文章编号: 1006-2475(2014)10-0066-05

# Chrome V8 引擎中的 JavaScript 数组实现分析与性能优化

刘 旭

(SAP 中国研究院商务智能部,上海 201203)

**摘要:** 随着实际应用中 JavaScript 程序的复杂度日益提高,程序执行的性能问题变得突出,而优化 JavaScript 程序中数组的使用方式可以提高程序效率。通过分析 JavaScript 数组的特点,结合广泛应用的 Chrome V8 引擎的源程序,描述 V8 对 JavaScript 数组的实现细节,重点分析 Fast Elements 和 Dictionary Elements 两种数组存储模式,给出对 JavaScript 数组性能优化的基本原则,并举例分析 4 个实际应用中可以优化的程序片段。

**关键词:** V8; Chrome; JavaScript; 数组; 性能; 优化

**中图分类号:** TP312; TP314 **文献标识码:** A **doi:** 10.3969/j.issn.1006-2475.2014.10.016

## Implementation Analysis and Performance Optimization for JavaScript Array in Chrome V8

LIU Xu

(Department of Business Intelligence of SAP Labs China, Shanghai 201203, China)

**Abstract:** With the complexity of JavaScript applications increasing, performance issues of JavaScript become increasingly prominent, but to optimize the use of JavaScript array can improve program efficiency. This paper, by analyzing the characteristics of JavaScript arrays and reviewing the source code of Chrome V8 JavaScript engine, describes the implementation details for JavaScript array in V8, and focuses on the Fast Elements and Dictionary Elements array storage modes. It also gives the basic principles of performance optimization of JavaScript array and four examples to show how to analyze and optimize programs in practical applications.

**Key words:** V8; Chrome; JavaScript; array; performance; optimization

## 0 引 言

作为一种动态的,弱类型的,兼具面向对象和函数式编程风格[1]的编程语言,JavaScript 已经成为 Web 应用开发事实上的标准,所有的现代 Web 浏览器均包含了 JavaScript 引擎。由于 JavaScript 在执行时一般并不会先编译为二进制文件,随着使用 JavaScript 编写的程序规模越来越大,JavaScript 程序的执行性能问题逐渐变得越来越突出,不理想的时间性能往往制约了 JavaScript 的进一步应用。

在 Web 应用程序的开发和使用中,Google Chrome 浏览器是最流行的几种 Web 浏览器之一[2]。Chrome V8 引擎(以下简称 V8)是 Chrome 浏览器中使用的 JavaScript 引擎,它同时也可以作为独立的 JavaScript 引擎应用于其他程序中。V8 是用 C++编写的,基于对商业友好的 New BSD 开源协议发布[3-4],其设计目的是高速执行大型 JavaScript 应用程序。V8 使用了一系列独有的技术以提高 JavaScript 的执行性能,比如在 JavaScript 程序首次运行时将其直接编译为机器代码,不使用中间字节码解释执行[5]。

数组是 JavaScript 语言的一项基本功能,几乎每个 JavaScript 应用程序中都会大量应用数组。由于 V8 的源程序公开,其运行机制可以通过分析其源程序来深入了解。通过分析 JavaScript 数组在 V8 中的实现,可以在编写 JavaScript 程序时明显提高程序性能。本文在分析中所使用的 Chrome 浏览器版本号为 35.0.1916.114,V8 引擎源码为 2014 年 5 月 30 日的 trunk 分支版本。

## 1 JavaScript 数组的特点

在很多高级语言中都实现了数组,C 语言里面的

收稿日期: 2014-07-28

作者简介: 刘旭(1982-),男,四川遂宁人,SAP 中国研究院商务智能软件部工程师,学士,研究方向: 数据可视化,用户数据的聚合、展现,人机交互及性能优化。

数组是一个典型实现,其特点是一组有序数据的集合[6],用索引表示数据的序号,数组中的每一个元素都属于同一个数据类型。C 语言中的数组长度是定义时指定的,数组不能动态扩展长度。

JavaScript 作为一种现代程序设计语言,对象是基本的数据类型。对象实际上是一种把很多值聚合在一起的方式,可以看作是属性名到值的映射。而 JavaScript 语言中的数组实际上是对象的特殊形式,即属性名是整数的对象。JavaScript 数组也可以看作值的有序集合,元素的位置即索引。

JavaScript 数组长度是动态的,在创建数组时无须声明一个固定的长度(但是也可以指定长度[7]),其长度根据需要可以自动增长。JavaScript 数组可能是数组索引不连续的稀疏数组。JavaScript 数组用 length 属性表示数组长度[8],对于非稀疏数组,该属性表示了数组元素的个数。JavaScript 数组元素的值是无类型的,同一个数组中的不同元素可能有不同的类型。

表 1　JavaScript 语言的对象、数组与 C 语言数组特点比较

| | JavaScript 对象 | JavaScript 数组 | C 数组 |
|---|---|---|---|
| 必须为整数键 | 否 | 是 | 是 |
| 必须具有 length 属性 | 否 | 是 | 否 |
| 动态改变长度 | 是 | 是 | 否 |
| 必须为同类型值 | 否 | 否 | 是 |
| 可以稀疏存储 | 是 | 是 | 否 |

通过 JavaScript 对数组的定义可以看出,数组的整数索引的特性可以让数组在实现中进行优化,比如在合适的条件下尽可能使用一块连续的存储空间来按照 C 语言中的数组方式存储和查找。连续存储的时间效率有可能极大地优于常规的对象属性处理方式。然而,相对于 C 数组,JavaScript 数组的动态性特点让其在某些条件下又必须改变连续存储方式,但是实际应用程序中,确实需要利用动态性的场景相当有限[9]。在实际的程序设计中,当这些改变并不必要时,就会造成性能的下降,而这些下降很多是可以在保持程序逻辑的前提下优化的。

## 2　V8 对 JavaScript 数组的实现分析

通过分析 V8 源程序,可知 V8 对 JavaScript 数组的实现是高效和具有代表性的。按照 JavaScript 标准,对象的属性名应该是字符串,而数组是特殊的对象,这意味着对象和数组的属性都可以通过“[]”来访问。在很多情况下,V8 需要首先区分应该按照对象还是数组进行操作。以下是 V8 对数组元素赋值的代码片段,可以看到 JavaScript 数组的 key 值将转换为字符串,然后判断其是否可以用作数组索引[10]:

```
// Call-back into JavaScript to convert the key to a string.
    Handle < Object > converted;
    ASSIGN_RETURN_ON_EXCEPTION( isolate, converted,
Execution::ToString( isolate, key) , Object) ;
    Handle < String > name = Handle < String > :: cast( converted) ;
    if ( name-> AsArrayIndex( &index) ) {
    return JSObject::SetElement( js_object, index, value, attr,
    strict_mode,true,set_mode) ;
    } else{
    return JSReceiver::SetProperty( js_object, name, value, attr, strict_mode) ;
    }
```

当 key 不能用作数组索引时,V8 将按照对象属性的方式处理,不会更新数组的 length 属性。当 key 为适当大小的无符号整数,可以用作数组索引时,V8 将对数组分为 Fast Elements 和 Dictionary Elements 两种存储方式进行存储[11]。Fast Elements 是传统的线性存储方式,而 Dictionary Elements 使用的是 Hash 表存储,在 V8 中需要用专门的类来进行处理。

### 2.1　Fast Elements 模式

对于一个新创建的空数组赋值时,默认使用 Fast Elements 方式,数组的存储空间是可以动态增长的。V8 代码中的 capacity 表示当前实际已经分配的空间,例如使用 new Array( 100) 创建的数组,其 capacity 为 100,而使用 new Array 创建的数组,其 capacity 为 0。如果对一个更大的索引赋值,V8 会动态开辟更大的内存。

Fast Elements 模式的一个扩展是 Fast Holey Elements 模式。此模式适合于数组中只有某些索引存有元素,而其他的索引都没有赋值的情况。在 Fast Holey Elements 模式下,没有赋值的数组索引将会存储一个特殊的值,这样在访问这些位置时就可以得到 undefined。但是 Fast Holey Elements 同样会动态分配连续的存储空间,分配空间的大小由最大的索引值决定。

当对数组上不连续的索引位置赋值时,V8 可能将 Fast Elements 转化为 Fast Holey Elements 模式以处理带“空洞”的数组,对应的 V8 源程序如下[12]:

```
    bool introduces_holes = true;
    if ( object-> IsJSArray( ) ) {
    CHECK( Handle < JSArray > :: cast( object) -> length( ) ->
ToArrayIndex( &array_length) ) ;
    introduces_holes = index > array_length;
    if( index >= array_length) {
    must_update_array_length = true;
    array_length = index + 1;
    }
    } else{
    introduces_holes = index >= capacity;
```

```
}
//If the array is growing, and it's not growth by a single element at the end, make sure that the ElementsKind is HOLEY.
ElementsKind elements_kind = object->GetElementsKind();
if (introduces_holes && IsFastElementsKind(elements_kind) &&! IsFastHoleyElementsKind(elements_kind)) {
ElementsKind transitioned_kind = GetHoleyElementsKind(elements_kind);
TransitionElementsKind(object,transitioned_kind);
}
```

### 2.2 Dictionary Elements 模式

在 Dictionary Elements 模式下,数组的值存储于 Dictionary < Derived, Shape, Key > 类的子类中[13],如 SeededNumberDictionary,而 Dictionary 类继承于 HashTable 类,实际上就是使用 Hash 方式存储。此方式最适合于存储稀疏数组,它不用开辟大块连续的存储空间,节省了内存,但是由于需要维护这样一个 HashTable,其存储特定值的时间开销一般要比 Fast Elements 模式大很多。

一种由 Fast Elements 转换为 Dictionary Elements 的典型情况是对数组赋值时使用远超当前数组大小的索引值,这时候要对数组分配大量空间则将可能造成存储空间的浪费。在 Fast Elements 模式下,capacity 用于指示当前内存占用量大小,通常根据数组当前最大索引的值确定。在数组索引过大,超过 capacity 到一定程度(由 kMaxGap 决定,其值为 1024),数组将直接转化为 Dictionary Elements 模式,其对应的 V8 源程序如下:

```
// Check if the capacity of the backing store needs to be increased, or if a transition to slow elements is necessary.
if (index >= capacity) {
bool convert_to_slow = true;
if ((index-capacity) < kMaxGap) {
new_capacity = NewElementsCapacity(index + 1);
ASSERT(new_capacity > index);
if (! object-> ShouldConvertToSlowElements(new_capacity)) {
convert_to_slow = false;
}
}
if (convert_to_slow) {
NormalizeElements(object);
return SetDictionaryElement(object, index, value, NONE, strict_mode, check_prototype);
}
}
```

## 3 运行于 V8 中的 JavaScript 数组性能优化

要优化 JavaScript 程序性能,实际上就是在保证程序正确的前提下,通过调整程序的逻辑结构提高执行效率[14],而基于对 JavaScript 引擎实现的深入分析,可以更有针对性地改进程序结构。从 V8 对数组的实现可知,要优化数组性能,基本的原则就是让数组少动态分配内存,尽可能保持数组的 Fast Elements 方式,尽量避免数组从 Fast Elements 方式切换为其他方式。在 V8 引擎下 JavaScript 对数组的赋值及访问操作中,从以下的例子中可以看到一些简单的数组操作方式改动会带来时间性能的明显提升。以下程序运行时间统计于一台 CPU 为 Intel Core i7 2.93 GHz,内存 16 GB,操作系统为 64 位 Windows 7 企业版的电脑。

### 3.1 避免使用负整数作为索引

JavaScript 的数组实际上并不支持负数作为索引,但是由于 JavaScript 是弱类型的,当程序试图在数组中使用负数索引时,JavaScript 并不会有任何出错提示,而是认为需要将其按照一般的对象属性进行处理。对应用程序来说,负数索引似乎可以工作,但是 V8 无法将这样的对象属性利用数组的特点进行线性存储,一般情况下,其时间和空间开销都将比数组大很多。以代码 1 为例:

```
var tempArray = [];
var i;
for(i=0; i>-5; i--)
{
tempArray[i] = 1;
}
```

在代码 1 中,实际上只有 tempArray[0] 是数组元素,从 -1 开始,所有的值都作为属性值添加到 tempArray 上了,而且 V8 不会更新 tempArray 的 length 属性,执行完之后,tempArray.length 还是 1。由于 V8 将为新添加的属性创建 hidden class[15],其速度比常规数组慢得多。如果使用 0 到 5 作为索引,则性能会有明显提高,因为对于这样的常规数组,V8 可以使用 Fast Elements 模式进行线性存储,其详见代码 2。

```
var tempArray = [];
var i;
for(i=0; i<5; i++)
{
tempArray[i] = 1;
}
```

表 2 代码 1 与代码 2 循环执行 1,000,000 次时间比较

| 代码 1 | 代码 2 | 代码 2 与代码 1 时间百分比 |
|---|---|---|
| 2001.000 ms | 63.000 ms | 3.15% |

可以看出两者性能的差异非常大。因此,在运行于 V8 的 JavaScript 应用程序中,在能使用数组的场合下,不要使用对象,而且需要尽量避免使用负数作为数组索引。同理,浮点数、普通字符串也最好不要用作数组索引。

### 3.2　预先指定数组大小

虽然 JavaScript 规定应用程序可以不用预先指定数组大小,但是在 V8 的实现中,如果程序预先指定了使用的数组大小,而数组大小适合使用 Fast Elements 进行连续存储时,则可以避免额外分配内存空间。以代码 3 为例:

```
var tempArray = new Array( ) ;
var i;
for ( i = 0; i < 100; i ++ ) {
tempArray [i] = 1;
}
```

V8 将在 tempArray 增长的过程中不断动态分配空间。如果使用 new Array( 100) 预先分配数组大小,则性能会有大幅度提高。这是因为指定大小后,数组 capacity 一开始就为 100,V8 不用频繁更新数组的长度,减少了不必要的内存操作,如代码 4 所示:

```
var tempArray = new Array( 100) ;
var i;
for ( i = 0; i < 100; i ++ ) {
tempArray [i] = 1;
}
```

表 3　代码 3 与代码 4 循环执行 1,000,000 次时间比较

| 代码 3 | 代码 4 | 代码 4 与代码 3 时间百分比 |
|---|---|---|
| 810.000 ms | 326.000 ms | 40.25% |

从结果可以看出,在已经知道需要使用的数组大小的情况下,应该考虑在程序中预先分配数组大小,在不必要的情况下,尽量不要使用超过数组大小的索引以导致数组空间的动态分配。

### 3.3　避免使用不连续的索引值

在一个未指定长度的数组中,对于索引的增加最好是连续的,因为不连续的索引可能会改变数组存储方式。考虑如下在数组中 5 个位置赋值的代码 5:

```
var tempArray = [];
var index = 0;
for ( var i = 0; i < 5; i ++ ) {
tempArray [index] = 1;
index += 20;
}
```

如果将这里的 20 替换为 1,则可以明显改善数组性能。这是因为 20 将触发数组由默认的 Fast Elements 向 Fast Holey Elements 模式的转换,V8 需要处理“空洞”的情况,带来时间上的开销,而 1 不会。而且在这 2 种方式下,由于创建时未指定数组长度,数组的空间都是连续动态分配的,最大的索引越大,分配的空间越多,使用 1 可以尽可能减少分配的数组空间,如代码 6 所示。

```
var tempArray = [];
var index = 0;
for ( var i = 0; i < 5; i ++ ) {
tempArray [index] = 1;
index += 1;
}
```

表 4　代码 5 与代码 6 循环执行 1,000,000 次时间比较

| 代码 5 | 代码 6 | 代码 6 与代码 5 时间百分比 |
|---|---|---|
| 288.000 ms | 51.000 ms | 17.71% |

从表 4 结果可以看出,在一般情况下,最好是使用从零开始连续的整数索引,减少数组动态分配的空间,避免数组切换到 Fast Holey Elements 模式。

### 3.4　避免逆序对数组赋值

对于数组中每个元素赋值是数组初始化的常见操作。在一个未指定长度的数组中,如果按照由大到小的索引对数组赋值,在数组长度较大的情况下,数组会直接切换为比较慢的 Dictionary Elements 模式,从而引起不必要的性能降低。考虑如下对 10000 个数组元素赋值的代码 7:

```
var tempArray = [];
var i;
for( i = 9999; i >= 0; i--)
{
tempArray [i] = 1;
}
```

原因是对 tempArray [9999] 赋值时,由于 9999 已经足够大,V8 将使用 Dictionary Elements 方式存储数组,导致存储速度变慢( 虽然 Dictionary Elements 方式还能在某些情况下转换为 Fast Elements 方式) 。解决的方法就是使用从零开始的从小到大的顺序对数组赋值。以下的代码 8 不会引发数组转换为 Dictionary Elements 方式:

```
var tempArray = [];
var i;
for( i = 0; i <= 9999; i ++)
{
tempArray [i] = 1;
}
```

表 5　代码 7 与代码 8 循环执行 1,000 次需要的时间

| 代码 7 | 代码 8 | 代码 8 与代码 7 时间比 |
|---|---|---|
| 687.000 ms | 102.000 ms | 14.85% |

由表 5 可以看出,代码 8 的时间性能提升比较明显。因此,对数组赋值时尽量使用从零开始的从小到大

的索引比较有利,特别是在数组长度比较大的情况下。

## 4 结束语

本文从分析 JavaScript 数组的特点入手,结合 Chrome V8 引擎的源程序,重点分析了 V8 对于 JavaScript 数组的 2 种实现模式以及每种模式适用的条件,提出使用 JavaScript 编写应用程序使用数组时在性能优化上需要注意的基本原则,并且给出几个在实际情况中比较容易进行优化的程序片段,统计优化前后的执行时间作为例证。在实际应用中,不仅可以使用本文的原则改善数组性能,还可以参照本文的方式,通过分析 V8 引擎的实现从而对 JavaScript 程序的其他方面进行分析与优化。

参考文献:


[1] David Flanagan. JavaScript: The Definitive Guide [M]. 6th ed. USA: O'Reilly Media, 2011.

[2] W3schools. com. Browser Statistics [EB/OL]. http://www.w3schools.com/browsers/browsers_stats.asp, 2014-06-22.

[3] V8 JavaScript Engine Developers. Project Information [EB/OL]. https://code.google.com/p/v8/, 2014-06-22.

[4] Open Source Initiative. The BSD 3-Clause License [EB/OL]. http://opensource.org/licenses/BSD-3-Clause, 2014-06-22.

[5] V8 JavaScript Engine Developers. Design Elements [EB/OL]. https://developers.google.com/v8/design, 2014-06-22.

[6] Brian W Kernighan, Dennis M Ritchie. C Programming Language [M]. 2nd ed. Englewood Cliffs: Prentice Hall, 1988.

[7] Stoyan Stefanov. JavaScript Patterns [M]. USA: O'Reilly Media, 2010.

[8] Douglas Crockford. JavaScript: The Good Parts [M]. USA: O'Reilly Media, 2008.

[9] 李世胜,程步奇,李晓峰,等. 基于预测的 JavaScript 类型系统研究 [J]. 计算机研究与发展, 2012,49(2):421-431.

[10] V8 JavaScript Engine Members. V8 JavaScript Engine Source Code [CP/OL]. http://v8.googlecode.com/svn/trunk/src/runtime.cc, 2014-05-30.

[11] Jay Conrod. A Tour of V8: Object Representation [EB/OL]. http://jayconrod.com/posts/52/a-tour-of-v8-object-representation, 2014-06-22.

[12] V8 JavaScript Engine Members. V8 JavaScript Engine Source Code [CP/OL]. http://v8.googlecode.com/svn/trunk/src/objects.cc, 2014-05-30.

[13] V8 JavaScript Engine Members. V8 JavaScript Engine Source Code [CP/OL]. http://v8.googlecode.com/svn/trunk/src/objects.h, 2014-05-30.

[14] 雷军程,柳小文,朱鸿鹏. JS 分支结构性能优化研究 [J]. 邵阳学院学报(自然科学版), 2011,8(4):27-29.

[15] Chris Wilson. Performance Tips for JavaScript in V8 [EB/OL]. http://www.html5rocks.com/en/tutorials/speed/v8/, 2012-10-11.


(上接第 60 页)

## 4 结束语

随着移动互联网的飞速发展以及智能手机的普及,越来越多的计算机上的应用程序转到了手机平台上,Android 是一种基于 Linux 的自由及开放源代码的操作系统,因其具有开放性以及丰富的硬件选择性,更加受到用户的青睐。本文设计了一种基于 Android 平台的校园故障报修系统,首先介绍校园报修系统发展的背景以及相关技术[15-17],接着分析了校园故障报修系统的整体架构,最后重点阐述 Android 手机客户端系统的设计与实现。基于 Android 平台的校园故障报修系统的运行使用,方便了师生报修,有利于提高校园后勤部门处理故障报修的效率。

参考文献:


[1] 杨海波,许礼捷,岳浩. 基于 Web 的网络中心报修管理系统的设计与实现 [J]. 电脑知识与技术, 2011,7(23):5610-5615.

[2] 崔洋,陈光,沈佳. 基于 B/S 模式的网络虚拟实验系统的设计与实现 [J]. 计算机与现代化, 2013(5):117-120.

[3] 唐存东,全上克,王志平. 高校实验室教学管理系统设计与实现 [J]. 计算机与现代化, 2013(9):113-116.

[4] 肖飞,乔荣爱. 基于 Android 的校园报警平台的设计与实现 [J]. 计算机与现代化, 2013(8):222-224.

[5] 王凌燕,吴华,王丽燕. 基于 Android 的英语语法移动学习系统设计与实现 [J]. 计算机与现代化, 2013(8):232-235.

[6] 徐富新,王洲,陈芳,等. 手机短信在实验选课系统的应用 [J]. 计算机技术与发展, 2013,23(11):152-156.

[7] 倪红军,钱昌俊. 基于 Android 平台的自发短信系统设计与实现 [J]. 电子技术应用, 2012,38(12):126-129.

[8] 顾宁,刘家茂,柴晓路,等. Web Services 原理与研发实践 [M]. 北京:机械工业出版社, 2006.

[9] 段运生,吴先良. 基于 Web 服务的统一身份认证系统的设计与实现 [J]. 安徽大学学报(自然科学版), 2008,32(1):28-31.

[10] Matthew MacDonald, Adam Freeman, Mario Szpuszta,等. ASP. NET 4 高级程序设计 [M]. 4 版. 博思工作室译. 北京:人民邮电出版社, 2011.

[11] 丁士锋. C#典型模块与项目实战大全 [M]. 北京:清华大学出版社, 2012.

[12] 余志龙,王世江. Google Android SDK 开发范例大全 [M]. 2 版. 北京:人民邮电出版社, 2010.

[13] Ed Burnette. Android 基础教程 [M]. 3 版. 田俊静,张波,黄湘情,等译. 北京:人民邮电出版社, 2010.

[14] 郭宏志. Android 应用开发详解 [M]. 北京:电子工业出版社, 2010.

[15] 史萌,蒋朝惠. 基于. NET 平台的网上营业厅系统设计 [J]. 计算机技术与发展, 2014,24(3):183-186.

[16] 裴文莲,詹林. Android 平台上 WiFi 技术在商场员工定位系统中的应用 [J]. 计算机与现代化, 2013(2):159-162.

[17] 肖飞,贾满磊. 基于 Android 平板电脑的电子菜单设计与实现 [J]. 计算机与现代化, 2013(8):236-238.