我是如何理解”Another JavaScript quiz”中的题目

这篇文章发布于 2013年05月7日，星期二，18:02，归类于 JS实例。 阅读 67347 次, 今日 9 次 28 条评论

一、你这个变态，给我滚开

时光冉冉，或多或少见过一些JavaScript相关的题目，其中很多属于变态级别的！各种奇怪符号写法拼在一起、尼玛还有兼容性问题，估计道格拉斯都不知道答案。

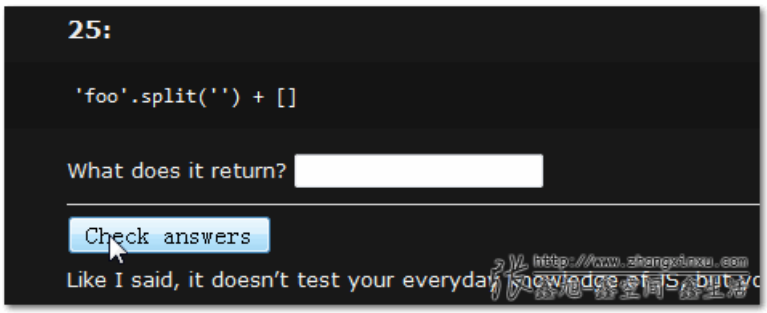
对于这种整得亲妈都不认识的变态问题，实际上是没有什么参考价值的。好比要考察外星人对人类的了解，结果你那下面这货来做测试，看到亲戚的外星人一定会云里雾里的，但有意义吗?

好的JavaScript测试题目应该是：门外汉见了皱眉，行家见了疑惑题目是不是简单了点，同时考察点覆盖全面。//zxx: 我目前还没有这个能耐设计出如此优秀的题目。

这里要介绍的”Another JavaScript quiz“(by james)中的题目不是属于变态题目，而是确实属于变态题目，不过是表面上的，很多内容确实可能会遇到的。综合评价下就是：面试价值不及格，学习价值是很赞的，因此，探讨分享很有意义。

“Another JavaScript quiz“中有25个很简洁的JavaScript测试题，全部都是考察返回值的，例如`1 && 3`的返回值是?

25道题目全部完成后，可以点击下图所示的按钮，检查你的正确率以及那些题目出错了。



我自己快速测试了下，如果每题算4分的话，我的成绩是56分，不及格，可见自己对JS的学习以及理解还有一段路要走。

您也可以做一下这些题目，完成之后，若有疑惑，可以参考我下面纯个人的理解 – 部分理解可能不准确，欢迎指正。同时，网上应该有其他一些前辈的解释，您也可以参考参考。

二、公子，不急，慢慢来嘛

1.
```
1 && 3 //返回的是?
```

结果是`3`.

`&&`是几乎无人不知，表示“与”。男方要娶水灵灵的妹子，七大姑八大婆都要同意，否则继续锻炼右手吧。

上面`1 && 3`本质上等同于`大姑妈 && 三姑妈`。

从左往右，如果“大姑妈”通过，继续“三姑妈”；否则直接返回“大姑妈”。因为“三姑妈”后面没有其他亲戚了，因此直接返回“三姑妈”。

在JavaScript中，`1 == true // true`，因此，`1 && 3`等同于“大姑妈”通过了，返回最后一个检测的亲戚“三姑妈”，也就是这里的结果`3`.

因此，实际上，我们平时的`if (1 && 3) {}`等同于`if (3) {}`.

如果这里的题目换成`0 && 3`返回的是？那结果是如何呢?

因为`0 == false // true`,因此走不到“三姑妈”这一关，直接返回了`0`.也就是`if (0 && 3) {}`等同于`if (0) { /* 不会执行 */ }`.

实用性
可以避免`if`嵌套。例如，要问页面上某个dom绑定点击事件，我们需要先判断这个dom元素存不存在，我们可能会这样做:

```
var dom = document.querySelector("#dom");
if (dom) { dom.addEventListener("click", function() {});  }
```

实际上，我们可以使用`&&`做一些简化:

```
var dom = document.querySelector("#dom");
dom && dom.addEventListener("click", function() {});
```

2.
```
1 && "foo" || 0 //返回的是?
```

很简单的一道题，结果是`"foo"`.

这里出现了一个新的关系符`||`,表示“或”的意思。男方要娶干巴巴的妹子，七大姑八大婆只要一个说好就可以了。比方说`1 || 3`表示：如果大姑妈说可以，则成了，返回大姑妈；如果大姑妈不允许，再看看三姑妈的意思。

因此，`if (1 || 3) {}`实际等同于`if (1) {}`;`if (0 || 3) {}`等同于`if (3) {}`.

实用性
`||`可以让我们使用一种更快捷简易的方式为参数添加默认值。例如，写jQuery插件的时候，可选参数是可以缺省的，此时实际上值为`undefined`，会让后面的参数`extend`产生困扰。因此，我们会经常见到类似这样的代码:

```
$.fn.plugin = function(options) {
    options = options || {};
    // ...
};
```

显然，根据上面的理解，`1 && "foo"`返回的是`"foo"`，而`"foo" || 0`，因`"foo" == true`为`true`，自然返回是“大姑妈”–`"foo"`. 这就是最终结果`"foo"`的由来。

3. `1 || "foo" && 0 //返回的是?`

结果是`1`.

据说逻辑与运算符(`&&`)优先级要大于逻辑或(`||`)。如果是这样，那这里的结果返回应该是这样子的：`1 || "foo" && 0` → `1 || 0` → `1`.

4. `(1, 2, 3) //返回的是?`

结果是`3`.

这里考察的是逗号运算符，也称多重求值。逗号运算符据说是优先级最低的运算符。
一般表现形式是：大姑妈，二姑妈，三姑妈，……N姑妈。
运算规则如下：折腾大姑妈，折腾二姑妈，折腾三姑妈，……折腾N姑妈，最后返回的N姑妈的返回值。

因此，这里`1, 2, 3`就是折腾了`1`，折腾了`2`，折腾了`3`并且返回了`3`的返回值。因此，结果是`3`.

好，现在提问：`alert(1, 2, 3);`的弹出值是?

结果是`3`……是不可能的！正确结果是**1**.

不要困扰。原因很简单，`alert`身后的`()`实际上也是一种运算符，这里指函数调用，优先级相当的高。这里，`1, 2, 3`已经被奴役成`alert`方法（弹出第一个参数值）的参数了，因此，弹出的是`1`.

如果是`alert((1, 2, 3));`则弹出的就是`3`了。

实用性
举个很简单的例子，`i`，`j`两个变量同时递增，可能我们会这样写：

```
var i=1;
var j=1;
i+=1;
j+=2;
```

我们可以借助逗号运算符获得更好地阅读体验，至于性能是否有提高，我个人并不清楚，就算有，差异也是可以忽略的。

```
var i=1, j=1;
i+=1, j+=2;
```

5.
```
x = { shift: [].shift };
x.shift();
x.length; //返回的是?
```

结果是IE6/IE7 `undefined`, 其他浏览器 `0`. 不过，个人观点，IE6/IE7其实并不属于这个题目的考察浏览器。因此，我们着重讨论为何IE8+等浏览器为何结果是 `0`.

首先，我们了解下数组的 `shift` 方法。 我个人一般把 `pop/push` 归为一对基友，移除尾部元素和尾部添加元素，都是做扫尾工作的。 `shift/unshift` 当作另外一对基友，移除头部元素和头部添加元素，都是在头部打点的。记住，*单词字符个数少的(`pop` , `shift`)都是做删除的*，这样记忆就不会混淆了。

从外表上看， `shift` 方法移除数组中的第一个元素并返回这个元素。如果这个数组是个空数组，则返回 `undefined` . 一般而言， `shift` 操作会改变后面所有数组项在内存中的地址，因此，相比 `pop` 方法要慢得多。

从内在来看， `shift` 还能给对象自动添加 `length` 属性。~~Mozilla开发者中心(MDC)~~Mozilla开发者网络(MDN – Mozilla Developer Network)的解释是这样子的:

> `shift` is intentionally generic; this method can be `called` or `applied` to objects resembling arrays. Objects which do not contain a length property reflecting the last in a series of consecutive, zero-based numerical properties may not behave in any meaningful manner.

简体中文表示就是（这里的释义自己只有80%确认，若有不准确，请极力指证）：

> `shift` 可以有意泛化（变身成鸭子）；该方法可以被类数组对象 `call` 或者 `apply` . 对象如果没有 `length` 属性，可能会以无意义的方式在最后反射一系列连续，基于 `0` 的数值属性。

什么意思呢？我们看下面这个更容易理解的问题:

```
x = {};
[].shift.call(x);
x.length; //返回的是?
```

结果是 `0` . //zxx: IE6/IE7 `undefined` .

`x` 对象原本木有 `length` 属性，在被数组 `shift` 方法 `call` 后，添加了一个值为 `0` 的 `length` 属性。此为数组 `shift` 方法的泛化性，专业术语为*泛型(generic)*。其基友方法，例如 `pop` , `push` 等都是如此。

OK, 回到原题，实际上， `x.shift()` 的调用等同于 `[].shift.call(x)` , 不明白？看下面的一步一步分析。

```
var x = {
    shift: function() {
        console.log(this === x); // true
    }
};
x.shift();
```

因此，在

```
x = { shift: [].shift }
```

条件下，
`x.shift`执行的就是`[].shift`的执行，只不过，`[].shift()`函数中的`this`上下文就是`x`(因为`this===x`)，就等同于直接的`[].shift.call(x)`调用。

这条题目中`x`对象的`shift`属性名实际上是用来干扰，提高解答难度的刻意命名。我们使用其他命名，结果也是一样的。

```
x = { shit: [].shift };
x.shit();
x.length; // 0
```

6.
```
{foo:1}[0] //返回的是?
```

结果是`[0]`,或者这种表现形式`0 { 0 : 0 }` – 来自IE控制台.

不要试图使用`alert`或者控制台`console.log`输出，这只会返回不一样的结果`undefined`，哦？为何会有这等差异?

出题者james在”Labelled blocks, useful?“中有这样的解释：

> Since JavaScript doesn’t have block scope, using a block anywhere other than in the conventional places (if/while etc.) is almost totally pointless. However, as I mentioned, we could use them to a nnotate and contain logically related pieces of code…

意思是说：

> 因为JavaScript没有块作用域，所以，如果语句块不是常规使用，如`if/while`等，其几乎就是打酱油的。甚至，我们可以利用这个特性注释或者包含相关的逻辑片段代码…

我们有必要好好理解这里“打酱油的”意思，这里的“打酱油”并不是指`{}`块中语句是打酱油，而是其本身就是个酱油。嘛意思，实例说明一切：

```
2 // 返回值为2
```

```
{2} // 返回值为2
```

```
str = "string" // 返回值为string
```

```
{ str = "string" } // 返回值为string
```

```
foo: 1 // 返回值为1
```

```
{ foo: 1 } // 返回值为1
```

也就是说花括号几乎就是皇帝的新衣。因此，这里的答案就不难理解了，`{foo:1}[0]`实际上就是`foo:1; [0]`.返回的就是`[0]`本身。

注意这里反复出现的措辞“几乎”。“几乎酱油”的潜台词是有时候还能顶个臭皮匠。james举了个在块中使用 `break` 语句的例子，如下：

```
var x = 1;
foo: {
    x = 2;
    break foo;
    x = 3;
}
x === 2; // true
```

非这种情况的 `break/continue` 只能在 `switch` 语句以及循环中使用。很有意思吧，我反正是学习了  ！

7. `[true, false][+true, +false] //返回的是?`

结果是 `true`.

首先，我们了解下 `+true` 和 `+false` 是个什么东西。

想必都清楚 `+1` 与 `-1` 是什么东东。指的是 `正数1` 与 `负数1`. 同时，稍微对JS有了解的人也清楚， `true == 1`, `false == 0`. 因此，实际上， `[+true, +false]` 就是 `[+1, +0]`.

`[true, false]` 为数组，后面的 `[+true, +false]` 实际为索引，然而索引只需要一个值，因此， `[+true, +false]` 返回的实际是我们上面提到的逗号运算——返回最后一个值，也就是 `+0`，也就是 `0`.

因此，本题的问题其实是：

```
[true, false][0] //返回的是?
```

旺财估计都已经明白了，就不多说了。

8. `++'52'.split('')[0] //返回的是?`

结果是 `6`.

此题难点在于运算符的优先级，看来有必要把运算符优先级都展示下。参见下表（数据取自美拓blog）：

运算符	
描述	
1	
. [] ()	
字段访问、数组下标、函数调用以及表达式分组	
2	
++ — - ~ ! delete new typeof void	
一元运算符、返回数据类型、对象创建、未定义值	

3
* / %
乘法、除法、取模
4
+ – +
加法、减法、字符串连接
5
< < >> >>>
移位
6
< <= > >= instanceof
小于、小于等于、大于、大于等于、instanceof
7
== != === !==
等于、不等于、严格相等、非严格相等
8
&
按位与
9
^
按位异或
10
|
按位或
11
&&
逻辑与
12
||
逻辑或
13
?:
条件
14
= += -=
赋值、运算赋值
15
,
多重求值

从上表可以看出点`.`以及`[]`要比`++`的优先级高。因此，这里的问题等同于：

```
var result;
result = "52".split(""); // ["5", "2"]
result = result[0]; // "5"
result = ++result;  // 6
```

注意：
虽然说`++["5", "2"][0]`的返回值是`6`，但是直接`++"5"`或者`++5`却是会报错的（FireBug控制台显示*“invalid increment operand”*错误），据说是因为只有变量才能`++`而`'5'`只是一个字符串。

9.
```
a: b: c: d: e: f: g: 1, 2, 3, 4, 5; //返回的是?
```

结果是：`5`.

这里需要讲下标记语句(Labelled Statements). 下文为我理解的ECMAScript文档中的解释：

> 语句可以有个标签前缀。标记语句只和`break`或`continue`标记的语句结合使用（想想第7题最后那个`break`的例子）。ECMAScript中并无goto语句。
>
> ECMAScript程序中，如果标记语句的标签标识符有重复，则会出错。比方说`a: {a: 1};`就会嗝屁(FireBug显示SyntaxError: duplicate label错误)，而`a: {b: 1};`继续逍遥快活。但是，这并不适用于标记语句中或嵌套或直接或间接FunctionDeclaration体中出现的标签。如`a: (function() {a: 1});`就是OK的。
>
> 生产标识符(Identifier)：语句的执行是通过给标签集增加标识符，然后再执行语句。如果标签语句本身就有一个非空的标签集（冒号后面的语句还有类似结构，例如`a: b: 1`，`b:1`为`a`标签集的标签语句，但这个语句本身就含有一个标签集），则在语句执行之前这些标签们添加到标签集之中。如果语句执行的结果是`(break, V, L)`，其中L等于标识符，则产生的结果是`(normal, V, empty)`. //zxx: 标识符实际就是指变量名、函数名、数组名等. 此处我是看不懂的，因此无法具体解释
>
> 执行标记语句(LabelledStatement)之前，其包含的语句被认为具有一个空的标签集，除非他是一个迭代语句或switch语句。在这种情况下,它被视为拥有一个单一元素组成的空标签集。

上面这点东西折腾了个把小时都没搞清楚要说的是什么，貌似也不能很好解释这里的结果，投入与产出比太低，郁闷！只怪功力不够，或许过两年再过来看，就很轻松了。

后又搜索了下，发现~~MDC~~MDN的解释要通俗的多：
所谓标记语句，语法如下：

```
label :
   statement
```

就是：

```
标签 :
   语句
```

其中`label`可以是JavaScript中任意非保留关键字的标识符；`statement`中`break`可以用在任何标记语句中，`continue`用在循环标记语句中。

~~MDC~~MDN建议不使用labels
`labels`在JavaScript中并不常用，因为其降低了可读性以及易理解性，尽可能避免使用`labels`标记，根据不同情况，选择函数调用或者抛出错误。

下面为一个典型的标记语句的例子：

```
var i, j;

loop1:
```

```
for (i = 0; i < 3; i++) {      // 首页语句标记是 "loop1"
   loop2:
   for (j = 0; j < 3; j++) {   //第二个语句标记为 "loop2"
      if (i == 1 && j == 1) {
         continue loop1;
      } else {
         console.log("i = " + i + ", j = " + j);
      }
   }
}

// 输出是:
//   "i = 0, j = 0"
//   "i = 0, j = 1"
//   "i = 0, j = 2"
//   "i = 1, j = 0"
//   "i = 2, j = 0"
//   "i = 2, j = 1"
//   "i = 2, j = 2"
// 注意是如何不输出 "i = 1, j = 1"和"i = 1, j = 2"的
```

扯了这么多，回到问题本身。

根据规范，`a: b: c: ... g:`在语句执行之前会归到一个标签集中，为一个集合。因此，走个极端的话，我们可以这么理解：
`abcdefg: 1, 2, 3, 4, 5`. `1, 2, 3, 4, 5`一开始有说明的逗号多重运算啦——返回最后一个值，因此，本题就类似于提问：

```
abcdefg: 5 // 返回的是?
```

10.
```
{a: 1, b: 2}[["b"]] //返回的是?
```

FireBug控制台显示：SyntaxError: invalid label.

得知是这个结果的时候，我的懵懂硕大的眼睛立马无语成了一条缝，￣﹏￣

第6题中，我们已经讲过，JavaScript没有块作用域，块本身几乎是个酱油，因此这里的测试题实际等同于：

```
a: 1, b: 2; [["b"]]
```

结果显示：SyntaxError: invalid label.

现在的疑问是，为何`a: 1, b: 2`会报错？微博提问……

30分钟后，@紫云妃给出了这样的解释：

> 逗号运算符右侧必须是个表达式,不能是非表达式的语句,这个例子中`a: 1, b: 2`的右侧的`b: 2`是一个LabelledStatement, 不是表达式。

我觉得是相当靠谱的回答。

11. `"b" + 45 //返回的是?`

尼玛是不是感动地眼泪哗啦的。这种感觉就像是被后妈饿了个把月，在今天这个阳光明媚的日子赏了一块红烧肉，一下子感动地眼泪喷泉般涌出来。

结果是：`"b45"`.

字符串+数值=字符串。

12. `{a:{b:2}} //返回的是?`

结果是：`2`.

自从翻越了珠穆朗玛，才知道原来佘山就是个小土包。这个问题已经是小菜了，JavaScript没有块作用域，因此，`{a:{b:2}}`近乎于`a:b:2`，想起`a,b,c,...g`的例子没有，显然，这里返回值是`2`.

13. `(function(){}()) //返回的是?`

结果是：`undefined`.

函数如果是空，或者没有指定返回值，则返回的是`undefined`.

关于此题，我在眼睛微博上抛出了这么个问题：*(function(){}())和(function(){})()这两种写法有什么区别呢?*

1个小时过去了，以下几个评论比较见血：

> @川川哥哥勤劳致富 效果是一样的,周爱民那本书上提到了语法解读上的区别,不过感觉讲得不是很清晰
>
> @貘吃馍香 括号 匿名函数表达式执行 括号 跟 括号 函数表达式定义 括号 => 返回匿名函数句柄执行 的区别……
>
> @_Franky 抽象语法树 和 运行时 无差异. 除非这个解释引擎脑残了. 因为分组运算符 “()” 在生成语法树的过程中被消除了. http://t.cn/zWTxgKa 这篇我有顺道提到过这个问题

14. `[1,2,3,4,5][0..toString.length] //返回的是?`

结果还是：`2`.

本题涉及的知识点挺多的。

① 首先是诡异的连续两个点`..`. 我们在写CSS的时候，常常会有这样的写法：

```
font-size: .9em;
color: rgba(0, 0, 0, .35);
```

点前面如果是`0`则可以自动缺省。这条规则似乎在JavaScript中也是适用的，比方说`.9`，返回的

就是 `0.9`.

因此，这里 `0..toString` 实际上等同于 `0.0.toString`. 下面有个疑问了，为什么不直接 `0.toString` 而是 `0.0.toString` 要多搞出一个 `0` 呢？据说是这样子的，数值后面的点 `.` 有两种作用，一种就是当作小数点适用，另外就是用做字段访问。显然， `0.toString` 这里的点会被当作小数点，于是，直接报错了！写成 `(0).toString` 可以避免此问题。再来看看 `0.0.toString`，显然，再做解析的时候， `0.0` 中的点当作了小数点，显然， `0.0` 后面的那个点没有任何理由再被当作小数点了（小数不可能有两个小数点的），而是表示字段访问。于是，万事大吉，阳光明媚（下图版权所有）！

② 下面看看 `toString`，`toString` 为JS内置方法，ECMAScript规范上转换规则如下：

输入类型 结果
Undefined “undefined”
Null “null”
Boolean 如果参数是 true，那么结果为 “true”。 如果参数是 false，那么结果为 “false”。
Number 结果等于输入的参数（不转换）。
String 参见下文的文法和注释。
Object 应用下列步骤： 1. 调用 `ToPrimitive`（*输入参数*，暗示 *字符串类型*）。 2. 调用 ToString(Result(1))。 3. 返回 Result(2)。

从上表可以看出，数值使用toString方法是有参数的。

在JavaScript中，这个参数仅 `1` 个，称之为“基”，决定了数值转换的进制大小。默认情况下是10进制。例如：

```
(10).toString(); // "10"
```

我们还可以设置“基”为2进制，8进制或16进制，则结果大变样：

```
(10).toString(2); // "1010"
(10).toString(8); // "12"
(10).toString(16); // "a"
```

在实现颜色值16进制与rgb转化的时候，这些基数参数就很管用。

③ 最后了解下函数的length与参数个数，看这个： `(function(){}).length` 的结果是 `0`. 为何？

每个 `function` 函数似乎都有一个不可写的 `length` 属性，对应这个函数的参数个数。

```
var fun = function() {};
fun.length = 3;

fun.length; // 0
```

④ 现在，回到我们的问题，事情就豁然开朗了：`[0..toString.length]` 实际上就是数值(`0.0`)应用toString方法的参数个数是多少？根据上面描述，数值使用 `toString` 转换参数个数为 `1` 。于是，本题结果就是 `[1,2,3,4,5][1]` => `2` .

15.
```
({} + 'b' > {} + 'a') //返回的是?
```

结果是： `true` .

需要注意最外面的括号。如果没有最外面的括号，则 `{}` 则几乎无意义，但是这里，作为常规用法， `{}` 表示原生对象。因此，这里的比较实际上就是比较 `("[object Object]b" > "[object Object]a")` , 因此返回的是 `true` .

说点题外的，如果最外部没有括号， `{} + 'b'` 返回的是 `NaN` . 于是 `{} + 'b' > {} + 'a'` 变成了比较 `NaN > NaN` , 结果为 `false` .

16.
```
Number.prototype.x = function(){ return this === 123; };
(123).x(); //返回的是?
```

结果是： `false` .

我们这里使用了严格相等。实际上 `this` 和 `123` 属于不同的类型。

```
typeof this === “object”
typeof 123 === “number”
```

因此，结果为 `false` . 如果我们把题目修改成弱等于，则返回结果就是 `true` 了，见下截图：

```
Number.prototype.x = function(){ return this == 123; };
(123).x(); //返回的是?
```

17.
```
Array(2).join() //返回的是?
```

结果是： `","`

`Array(2)` 的返回值是 `[undefined, undefined]` ，因此，其使用join连接之后，就是个逗号 `","` （数组join为指定连接符时候使用默认的逗号 `","` ）。

这里我要抛出一个微博上没有满意解答的问题：`Array(2)`和`new Array(2)`的区别在什么地方呢？ 求指点迷津！

18.
```
vars: var vars = vars; //返回的是?
```

结果是：`undefined`.

现在看此题就简单多了，标记语句，返回值就是`var vars = vars`的返回值`undefined`.

`var vars = vars`并不会报`vars`为定义的错误是在于JS的置顶解析，其实`var vars = vars`的运作是这样子的：

```
var vars;
vars = vars;
```

19.
```
{ foo = 123 } //返回的是?
```

结果是：`123`.

花括号不在括号中、if语句或者循环中，属于非常规酱油用法，形同虚设，问题等同于`foo = 123`的返回值是?

20.
```
x = 1; (function(){return x; var x = 2;}()) //返回的是?
```

结果是：`undefined`.

此处考察的是JavaScript的“预解析(hoisting)”，也称“置顶解析”，我之前有翻译过相关文章。

所谓“预解析”指的是在当前的作用域内，无论在哪里变量声明，在幕后，其都在顶部被“预解析”了。因此，本题的实际“解析”是：

```
x = 1; (function(){var x; // 此时x为undefined
return x;  x= 2;}())
```

因此结果是`undefined`.

21.
```
delete [].length; //返回的是?
```

结果为`false`.

`delete`用来删除对象属性，成功删除返回`true`，如果对方防御很强删不动则返回`false`.

数组中的`length`属性是不可删除的，因此这里返回的是`false`.

`delete`相关的知识点是很多的，比说法`window.x = 1`这里的`x`可以被`delete`. `var x =1`这里的`x`就不能被`delete`. 图灵社区有篇不错的译文：“理解delete”，推荐阅读。

22.
```
RegExp.prototype.toString = function() {return this.source};
/3/-/2/; //返回的是?
```

结果是：`1`.

正则表达式有如下一些属性：`source`，`global`，`ignoreCase`，`multiline`，`lastIndex`. 其中：`source`属性为构成正则表达式Pattern的字符串；`global`属性是一Boolean值，表示正则表达式flags是否有`"g"`；`ignoreCase`属性是一Boolean值，表示正则表达式flags是否有`"i"`；`multiline`属性是一Boolean值，表示正则表达式flags是否有`"m"`；`lastIndex`属性指定从何处开始下次匹配的一个字符串类型的位置索引，当需要时该值会转换为一个整型数。

`RegExp.prototype.toString`扩展改变了默认的`toString`方法，当正则表达式需要应用`toString`方法进行字符串转换的时候，返回的就是正则表达式的`source`属性值。

例如：`/^\.dd\d+$/ + ""`的结果就是`"^\.dd\d+$"`.

于是，/3/-/2/实际上等同于`"3"-"2"`，因此结果就是`1`.

如果没有`toString`方法的重置，/3/-/2/实际上等同于`"/3/"-"/2/"`，因此结果就是`NaN`.

23.
```
{break;4;} //返回的是?
```

结果是报如下错误：SyntaxError: unlabeled `break` must be inside loop or `switch`.

意思是——解析错误：未标记的`break`必须在循环或`switch`中。

对照错误，我们加个标记，使之成为标记语句，就不会出错了。类似下面：

```
foo: { break foo; 4;}
```

24.
```
'foo' == new function(){ return String('foo'); }; //返回的是?
```

结果为：`false`.

由于这里是`==`，`'foo'`又是正宗的字符串，因此，后面的`new...`需要转换成字符串。

```
new function(){ return String('foo'); } + ""; // "[object Object]"
```

显然，`'foo' == "[object Object]"`为`false`.

本题如果稍作一点修改，则结果完全不一样：

```
'foo' == new function(){ return new String('foo'); }; //返回的是?
```

结果为：`true`.

为何?

在怿飞（圆心）“详解new function(){}和function(){}()”一文中如下的解释(//zxx: 08年初的文章啊，那个时候我处在被女神嫌弃，发奋图强的阶段。)：

> 只要`new`表达式之后的`constructor`返回（`return`）一个引用对象（数组，对象，函数等），都将覆盖new创建的匿名对象，如果返回（`return`）一个原始类型（无`return`时其实为`return`原始类型`undefined`），那么就返回`new`创建的匿名对象。

什么意思呢？众所周知，JavaScript中有`5`种基本类型（Undefined类型、Null类型、Boolean类型、Number类型、String类型），如果`new`后面的`function` `return`的是这5中基本类型之一，`new`会认为你是纯屌丝，不理你，还是返回自己创建的匿名对象；当然，如果返回数组啊、函数、对象这类高富帅，`new`立马变龟孙子了，返回的就是这些高富帅。

由于`String("foo")`是字符串，而`new String("foo")`是对象。因此，前者返回的是匿名函数对象——显然不等于`"foo"`；后者就是`new String("foo")`对象，加上`"foo" == new String("foo")`，于是，结果为`true`.

25.
```
'foo'.split('') + [] //返回的是?
```

结果是：`"f,o,o"`.

记住，`数组加数组，字符成老母`。`'foo'.split('')`结果为数组`["f", "o", "o"]`，其变身字符串就是`"f,o,o"`跟后面的`[]`也就是`""`相加，就是最终的`"f,o,o"`了。

下面考考你，

```
[1, 2] + [3, 4] //返回的是?
```

是不是`"1,2,3,4"`？恭喜你，🤔，回答.....错误！🥀

这又是整哪样啊！哥，你只是稍微粗心了点。`[1, 2]`变成字符串是`"1,2"`，`[3, 4]`变成字符串是`"3,4"`，因此两者相加是`"1,23,4"`而不是`"1,2,3,4"`.

空数组实际上是个很有意思的东西。

```
[] == 0 //true
!0 // true
![] // false
```

纯属题外东西，就不扩展了。

三、公子来时雪花飘，公子离去知了叫

不容易啊，终于看到结尾了，从五一前写到五一后。

正如开始提到了，纯属个人理解，虽观点基本都多方考证，难免还有不准确的地方，欢迎有相关研究的同行指正。

油枯灯尽，想不出什么油麦的话语了，就这样吧。我个人是学到很多东西，希望对您的学习也能有所帮助。

文章的一些提问欢迎回答，您的回答会对其他过来学习的人提供很多的帮助，人的价值不正是在于留下什么吗？

末了，附上ECMAScript5.1中文版链接：http://ecmascript.cn/

（本篇完）

« ES5中新增的Array方法详细说明　　小tip: DOM appendHTML实现及insertAdjacentHTML »

猜你喜欢

- 翻译-高质量JavaScript代码书写基本要点
- 翻译 - 解释JavaScript的“预解析(置顶解析)”
- 翻编-JavaScript有关的10个怪癖和秘密
- 翻译：ECMAScript 5.1简介
- HTML5 DOM元素类名相关操作API classList简介
- 我对原型对象中this的一个懵懂错误认识
- ES5中新增的Array方法详细说明
- Ajax Upload多文件上传插件翻译及中文演示
- JavaScript实现http地址自动检测并添加URL链接
- 翻译-你必须知道的28个HTML5特征、窍门和技术
- IE6下png背景不透明问题的综合拓展

分享到：1

标签： javascript, toString, 作用域, 对象, 数组, 标记语句, 正则表达式, 泛型, 类型转换, 逗号运算符, 面试题, 预解析

发表评论（目前28条评论）

名称 (必须)

邮件地址(不会被公开) (必须)

网站

提交评论

1. **wl22说道：**
2017年03月6日 09:55

{ foo: 1 }输出不再是是1，是一个对象

回复

2. **hymin说道：**
2016年08月9日 11:13

来来回回看了三四遍
计划过两周再看一遍

回复

3. **carl说道：**
2016年07月26日 19:33

犀牛6书和高程3里面这知识点都说过

回复

4. **xdsnet说道：**
2016年05月12日 09:56

10题改为 ({a:1,b:2})[[“b”]] 就会得到2的结果啦

回复

5. **xdsnet说道：**
2016年05月12日 09:42

6题在firefox中返回是 0 而不是 [0] !

回复

6. **KilArmd说道：**
2015年01月21日 00:29

我也认为Array(2)和new Array(2)是相同的，

Array类型不同于基本类型，基本类型包装对象的构造函数，作为构造函数调用时，返回的是一个包装对象，作为函数调用时，返回的是一个基本类型的值。

String.prototype.isPrototypeOf(String(”)); //false
String.prototype.isPrototypeOf(new String(”)); //true

而数组就不一样了，其本身就是对象。Array函数通不通过new调用，返回的都是一个对象，且具有相同的属性的方法：行为都继承自Array类，都能扩展属于实例的属性。

所以，鄙人认为，Array函数是通过类似的机制运行的：

```
var F=function(x){
if (this && this!==window){
this.x=x;
}else{
return new arguments.callee(x);
}
}
```

回复

7. **vark说道：**
2014年10月17日 17:29

[] == 0 //true

————————

这里的[]如果用Boolean([])强制转换之后是true，然后 == 0 应该返回false啊，为什么会返回true？求指教！

回复

张 鑫旭说道：
2014年10月18日 23:09

@vark 你好，不是Boolean转化，因为是与数字比较，因此是数值转化。[] → [] * 1

回复

8. 扯淡专家说道：
2014年08月27日 17:42

必须得匿了，鄙人表示菜性十足。

回复

9. 华续说道：
2014年07月8日 10:48

关于Array(2)和new Array(2) 的区别在Javascript权威指南有解释，相信博主现在已经知道了。区别在于采用String()、Array()、Object()等构造函数会显示的创建包装对象。
var men = ‘huaxu’;
men.age = 18;
console.log(men.age);
第二步的过程分解出来：
1.创建一个临时对象;
2.为这个临时对象添加属性age;
3.销毁这个临时对象;
men.age属性是undefined.
根本上：
var men = new String(‘huaxu’); typeof men 是 ‘object’;
var men = ‘huaxu’; typeof men 是 ‘string’.
和你的第十六题有些像。

回复

10. js观众说道：
2014年06月13日 09:49

看文章前，觉得水平很次；看完了，觉得自己水平简直太次了。多多学习，有效提升。

回复

11. lihuabest说道：
2013年06月28日 22:11

果然够坑爹，js之路还很长呢

回复

12. NetPuter说道：
2013年06月16日 11:16

我觉得第三题解释得不对。

AND 和 OR 是同等优先级的运算符，同时出现时，按从左到右的顺序。

所以 `1 || “foo” && 0` 的结果，也是 1 ，但却因为 1 OR Any 结果都是 1 。

还可以通过以下代码验证：

var i = 10;
++i || –i && –i; // i = 11

回复

NetPuter说道：
2013年06月16日 11:32

好吧，看到后文提到的美拓的那篇文章，用这个例子就把我说的击败了……

```
3 || 0 && 2 // 3
( 3 || 0 ) && 2 // 2
3 || ( 0 && 2 ) // 3
```

回复

13. **moo**说道：
2013年06月13日 18:15

http://alpteam.pl/IT/js_quiz/

不是喜欢被虐吗？ 请点击

回复

14. **wyljkz**说道：
2013年05月31日 12:07

题10：
({a: 1, b: 2}[[“b”]]) //2
+{a: 1, b: 2}[[“b”]] //2
0, {a: 1, b: 2}[[“b”]] //2

回复

15. **lugesot**说道：
2013年05月30日 09:51

foo = 123 改成 var foo =123; 结果为什么是undefined？然后再打印foo，就显示123了。和变量置顶有关系吗?

回复

16. **lugesot**说道：
2013年05月29日 18:24

第5题，你应该举出反例：
x = { shift: [].shift };
x.length
返回的结果是undefined

回复

17. @陶子吃鱼儿说道：
2013年05月28日 19:32

学习了！一遍下来，正确13题。前路漫漫呀！

回复

18. 昔影说道：
2013年05月12日 14:58

打击啊。。菜到不行

回复

19. 阿良说道：
2013年05月12日 12:15

Array(2)和new Array(2)应该是一样的，还有String…之类的，估计都做了类似这样的处理

```
X = function(a){
//…….
if(!(this instanceof X)){
return new X(a)
}
}
```

没错，就是这样：

当调用 Array 函数，采用如下步骤：

创建并返回一个新函数对象，它仿佛是用相同参数给标准内置构造器 Array 用一个 new 表达式创建的 (15.4.2)。

http://ecmascript.cn/#334

微博不能评论。。。

回复

> 张 鑫旭说道：
> 2013年05月13日 10:27
>
> @阿良 很精彩的回复！
>
> 回复
>
> > 阿良说道：
> > 2013年07月23日 11:45
> >
> > 我发现new String(”)和String(”)的区别了，前者可以扩展，后者不能扩展
> > var s = String(”);
> > s.o = 1;
> > console.log(s.o); //undefined
> >
> > var S = new String(”);
> > s.o = 1;
> > console.log(s.o); //1
> >
> > 跟这类似的还有Number 和 Boolean
> >
> > 回复

20. **imyuan说道：**
2013年05月8日 14:20

当加上字符串的引号后变成15/25,稍稍好受了一点.

回复

21. **imyuan说道：**
2013年05月8日 14:17

12/25,哎.还得继续

回复

22. **marinesky说道：**
2013年05月8日 11:12

我咋觉得这些写法用到的很少,而且有点不放心

回复

23. **justjavac说道：**
2013年05月8日 10:34

收藏了，慢慢消化。

回复

24. 玉面小飞鱼说道：
2013年05月8日 09:17

做完了好受打击，js学习之路还很漫长，这篇文章收了，仔细体会再来提问。

回复

最新文章

» 常见的CSS图形绘制合集
» 粉丝群第1期CSS小测点评与答疑
» 分享三个纯CSS实现26个英文字母的案例
» 小tips: 纯CSS实现打字动画效果
» CSS/CSS3 box-decoration-break属性简介
» CSS :placeholder-shown伪类实现Material Design占位符交互效果
» 从天猫某活动视频不必要的3次请求说起
» CSS vector-effect与SVG stroke描边缩放
» CSS ::backdrop伪元素是干嘛用的?
» 周知：CSS -webkit-伪元素选择器不再导致整行无效

今日热门

» 常见的CSS图形绘制合集 (193)
» 未来必热：SVG Sprite技术介绍 (120)
» 粉丝群第1期CSS小测点评与答疑 (115)
» HTML5终极备忘大全（图片版+文字版） (93)
» 让所有浏览器支持HTML5 video视频标签 (86)
» Selectivizr-让IE6~8支持CSS3伪类和属性选择器 (82)
» CSS3下的147个颜色名称及对应颜色值 (80)
» 视区相关单位vw, vh..简介以及可实际应用场景 (77)
» 写给自己看的display: flex布局教程 (76)
» 小tips: 纯CSS实现打字动画效果 (76)

今年热议

» 《CSS世界》女主角诚寻靠谱一起奋斗之人 (76)
» 不借助Echarts等图形框架原生JS快速实现折线图效果 (64)
» 看，for..in和for..of在那里吵架！ (60)
» 是时候好好安利下LuLu UI框架了！ (47)
» 原来浏览器原生支持JS Base64编码解码 (35)
» 妙法攻略：渐变虚框及边框滚动动画的纯CSS实现 (33)
» 炫酷H5中序列图片视频化播放的高性能实现 (31)
» CSS scroll-behavior和JS scrollIntoView让页面滚动平滑 (30)
» windows系统下批量删除OS X系统.DS_Store文件 (26)
» 写给自己看的display: flex布局教程 (26)

猜你喜欢

- 翻译-高质量JavaScript代码书写基本要点
- 翻译 - 解释JavaScript的“预解析(置顶解析)”
- 翻编-JavaScript有关的10个怪癖和秘密

- 翻译：ECMAScript 5.1简介
- HTML5 DOM元素类名相关操作API classList简介
- 我对原型对象中this的一个懵懂错误认识
- ES5中新增的Array方法详细说明
- Ajax Upload多文件上传插件翻译及中文演示
- JavaScript实现http地址自动检测并添加URL链接
- 翻译-你必须知道的28个HTML5特征、窍门和技术
- IE6下png背景不透明问题的综合拓展