XMLHttpRequest

- XHR可以建立异步线程来请求资源。
  - ✓ open方法明确的同步异步选择参数。
  - ✓ （HTML5前）JavaScript中唯一明确使用API创建线程的途径。
  - ✓ 数据请求过程不在JavaScript执行线程中。
  - ✓ 但回调函数的执行过程在JavaScript执行线程中。
- XHR不可以跨域请求数据，受同源策略限制。
  - ✓ GRIS要求用户把地址加入安全站点来绕过这个问题。
  - ✓ 上面那个解决方案是在耍流氓。

JSON-P

- 2005年George Jempty提出的一种不受同源策略限制的Ajax方式，目前在jQuery，Dojo、GWT框架中都有封装。
- 原理：动态生成<Script>对象，将回调函数传递给服务端，服务端在返回数据时加入回调前缀。
- 优劣势：
  - ✓ 除了不能POST数据，能做到XHR其他所有功能，而且不受同源策略限制。
  - ✓ 需要服务端做简单配合；本质上是脚本注入，不应用于安全敏感领域。

JSON-P

```
// 客户端代码：
var JSONP = document.createElement("script") ;
//FF:onload IE:onreadystatechange
JSONP.onload = JSONP.onreadystatechange = function(){
  //onreadystatechange，仅IE
  if (!this.readyState || this.readyState === "loaded" || this.readyState ===
    "complete") {
    // 处理状态变化
  }
}
JSONP.type = "text/javascript";
JSONP.src = "http://diff_domain/JSONPServlet?callback=parseResponse";
document.getElementsByTagName("head")[0].appendChild(JSONP);

function parseResponse(data){
  // 处理返回请求
}
```

JSON-P

- 服务端支持：

{"Name": "小明", "Id" : 1823, "Rank": 7}

→

parseResponse({"Name": "小明", "Id" : 1823, "Rank": 7})

使用异步典型场景举例之二

执行队列让步

典型场景

- 需要定时或者延后触发的场景
  - ✓ 闹钟、时钟之类的应用
- 需要对线程进行Sleep的场景
  - ✓ 前面那个例子（目前没有完美的解决方案）
- 需要JavaScript执行线程让步给UI线程的场景
  - ✓ 分时函数
- 还有很多，欢迎补充……

分时函数

- 解决JavaScript线程阻塞UI线程的问题。
- 分时函数会降低效率（增加UI Update次数），但能从感觉上提升用户体验。
- 让JavaScript和UI线程“看起来”是并发执行的。

分时函数

想象中是这样的：

```
Js Code
$
('#ID').innerHTML='sth'
```

浏览器render操作

```
Js Code
$
('className').css('widt
h','100px');
```

浏览器render操作

现实中是这样的：

```
$('#ID').setHeight
$('#ID').setWidht
…… ……
$('#ID').setInnerHTML
$('#ID').deleteChild
…… ……
```

浏览器render操作

分时函数

```
// 分时函数实例
function timedChunk(items, process, context, callback) {
  var todo = items.concat(), delay = 25;
  setTimeout(function() {
    var start = +new Date();
    do {
      process.call(context, todo.shift());
    } while (todo.length > 0 && (+new Date() - start < 50))
    if(todo.length > 0) {
      setTimeout(arguments.callee, delay);
    } else if(callback) {
      callback();
    }
  }, delay);
}
```

DEMO示例

另一个例子

- 下面程序的执行结果？

```
var t = true;
setTimeout(function(){ t = false; }, 1000);
while(t){ }
alert('end');
```

另一个例子

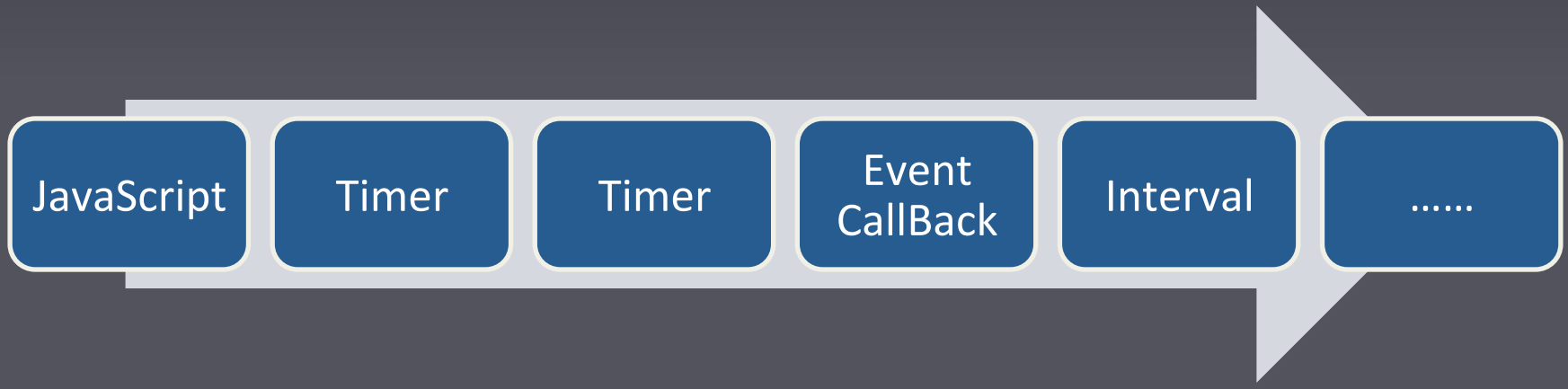
- 下面程序的执行结果？

```
var t = true;
setTimeout(function(){ t = false; }, 1000);
while(t){ }
alert('end');
```

你的浏览器挂了吗？

执行队列

- JavaScript执行引擎在完成当前任务之后，从执行队列中取出下一个任务，直到队列为空时，将执行权限交回给UI线程。
- 定时器、事件等都共用这个队列。
- setTimeout的作用仅是在时间到期后在执行队列的最后插入一个任务。
- 任务只保证时间没到不会触发，不保证时间到了一定触发。



setTimeout缺陷

- 不能保证按时执行。
  - ✓ 只能保证不会提早执行，不能保证不会延后
- 无法精确确定时间分片。
  - ✓ Windows机器默认时钟精度为10-15.6毫秒（大部分情况是15.6毫秒）
- 无法传递参数（注1），易造成全局变量污染。
- 极大的破坏程序结构，比GOTO语句还严重。
- 绕过了异常检查（try-catch）。

使用异步典型场景举例之三

大规模后台运算

典型场景

- HTML5的Web Worker API是真正打破JavaScript执行线程樊笼的武器。
- 可以用于：
  - ✓ 编码/ 解码大字符串（如：解析巨大的JSON）
  - ✓ 复杂数学运算
  - ✓ 大数组排序
  - ✓ 图像或视频处理（基于HTML5的3D游戏）

典型场景

- Fibonacci数列
  - ✓ fibonacci数列被以递归的方法定义：F0=0，F1=1，Fn=F(n-1)+F(n-2)（n>=2，n∈N*）
  - ✓ 传统JavaScript的计算方法：

```
//在Chrome中执行fibonacci(39)耗时8099毫秒
var fibonacci = function(n) {
  return n <2 ? n : arguments.callee(n - 1) +
   arguments.callee(n - 2);
};
```

Web Worker解决方案

```
// 在fibonacci.js文件中
var fibonacci =function(n) {
    return n <2? n : arguments.callee(n -1) + arguments.callee(n -2);
};
onmessage =function(event) {
    var n = parseInt(event.data, 10);
    postMessage(fibonacci(n));
};
// 在HTML文件中
var worker =new Worker('fibonacci.js');
worker.addEventListener('message', function(event) {
  var timer2 = (new Date()).valueOf();
  console.log( '结果：'+event.data, '时间:'+ timer2, '用时：'+ ( timer2  -
    timer ) );
}, false);
```

Web Worker优势

- 可以加载一个JS进行大量的复杂计算而不挂起主进程，并通过postMessage，onMessage进行通信。
- 可以在worker中通过importScripts(url)加载另外的脚本文件。
- ECMAScript对象，如：Object/Array/Date 等
- 可以使用 setTimeout()、clearTimeout()、setInterval() 和 clearInterval()。
- 可以使用XMLHttpRequest来发送请求。
- 可以访问navigator的部分属性。读取（只读）Location对象

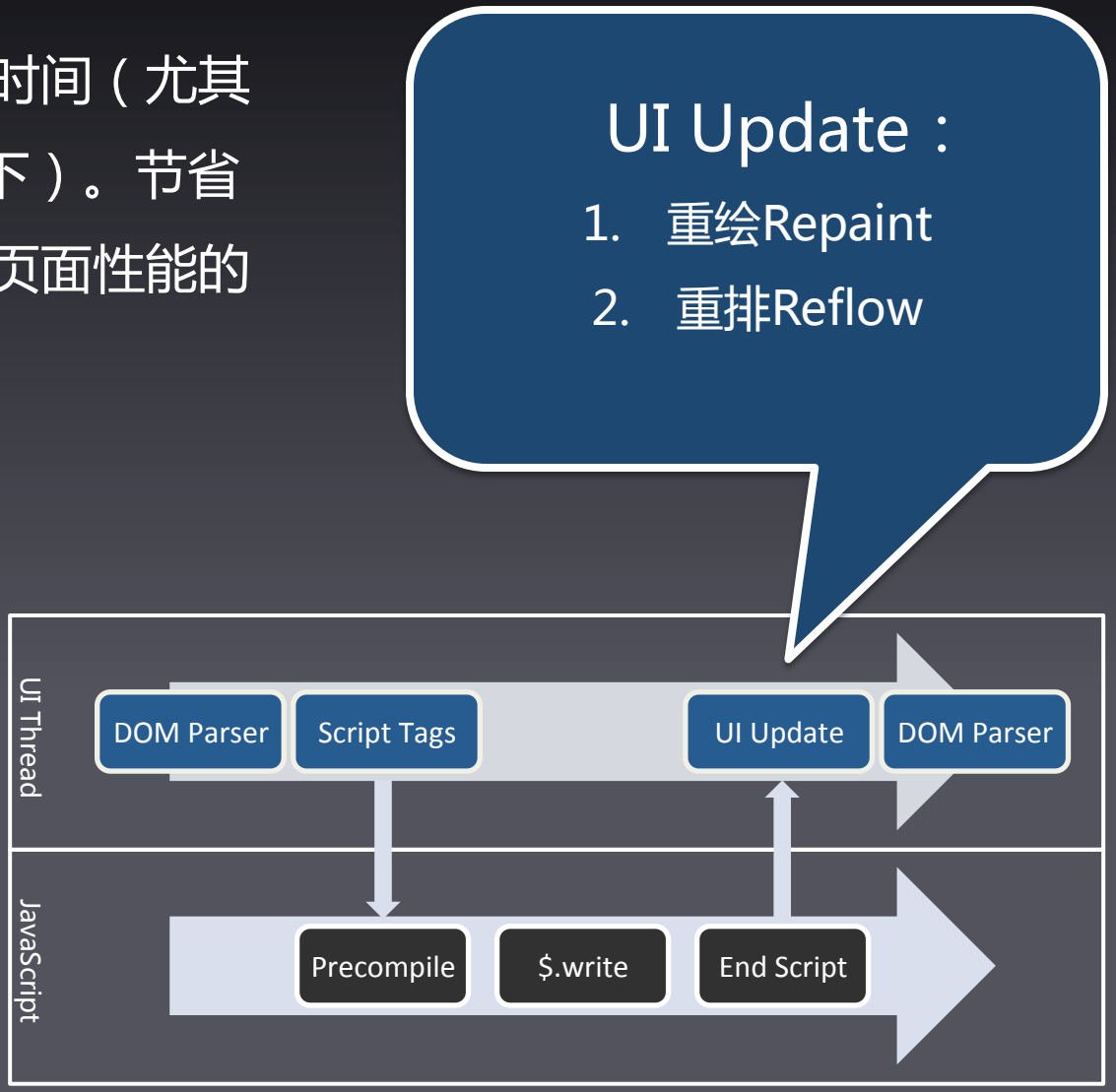
Web Worker限制

- 不能跨域加载JS文件。
- worker内代码不能访问DOM。
- 各个浏览器对Worker的实现不大一致，例如FF里允许worker中创建新的worker,而Chrome中就不行。
- 不是每个浏览器都支持这个新特性。

四、谈谈UI更新

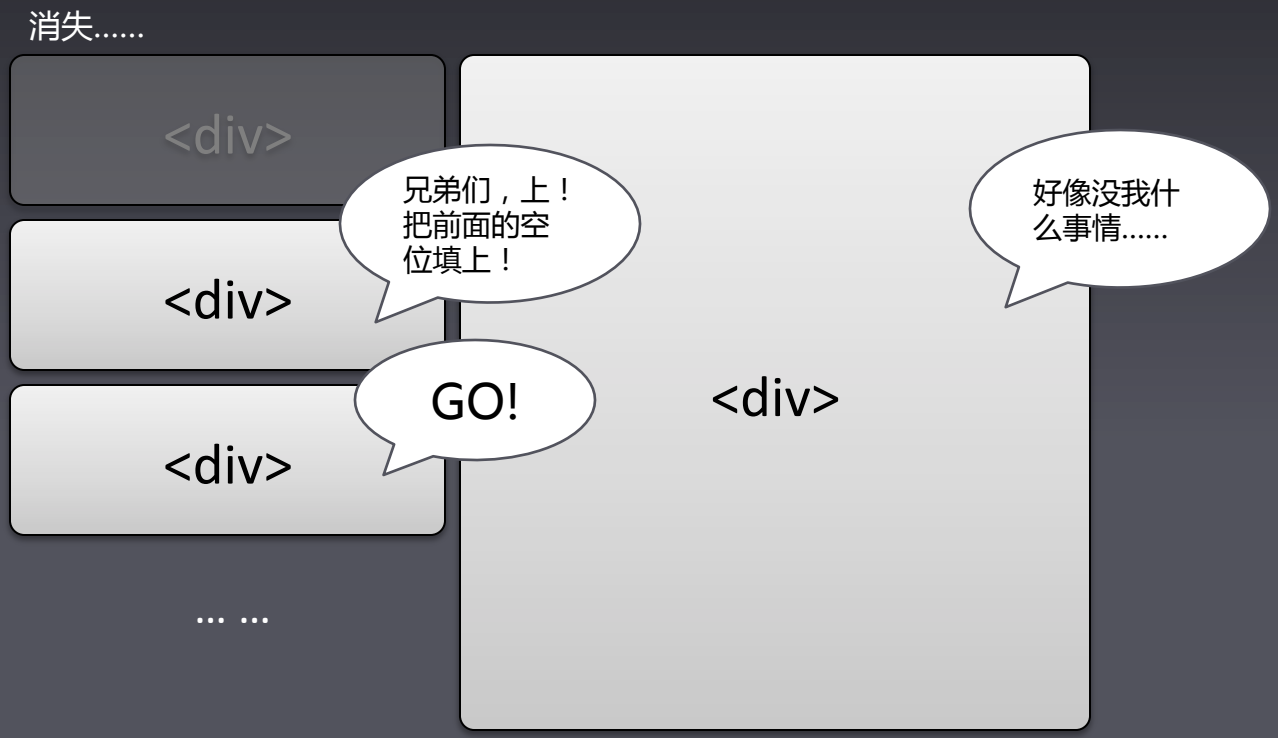
UI更新

UI更新同样需要花费时间（尤其在低版本的IE浏览器下）。节省UI更新次数也是提升页面性能的途径。



重排

重排（Reflow）：由于页面布局发生改变时，对页面元素进行重渲染。



重排

- 重排出现的原因：
  - ✓ 页面渲染过程中
  - ✓ Dom结构变化
  - ✓ 浏览器窗口大小改变
  - ✓ 布局变化

```
<div id="J">
  <script>
    $('J').append('<div>text</div>');
  </script>
</div>
```

重绘

重绘（Repaint）：当页面产生不影响布局的变化时，发生重绘。

重绘

- 重绘出现的原因：
  - ✓ 透明度更改
  - ✓ 文字颜色变化
  - ✓ 背景颜色变化
  - ✓ 背景图片替换

```
<div id="J">
  <script>
    $('J').css('color','red');
  </script>
</div>
```

布局

- 布局方式对渲染和重排有很大影响
- 布局方式
  - ✓ 流式布局
    - display: inline / inline-block / block
    - float: left / right
    - clear: left / right / both
    - position: static / relative / absolute / fixed
  - ✓ 表式布局
    - display: table / inline-table / table-row-group / table-header-group / table-footer-group / table-row / table-column-group / table-column / table-cell / table-caption

布局

- 流式布局 vs 表式布局
  - ✓ 流布局可自左向右、自上而下进行，流中靠后的元素不会影响流中靠前的元素的布局（无回溯）
  - ✓ table布局需要回溯才能够完成（知道每一个单元格的大小，才能完成整个布局）
  - ✓ 反对table布局的原因 – 回溯对渲染的影响

布局

- 不是table才是table布局
  - ✓ body {display: table}
  - ✓ div {display: table-row}
  - ✓ p {display table-cell; border: 1px solid #000;}

References

- 《淘宝前端技术演进》
- 《潜力无限的编程语言 Javascript》
- 《深入浅出Jscex》
- 《Inside-the-Browser》