UC优视

JavaScript引擎工作原理

孙昌青/王永光

2011年11月22日

什么是 JavaScript?

- JavaScript是一种客户端的脚本语言
- JavaScript 是一种解释性语言
- JavaScript直接嵌入 HTML 页面，用来向 HTML 页面添加交互行为
- JavaScript可在所有主要的浏览器中运行

- 内部脚本（**internal script**）

```
<html>
  <head>
     <script type="text/javascript">
     var a;
     a = 123 ;
     a = “Hello World";
     </script>
  </head>

  <body>
     <input onclick="alert('Hello,world!');" type=text />
  </body>
</html>
```

- 外部脚本（**external script**）

```
<html>
  <head>
     <script type="text/javascript“ src=“example_external.js”>
     </script>
  </head>

  <body>
     <script type="text/javascript">
    test()；//调用example_external.js中定义的函数test
     </script>
  </body>
</html>
```

- 注意:外部脚本的域名不需要与主文档域名相同。

JavaScript能做什么？

- **JavaScript** 可以将动态的文本放入 **HTML** 页面

例如:

```
document.write("<h1>" + name + "</h1>")
```

可以将一段可变的文本放入 **HTML** 页面：

- **JavaScript** 可以对事件作出响应

例如:可以将 **JavaScript** 设置为当某事件发生时才会被执行，

```
<input onclick="alert('Hello,world!');" type=text />
```

点击此**input**元素会触发**onclick**事件的执行**alert**函数

- **JavaScript** 可以控制**HTML** 元素

例如:

```
<input id=“aInput” type=text  value=“”/>,
<script type="text/javascript">
document.getElementById("aInput").value = "Input a value"
</script>
```

1. JavaScript概述

- **JavaScript** 可被用来验证数据

在数据被提交到服务器之前，**JavaScript** 可被用来验证这些数据。

```
<form  onsubmit="return chkLogin()" action="....login.php" method="post">
    <input type="text"  value=""   id="username" name="username">
    <input type="passwd"  value="" id="userpasswd" name="userpasswd">
    <input type="submit" value="登录">
</form>
<script type="text/javascript">
    chkLogin : function(){
            var username = document.getElementById("username").value;
            var password = document.getElementById("userpasswd").value;
            var loginType = document.showLogin.entry.value;
            if(username.length < 1 || username.length > 64){
                    alert("用户名非法，请检查!");
                    return false;
            }
            if(password.length < 1 || password.length > 30){
                    alert("密码非法，请检查!");
                    return false;
            }
            return true;
    }
</script>
```

1. JavaScript概述

- **JavaScript** 可被用来创建 **cookie**

JavaScript 可被用来存储和取回位于访问者的计算机中的信息。

```
document.cookie="usrname"+ "=" +escape("Lincon")//创建一个cookie数据
```

- **JavaScript** 可被用来检测访问者的浏览器

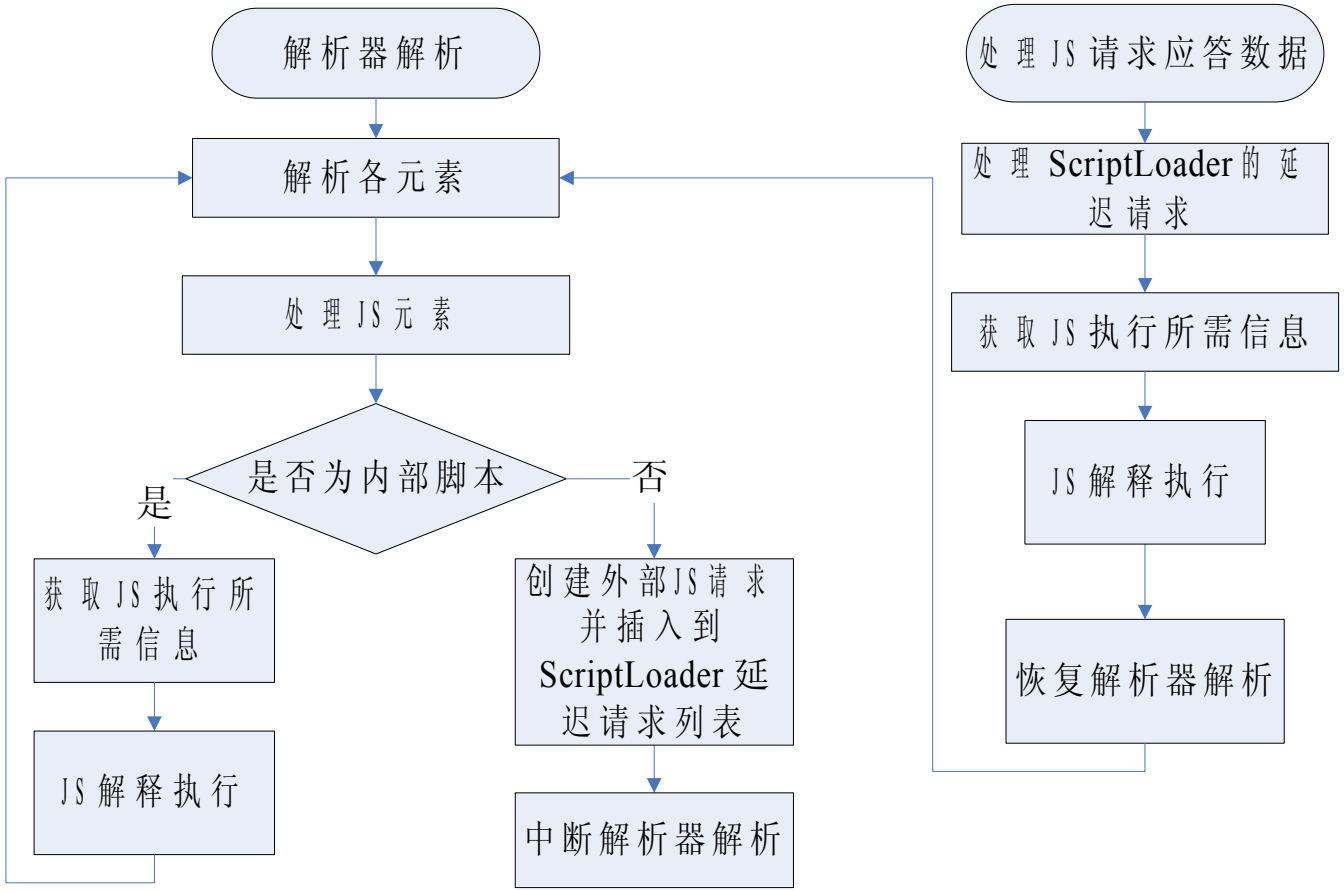
JavaScript 可被用来检测访问者的浏览器，并根据所检测到的浏览器，为这个浏览器载入相应的页面。

```
var ua = navigator.userAgent.toLowerCase(); //获取用户端信息
var info = {
  ie: /msie/.test(ua) && !/opera/.test(ua),   //匹配IE浏览器
  op: /opera/.test(ua),  //匹配Opera浏览器
  sa: /version.*safari/.test(ua),  //匹配Safari浏览器
  ch: /chrome/.test(ua),  //匹配Chrome浏览器
  ff: /gecko/.test(ua) && !/webkit/.test(ua)  //匹配Firefox浏览器
};
if (info.ie)
        window.location.href = ".....ie.html";
(info.op)
        window.location.href = ".....op.html";
(info.sa)
        window.location.href = ".....sa.html";
(info.ff)
        window.location.href = ".....ff.html";
(info.ch)
        window.location.href = ".....ch.html";
```

2. 中间件相关的JS处理

- 处理页面中的**JS**元素
- 处理元素的**JS**事件
- 定制模式的**JS**脚本

2.1处理页面中的JS元素



Firefox处理JS元素的流程

2.2处理元素的JS事件

由于目前web页面的加载与数据处理都是在中间件服务器端进行，手机客户端如果要执行元素的JS事件则必须再次请求中间件服务器。

主要流程：

1.中间件在进行页面数据转换输出时，会为每个元素分派一个唯一的id 并指明其具有的事件类型

2.用户在客户端点击某具有JS事件的元素时，触发对中间件的请求并将元素的id和事件类型发送给中间件

3.中间件根据收到的客户端的请求类型判断出这是一个客户端事件，然后根据元素的id找到其在DOM中对应的实际页面元素并在内核中执行它的JS事件。

2.3定制模式的JS脚本

在页面加载结束后，通过额外加载中间件自定义的本地JS脚本来达到对页面结构进行调整的目的。

具体应用：论坛模式

3. JS引擎的基本原理

UC优视

- 常见的**JavaScript**引擎

- V8，C/C++，Chrome
- Carakan，C/C++，Opera 10+，Link
- Linear A/B，Opera 4.0～9.2
- Futhark，Opera 9.5～10.2
- SquirrelFish（Nitro），C++，Safari 4，Link
- SquirrelFish Extreme，C++，Link
- Simple ECMAScript Engine，C
- SpiderMonkey，C/C++，Firefox 1.0～3.0，Link
- TraceMonkey，C++，Firefox 3.5～3.6，Link
- JaegerMonkey，C/C++，Firefox 4.0+，Link
- Chakra，IE9

3. JS引擎的基本原理

- JavaScript引擎中的基本概念：

◆ Parser：解析器，负责词法分析和语法分析。

◆ Bytecode :满足某种指令集架构的指令序列

◆ Bytecompiler：字节码编译器，用于根据语法树生成字节码。

◆ Assembler：封装了目标机器的汇编代码

◆ JIT：实时编译器，用于将中间字节码编译成机器代码

◆ Interpreter：JavaScript解释器，本身实现了虚拟机功能，同时通过调用Parser、 Bytecompiler 、 JIT各个模块的接口，以实现JavaScript解释器的功能

3. JS引擎的基本原理

- “编译型语言”的执行过程：

第1步：

第2步：

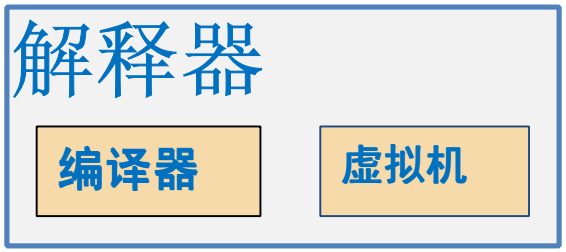
- “解释型语言”的执行过程：

只有一步：

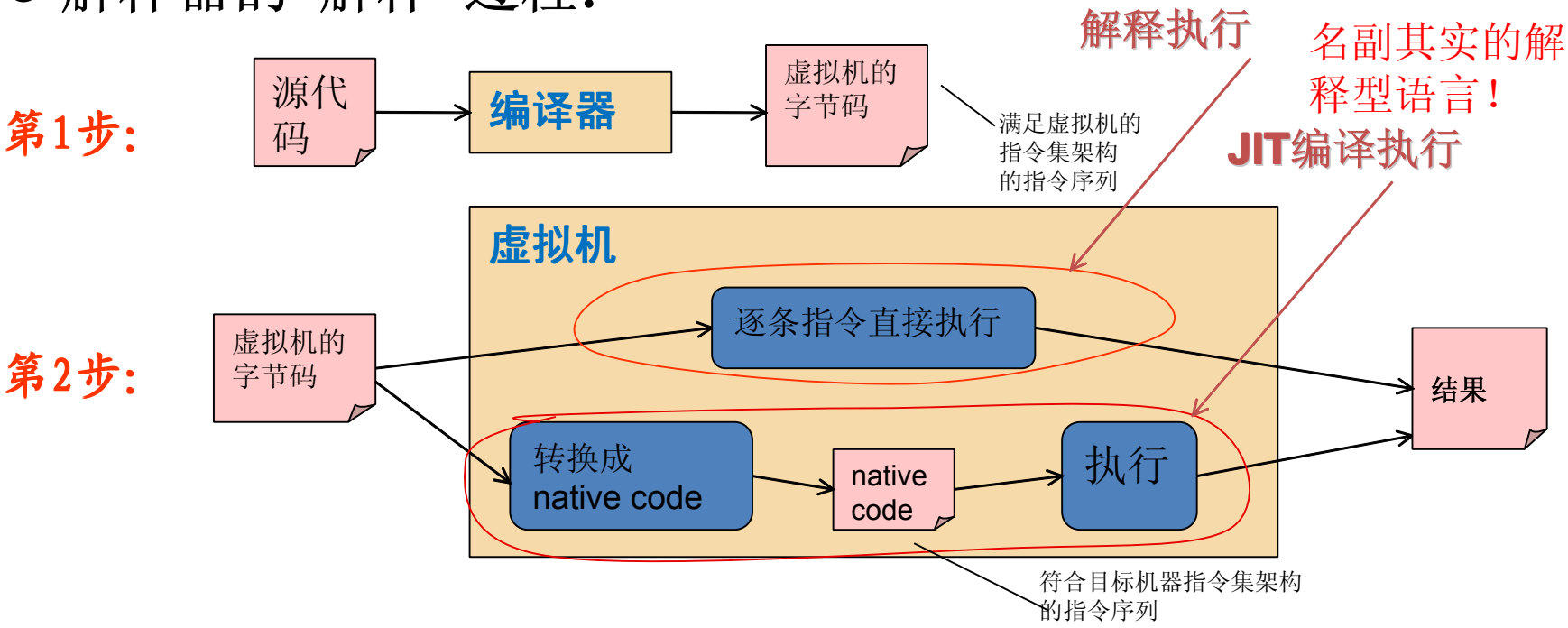
3. JS引擎的基本原理

- 解释器需要“编译”？

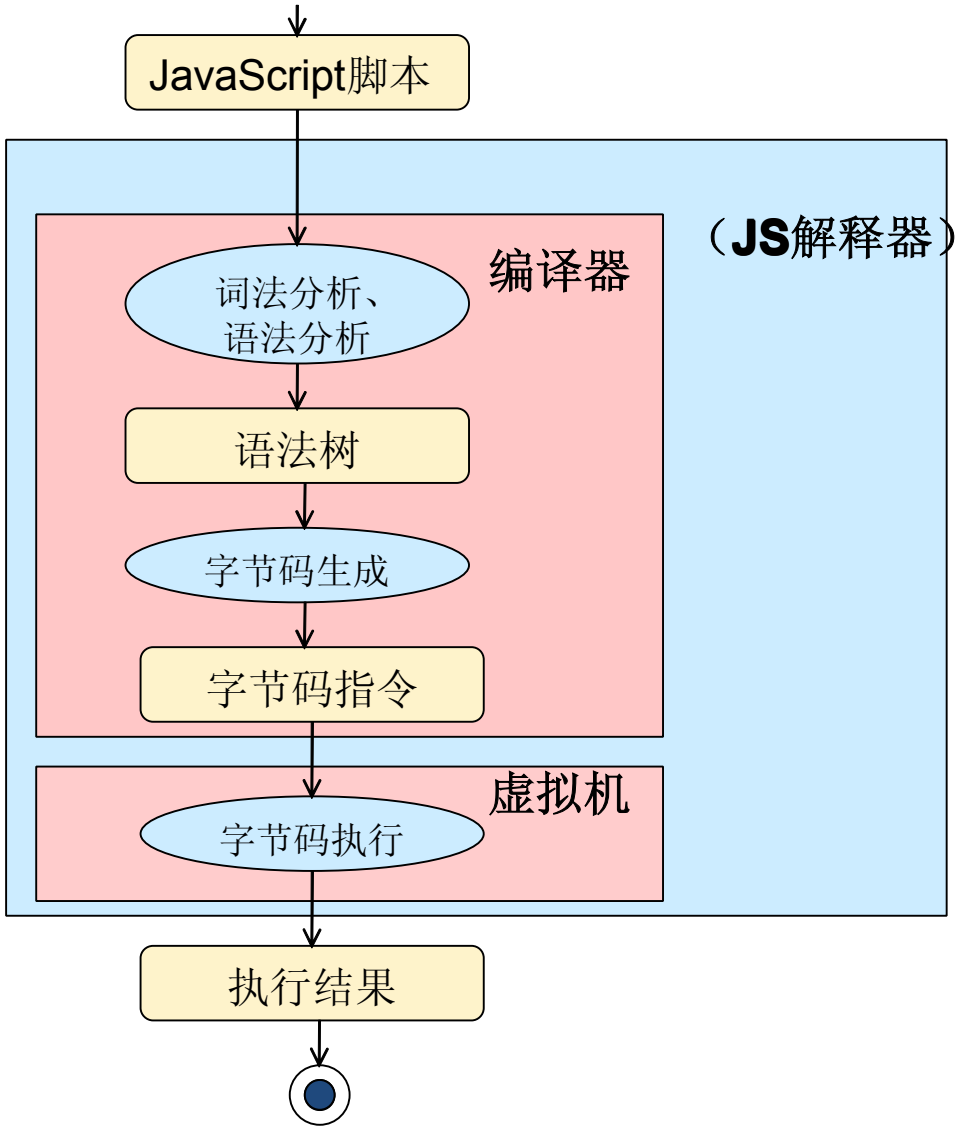
解释器 = 编译器 + 虚拟机



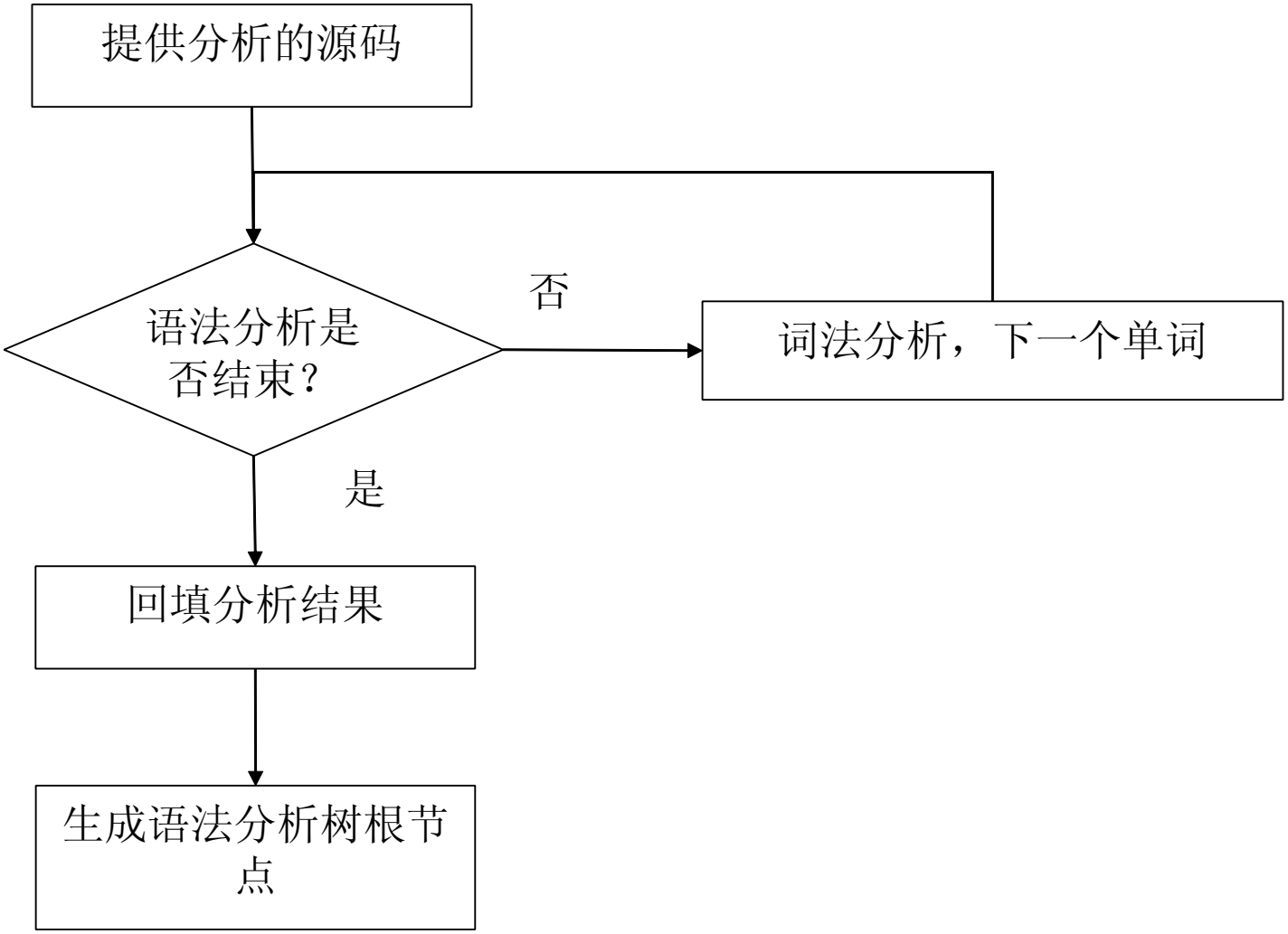
- 解释器的“解释”过程：



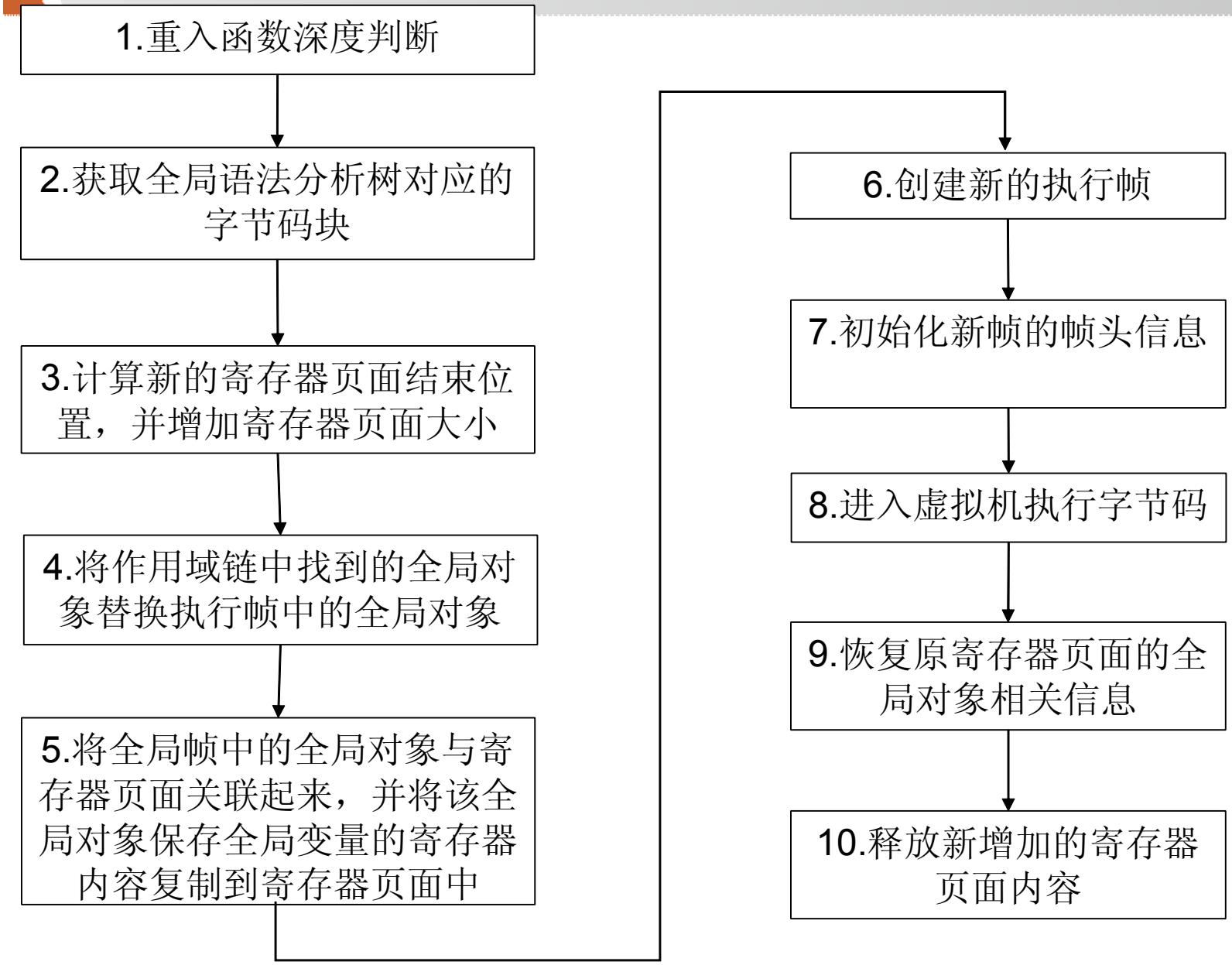
3.1JS脚本的执行过程



3.2JS引擎的语法解析过程



3. 3JS引擎字节码的执行流程



3. 3JS引擎字节码的执行流程

●JavaScript脚本：

```
var a = 1;

if (a > 0)

   alert(a);
```

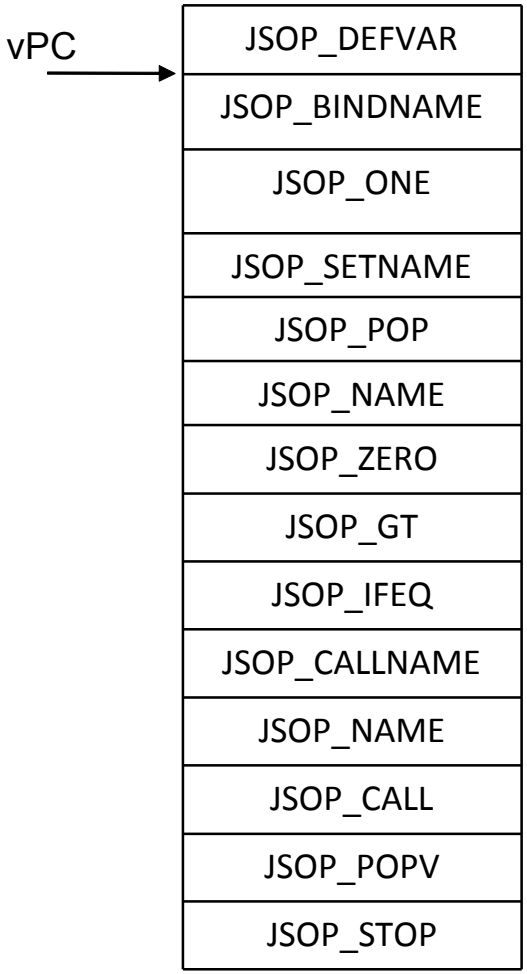
编译后的 字节码

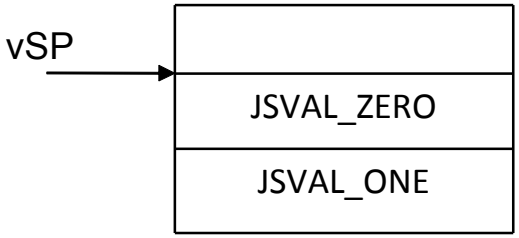
JSOP_DEFVAR
JSOP_BINDNAME
JSOP_ONE
JSOP_SETNAME
JSOP_POP
JSOP_NAME
JSOP_ZERO
JSOP_GT
JSOP_IFEQ
JSOP_CALLNAME
JSOP_NAME
JSOP_CALL
JSOP_POPV
JSOP_STOP

3. 3JS引擎字节码的执行流程

虚拟机执行字节码的过程：

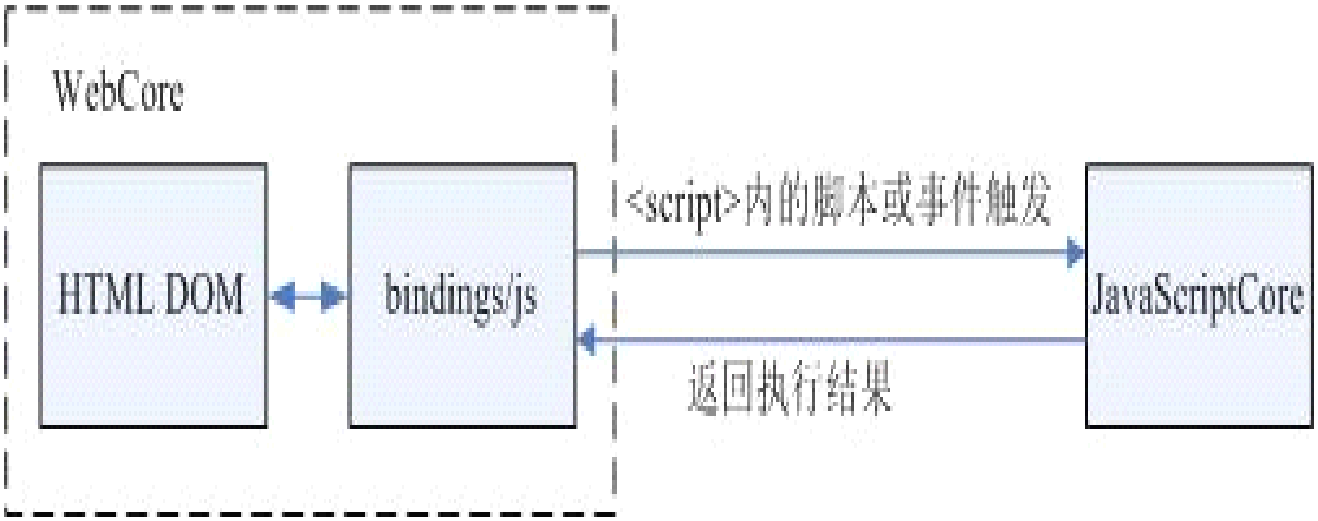
```
while (1)
{
    ……
    switch (vPC->u.opcode)
    {
      ……
      case ： JSOP_DEFVAR
      {
          ……
          goto nextOPcode;
      }
      ……
      case ： JSOP_BINDNAME
      {
        ……
        goto nextOPcode;
      }
      ……
      case ： JSOP_ONE
      {
          ……
          goto nextOPcode;
      }
       ……
    }
}
```





4. 与浏览器交互相关

- 1.概述
- 引擎核心+扩展
- 引擎核心：语言特性的主持以及计算能力
- 扩展层：负责与浏览器各个对象进行交互
- DOM树的操作等：浏览器端来实现



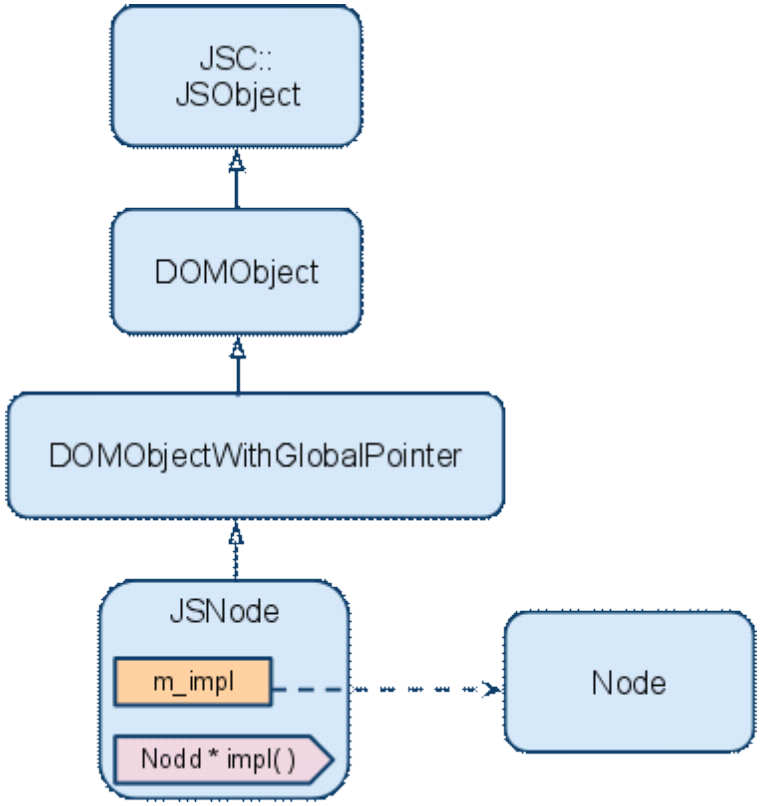
4. 与浏览器交互相关

UC优视

- 2.脚本执行的三种个起点
  - 全局JS代码
    - script标签
  - 定时器
    - setTimeout
    - setInterval
  - 事件监听器
    - onclick=‘alert ("hello")'等

4. 与浏览器交互相关

- 3.交互
  - 相互关系
  - DOM JS对象的属性操作
    - 只有get、set两种操作
    - 功能实现
  - 属性表——内嵌



4. 3交互

- <html>
- <body>
  -
- </body>
- <script>
- var domElement = document.getElementById("a");
- domElement.innerHTML = "hello world!";
- </script>
- </html>

4. 与浏览器交互相关

- 4.补充
  - 页面中每段执行的JavaScript都有一个与页面对应的执行环境
  - 页面指的是主页面以及主页面下面的Frame标签以及IFrame标签
  - 同一个主页面的不同Frame有一些关联

5. 引擎中其他的关键机制

- 1.对象的表示
- 2.内存回收机制
- 3.原型链的实现

5. 1对象的表示

- 1.概述
- JavaScript语言的特性之一——弱类型
- 内部表示：必须有类型
- 2.组成
  - 类型、值、属性
- 3.实现方式
  - 类型与值混合在一个字段里
  - 类型与值单独实现
  - 属性表——哈希表

5. 1对象的表示

- 4.代码示例
  - 类型包括：整数、浮点数、对象（包括字符串）
  - 基本数值

```
union {
        EncodedJSValue asEncodedJSValue;
        double asDouble;
#if (defined WTF_CPU_BIG_ENDIAN && WTF_CPU_BIG_ENDIAN)
        struct {
            int32_t tag;
            int32_t payload;
        } asBits;
#else
        struct {
            int32_t payload;
            int32_t tag;
        } asBits;
#endif
    } u;
```

5. 1对象的表示

- 5.属性
- {
- length：4
- name：”Ken"
- }
- 两种操作：获取（get）与设置（set）
- 内部表示：哈希表
  - 性能与内存的综合考虑
  - 缓存值、链表、哈希表的结合

5.2内存回收机制

UC优视

- 1.语言特性——不用自己管理内存的申请与释放
- 2.引擎负责这个功能
- 3.JavascriptCore使用的GC（垃圾收集）算法是Mark-Sweep算法。
- 4.Mark-Sweep算法的原理是：
- 1）GC管理着每一个堆内存单元(在JSC中简称Cell)，在JavaScriptCore中，一个JSCell对象所占用的内存就是一个对内存单元，JSC用一个CollectorCell的结构来表示和对应这个内存单元。
- 2）GC对每一个Cell都设置了一个对应的二进制的bit，“标记Cell”就是将bit设为1，清除cell的标记就是将bit设为0.

3）概括简单点说，GC执行Mark-Sweep的过程是：

- a.首先将所有Cell清除标记，也就是将所有bit设为二进制的0；
- b.然后从某几个具有全局性的Cell开始，遍历所有引用到的Cell，同时标记每一个遇到的Cell，也就是将Cell对应的bit设为二进制的“1”；
- c.标记完以后，遍历一次所有的bit，每遇到一个没被标记（也就是bit的值为0）的bit，根据bit找到对应的Cell，然后调用Cell对象的析构函数”~JSCell( )”。

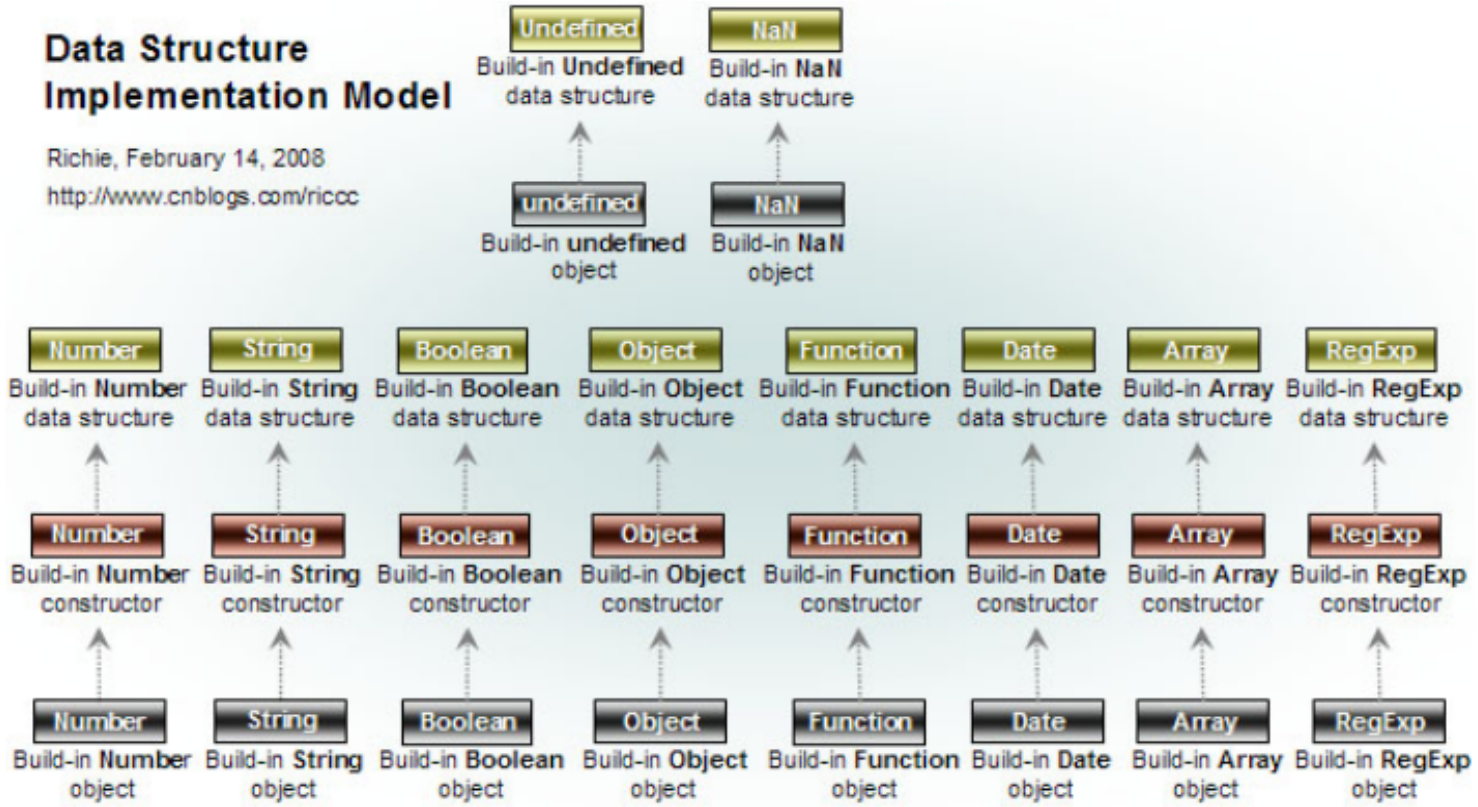
5. 2内存回收机制

UC优视

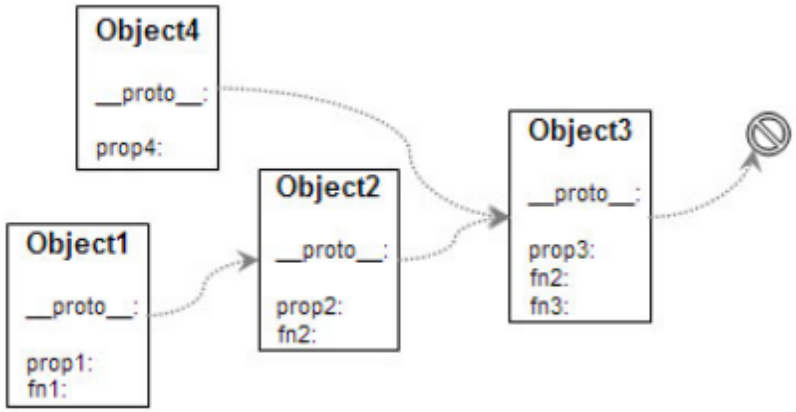
- 5.内存申请过程
  - 预分配内存、内存对应的位图
- 6.内存回收过程
  - 概念：可达性、根集
  - 根集：栈、记录的全局对象
  - mark与sweep
  - 标记过程：深度遍历与广度遍历的结合

5. 3原型链的实现

- 原型链是什么东西
- http://www.cnblogs.com/RicCC/archive/2008/02/15/JavaScript-Object-Model-Execution-Model.html

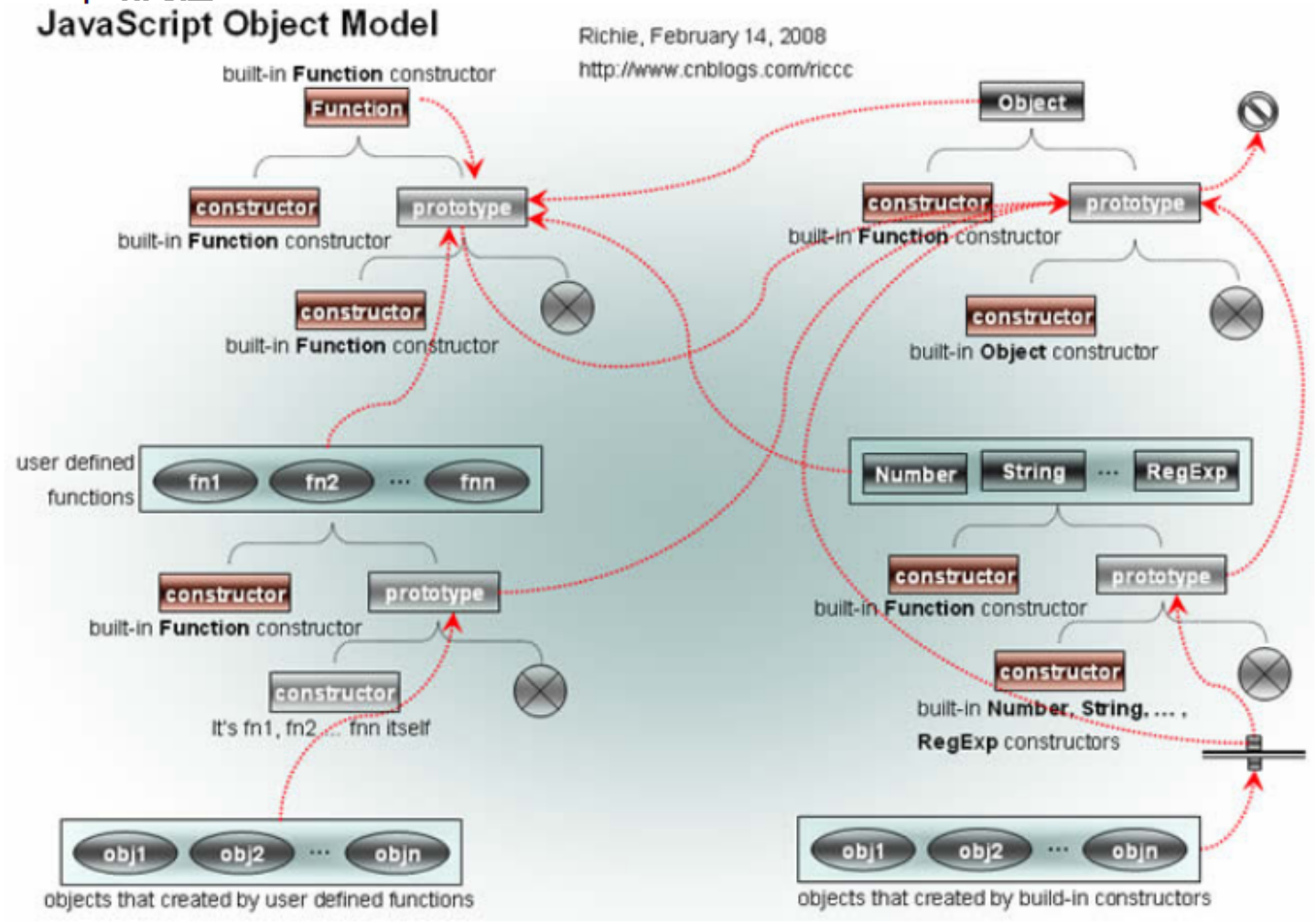


5.3原型链的实现


Object4
__proto__:
prop4:
Object3
__proto__:
prop3:
fn2:
fn3:
Object2
__proto__:
prop2:
fn2:
Object1
__proto__:
prop1:
fn1:

5.3原型链的实现

JavaScript对象模型



红色虚线表示隐式Prototype链。

6. webkit引擎补充说明

- 1.执行环境
  - 即，执行上下文
- 2.虚拟寄存器的解析执行原理
  - 与栈执行对应
- 3.JIT
  - 即时编译执行

6. 1执行环境

- 1.内容
  - 全局变量
  - 局部变量
  - 调用关系

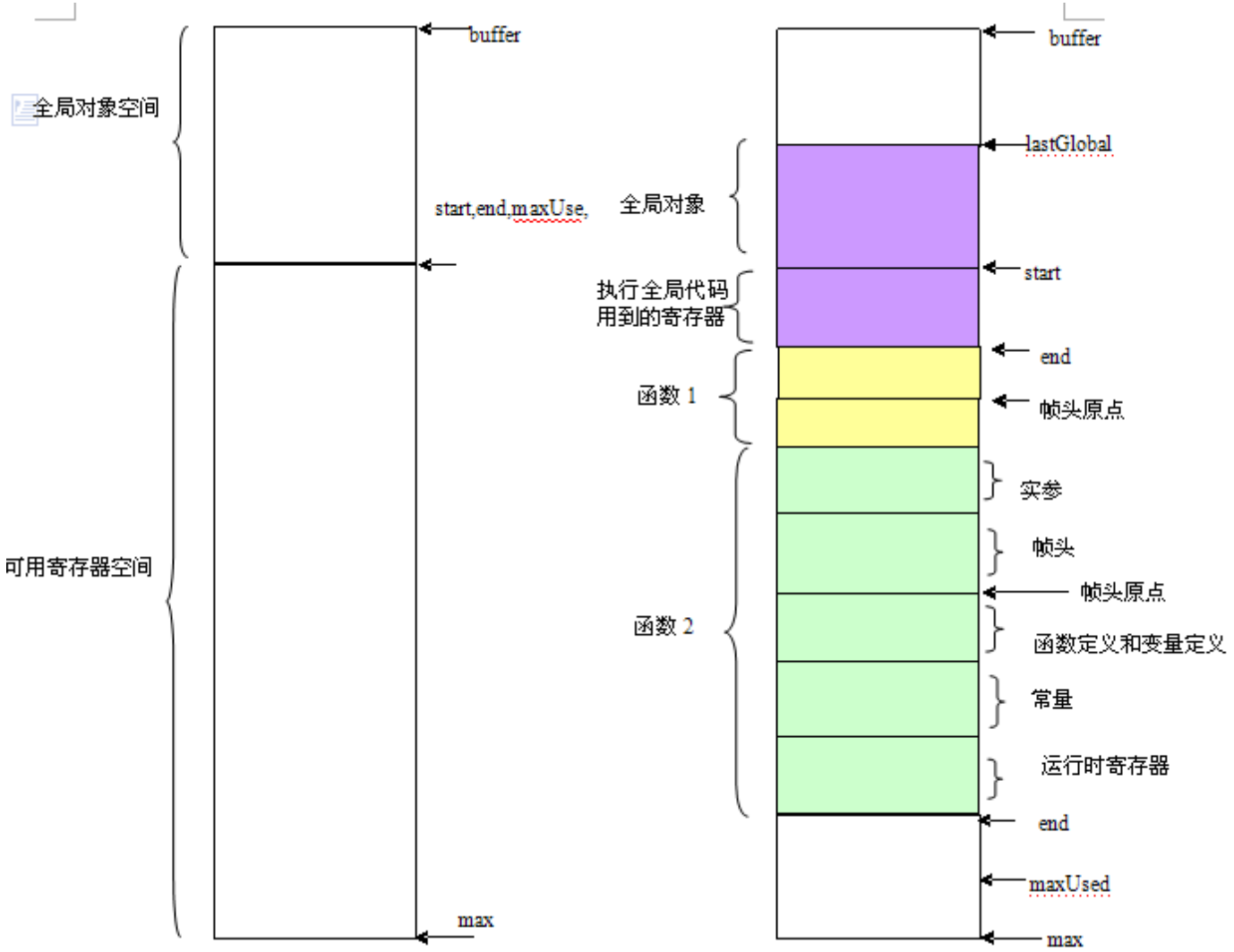
实例代码：

```
<script>
var a = "hello"
funtion foo()
{
    var b = "world"
    alert(a + " " + b)
}
foo()
<script>
```

6. 2虚拟寄存器的解析执行原理

- 1.概念
  - 基于栈与基于寄存器的指令集架构
  - 实例解析
  - 栈
  - iconst_1
  - iconst_2
  - iadd
  - istore_0
  - 寄存器
  - mov eax, 1
  - add eax, 2

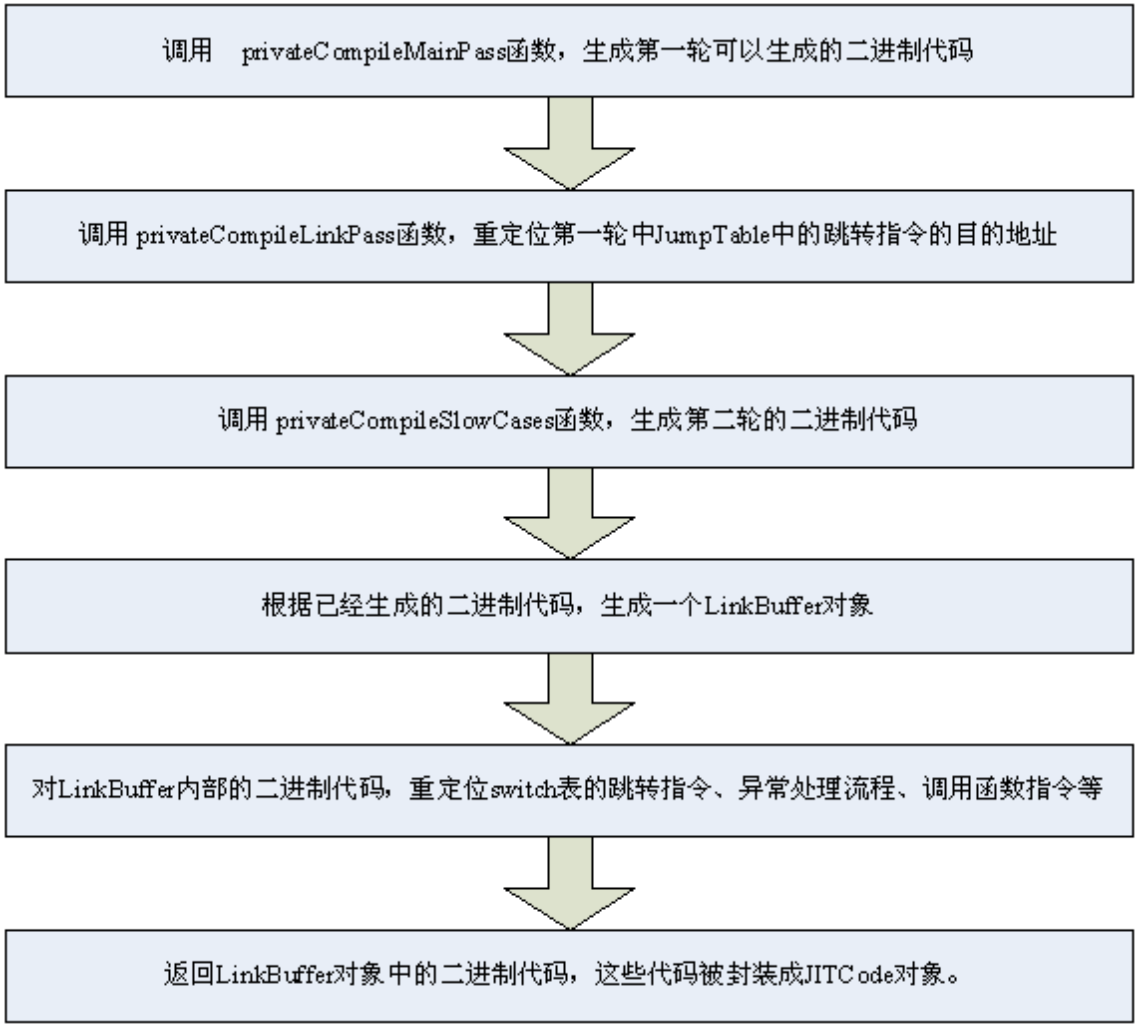
6.2虚拟寄存器的解析执行原理



6. 3JIT

UC优视

- 1.执行代码的生成（下页图）
- 2.生成后如何如何执行
  - 注意点：函数调用时的参数传递
- 3.其他注意点
  - a.机器相关
  - b.权限与对齐
  - c.指令缓存
  - d.常数池


调用 privateCompileMainPass函数，生成第一轮可以生成的二进制代码
调用 privateCompileLinkPass函数，重定位第一轮中JumpTable中的跳转指令的目的地址
调用 privateCompileSlowCases函数，生成第二轮的二进制代码
根据已经生成的二进制代码，生成一个LinkBuffer对象
对LinkBuffer内部的二进制代码，重定位switch表的跳转指令、异常处理流程、调用函数指令等
返回LinkBuffer对象中的二进制代码，这些代码被封装成JITCode对象。

thank you!

Q&A