**Javascript模式读书笔记**

**一javascript基本概念**

**1面向对象**

**（1）** javascript是一门面向对象的语言，看到的任何一段代码都有可能是一个对象，只有五种基本类型不是对象，数值类型，字符串类型，布尔类型，空类型，未定义类型，其中前三个类型有对应的以基本类型封装形式体现的对象表示。数值类型，字符串类型，布尔类型的值可以通过程序员或者位于幕后的javascript解析器来实现向对象的转换。

（2）函数实际上也是对象，有属性和方法。

（3）对象主要有两种类型：

原生的(Native)

在ECMAjavascript中有详细描述。

主机的（Host）

在主机环境中定义的（例如浏览器环境）。

2没有类

1. 在javascript中可以在需要的时候创建一个空对象，然后开始为该对象添加感兴趣的成员变量，可以为该对象添加基本类型，函数和其他对象来作为该对象的属性。
2. 在gang of four中一条通用规则是：尽量多使用对象的组合，而不是类的继承。
3. 原型
4. 可以使用多种方法来实现继承，这里通常使用继承，原型是一个对象，并且创建的每一个都会自动获取一个prototypes属性，该属性指向一个新的空对象。该对象几乎等同于采用对象字面量或object创建的对象，区别在于它的constructor属性指向所创建的函数，而不是指向内置的object()函数。
5. 环境

javascript需要运行环境来执行，通常是在浏览器中执行的，但这不是唯一的运行环境。

5.JSLint

JSLint是代码检查工具，在默认设置中，JSLint希望用户最好在代码中使用strict模式。

* 基本技巧

1. 编写可维护的代码
2. 花时间重新学习和理解相应的问题，
3. 花时间理解当时用于解决相应问题的代码

软件开发人员通常读代码比写代码更耗时间，有些代码当天可运行，但要想成为一个成熟的应用项目，需要我们对代码重新检查，重新校正，重新调整。例如

#发现bug

#项目加入新的特性

#项目需要在新的环境中运行

#代码变换意图

#代码在新的框架和体系下需要完全重写，甚至是应用一种新的语言。

易维护的代码有如下特性：

#阅读性好

#具有一致性

#预见性好

#看起来如同一人编写

#有文档

本章接下来将阐述在开始写javascript脚本时要注意的关键。

**尽量少用全局变量**

**全局变量的问题**

网页经常会包含一些非页面开发人员编写的代码，譬如

#第三方javascript库

#来自于广告合作伙伴的脚本。

#来自第三方用户的跟踪与分析脚本的代码

#各种小工具，徽章（badges）和ann。

因此，与同一个页面上的其他脚本友好共存非常重要，要尽可能少的使用全局变量。如命名空间模式或自执行立即生效函数，但最重要的方法还是使用var声明变量。

**注意：另一个避免全局变量的原因来源于代码移植。如果你希望你的代码运行在不同的环境中（主机），使用全局变量就会非常危险。因为可能偶然的，不存在原环境（所以看起来是安全的）。但存在于其他环境的主机变量就被覆盖了。**

**变量覆盖式的副作用**

隐含全局变量与明确定义的全局变量有细微的不同，不同之处在于能否使用delete操作符撤销变量。

#使用var创建的全局变量（这类变量在函数外部创建）可以删除。

#不使用var创建的隐含全局变量（尽管它是在函数内部创建）可以删除。

这表明隐含全局变量严格来讲不是真正的变量，而是全局对象的属性。属性可以通过delete操作符删除，但变量不可以。

**(1)单一var模式**

只使用一个var在函数顶部进行变量声明是一种非常有用的模式。它的好处在于：

#提供一个单一的地址以查找到函数需要的所有全局变量。

#防止出现变量在定义前就被使用的逻辑错误

#帮助牢记要声明变量，以尽可能地使用全局变量

#更少的编码（无论是输入代码还是传输代码都更少了）。

**(2)零散变量的问题**

javascript允许在函数的任意地方声明多个变量，无论在哪里声明，效果都等同于在函数顶部进行声明。这就是所谓的“提升。当先使用变量再在函数后面声明变量时可能会导致逻辑错误。对于javascript而言，只要变量是在同一范围内（同一函数）里，就视为已经声明，哪怕是在变量声明前就使用。

**(3)for循环**

for循环经常用在遍历数组或类数组对象，如引数（arguments）和html容器（HTMLColltion）对象。通常for循环模式使用如下。

这种模式的问题在于每次循环迭代时都要访问数据的长度。这样会使代码变慢，特别是当myarray不是数据，而是HTML容器对象时。

HTML容器是DOM方法返回的对象，如：

#document.getElementsByName()

#document.getElementsByClassName()

#document.getElementsTagName()

还有很多其他HTML容器，它们在DOM标准以前就引入了，并一直使用至今。包括（除此之外还有很多）：

documents.images

页面上所有的IMG元素

documents.links

所有的A元素

documents.forms

所有的Forms

document.forms[0].elements

页面上第一个form内的所有字段。

**(4)for-in循环**

for-in循环应该用来遍历非数组对象。使用for-in循环也被称为枚举（enumeration).

**当遍历对象属性来过滤遇到原形链的属性时，使用hasOwnproperty()方法是非常重要的。**

**注意：严格来讲，不使用hasOwnProperty（）并没有错。依赖于具体任务和对代码的自信，你可以略过该方法并稍微加速循环执行的速度。但是当确定不了对象的内容（和原形链）时，最好还是加上hasOwnProperty()这样的安全检查。**

**不要增加内置的原型**

增加内置构造函数（例如Object（）。Array（）和Function（）等）的原型是很有诱惑的，但是这可能会影响可维护性，因为这种做法是更加不可预测的。此外，给原型添加的属性在没有使用hasOwnproperty()时可能会在循环中出现，这会导致一些混乱。

因此，最好的方法就是不要给内置的原型添加属性。

**(5)避免使用隐式类型转换**

为了避免隐式类型转换导致的混血不清，请在使用语句的时候使用===和！===操作符来对数值和类型进行比较：

**(6)避免使用eval（）**

new Function()和eval（）会影响到作用域链，而Function更多的类似于一个沙盒。无论在哪里执行Function，它都仅仅能看到全局作用域。因此对局部变量的影响较小。在接下来例子子中eval（）可以访问和修改它外部作用域的变量，然而function不行。

**(7)构造函数的首字母大写**

javascript没有类，但是可以通过new调用构造函数

因为构造函数仍然仅仅是一个函数，他看起来是一个函数名，它和构造函数或者普通函数的行为差不多。构造函数的首字母大写这样的命名方式就包含了以上含义。首字母小写的函数名和方法表明这些函数和方法不能使用new来调用

**(8)其他命名模式**

#使用下划线结尾来表明是私有变量，例如name\_和getElement\_().

#使用一个下划线前缀来标识受保护属性，使用两个下划线前缀来标识私有属性。

在Firefox中有一些属性，这些属性技术上不是JavaScript语言的一部分，它们采用两个下划线前缀和两个下划线后缀来命名，例如\_\_proto\_\_and\_\_parent\_\_.

**(9)编写API文档**

**生成API文档的步骤如下：**

#编写特殊的格式的代码块。

#运行工具来解析代码和注释。

#发布工具解析的结果，大多数情况是采用HTML格式发布。

**小结：**

**#减少使用全局变量，最理想的情况是一个应用程序仅有一个全局变量。**

**#在每个函数中仅使用一个var变量声明，这有助于在一个地方查看所有变量，可以有效防范变量提升导致的错误。**

**#for循环，for-in循环，switch语句，evel（）是魔鬼和不要扩充内置的原型。**

**#遵循一些编码约定（一致性的空格，缩进，在可选的时候都坚持使用大括号和分号）和命名约定（针对构造函数，函数和变量的命名约定）。**

**三、字面量和构造函数**

**（1）对象字面量语法**

#将对象包装在大括号中（左大括号{和右大括号}）。

#对象中以逗号分隔符属性和方法。在名称-值（name-value）最后的尾随逗号是允许的，但是在IE浏览器中会产生错误，因此尽量不要使用它。

#用冒号分割属性名称和属性的值。

#当给变量赋值时，请不要忘记右大括号后的分号。

**（2)自定义构造函数**

**当new操作符调用构造函数时，函数内部将会发生如下情况：**

#创建一个空对象并且this变量引用了该对象，同时还继承了该函数的原型。

#属性和方法被加入到this引用的对象中。

#将创建的对象由this所引用，并且最后隐式的返回this（如果没有显式地返回其它对象）。

**(3)构造函数的返回值**

当用new操作创建对象时，构造函数总是返回一个对象，默认情况下返回的是this所引用的对象。

**(4)自调用构造函数**

在Es5的严格模式下并不支持arguments.callee属性，因此，最好限制在将来才使用该属性，并且在现在有代码中删除所找到的任何实例。

**(5)数组字面量**

**var a=new Array("itsy","bitsy");**

**var a=["itsy","bitsy"];**

**数组构造函数的特殊性**

当向Array()构造函数传递单个数字时，他并不会成为第一个数组元素的值。相反，它却设定了数组的长度。这意味着newArray（3）这个语句创建了一个长度为3的数组。并没有实际的元素，如果试图方位这个数组中的任何元素，将会获得undefined值。

为了避免您在运行时创建动态数组可能产生的潜在错误，坚持使用数组字面量表示法，程序将会更加安全。

**检查数组性质**

以数组作为操作数并使用typeof操作符，其结果将会返回"Object"。

**JSON**

JSON和文字对象之间唯一的语法差异在于，在JSON中，属性名称需要包装在引号中才能成为合法的JSON。而在对象字面量中，仅当属性名称不是有效的标识符时才会需要引号。此外，在JOSN字符串中，不能使用函数正则表达式字面量。

使用JSON

并不推荐盲目使用eval()对任意JSON字符串进行求值，其原因在于安全性的影响。如果使用JSON.parse()方法解析字符串，其安全性更好。

**正则表达式字面量**

正则表达式也是对象，可以用两种方法创建正则表达式：

#使用new RegExp()构造函数。

#使用正则表达式字面量

**正则表达式字面量语法**

正则表达式字面量表示法使用了斜杠来包装用于匹配的正则表达式模式。第二个斜杠之后，可以将该模式修改为不加引号的字母形式；

#g---全局匹配

#m--多行

i--大小写敏感的匹配。

使用new RegExp()的原因在于，某些场景中无法事先确定模式，而只能在运行时以字符串方式创建。

正则表达式字面量和构造函数之间的其中一个区别在于，字面量在解析时只有一次创建了一个对象。也就是说，如果你在一个循环中创建了相同的正则表达式，那么后面返回的对象与前面创建的对象相同，并且其所有的属性将被设置为第一次的值。

**注意：这种行为已经在ES中已经得到了改变，并且字面量会创建新的对象。在许多浏览器环境中，这种行为也得到了更正，因此，这种模式并非不可依赖。**

**最后需要说明的是，调用 RegExp（)时不使用new的行为是相同的。**

**基本值类型包装器**

包装对象包含了一些有用的属性和方法。例如，数字对象具有形如toFixed()和toExponential()的方法。字符串对象具有substing(),charAt().toLowerCase()等方法及length属性。与使用基本类型相比较而言，这些包装对象的方法使用起来非常方便，这也是用这种方法创建对象的一个很好的理由。但是这些方法在基本类型上也能够起作用。只要你调用这些方法，基本值类型就可以在后台被临时转换为一个对象，并且表现的如一个对象。

通常使用包装对象的原因之一是您有扩充值及持久保存状态的需要。这是由于基本值类型并不是对象，它们不可能扩充属性。

**错误对象**

javascript中有一些内置错误构造函数，比如Error（）SyntaxErroe(),TypeError()以及其他，这些处错误构造函数都带有throw语句。通过这些错误构造函数创建的错误对象具有夏列特性：

name用于创建对象的构造函数的名称属性，它可以是一般的Error或者更为专门的构造函数，比如RangeError。

message

当创建对象时传递给构造函数的字符串。

错误构造函数以函数的形式调用（不带new），其表现行为与构造函数（带new相同），并且返回同一个错误对象。

**小结**

**在本章中，已经学习到不同的字面量模式，与使用构造函数相比，这些都是更为简单的替代方法。本章主要讨论了下列主题：**

**#对象字面量表示法，这是一种优美的对象创建方式，它以包装在大括号中的逗号分隔键-值对的方式创建对象。**

**构造函数，主要包括内置构造函数（几乎总有一个更好且更短d字面量表示法）和自定义构造函数。**

**#一种可以确保自定义构造函数总是表现的像以new调用的方法。**

**#数组字面量表示法，方括号内以逗号分隔的之列表。**

**#JSON，一种包含对象和数组字面量的数据格式。**

**#正则表达式字面量**

**#其他内置构造函数以避免使用String(),Number().boolean()等，此外还讨论了多种不同的Error（）构造函数。**

**四，函数**

**背景**

javascript中的函数有两个主要特点使其显得比较特殊。第一特点在于函数是第一类对象，第二个特点在于它们可以提供作用域。

函数就是对象，其表现形式如下：

#函数可以分配给变量，可以将它们的引用复制到其他变量，可以被扩展，此外，除少数特殊情况外，函数还可以被删除。

#可以作为参数传递给其他函数，并且还可以由其他函数返回。

#函数可以有自己的属性和方法。

**库中的回调模式**

回调模式是一种简单又强大的模式，当设计一个库时他可以派上用场。进入软件库的代码应该尽可能的是通用和复用的代码，而回调可以帮助实现这种通用化。可以专注于核心功能并提供“挂钩”形式的回调函数，这将是您很容易的构建，扩展，以及自定义库方法

**立即执行函数**

这种模式也经常用于书签工具中，因为书签工具可以在任何网页上运行，并且保持全局命名空间的整洁 也是至关重要的。

即时对象初始化

这种模式使用带有init（）方法的对象，该方法在创建对象后将立即执行。init（）函数需要负责所有的初始化任务。

**两种方法：**

**（{...}).init()；**

**({...}.init(））；**

**注意：这种模式主要适用于一次性的任务，而且在init()完毕后也没有对该对象的访问。如果想在init（）完毕后保存对该对象的一个引用，可以通过在init（）尾部添加“return this；”语句实现该功能**

缓存函数结果也被称为备忘。

请注意在序列化过程中，对象的“标识”将会丢失。如果有两个不同的对象并且恰好都具有相同的属性，这两个对象将会共享统一个缓存条目。

ECMAScript5的严格模式中并不支持arguments.callee.

**小结**

**函数的两个重要特征**

**1.函数是第一类对象，可以作为带有属性和方法的值以及参数进行传递。**

**2.函数提供了局部作用域，而其他大括号并不能提供这种局部作用域，此外还需要记住的是，声明的局部变量可被提升到局部作用域的顶部。**

**在涵盖了函数的背景和语法之后iou，将学习到一些有用的模式，可以将它们分为以下几类：**

**1API模式，它们可以帮助您为函数提供更好且更整洁的接口。这些模式包括以下几个：**

**回调模式**

**将函数作为参数进行传递。**

**配置对象**

**有助于保持受到控制的函数的参数数量**

**返回函数**

**当一个函数的返回值时另一个函数时**

**Curry化**

**当新函数是基于现有函数，并加上部分参数列表创建时。**

**2。初始化模式，它们可以帮助您在不污染全局命名空间的情况下，使用临时变量以一种更加整洁，结构化的方式执行初始化以及设置任务。这些模式包括**

**即使函数**

**只要定义之后就立即执行。**

**即使对象初始化**

**匿名对象组织了初始化任务，提供了可被立即调用的方法。**

**初始化时分支**

**帮助分支代码在初始化代码执行过程中仅检测一次，这与以后在程序生命期内多次检测相反**

**3.性能模式，可以帮助加速代码运行，这些模式包括：**

**备忘模式**

**使用函数属性以便使得计算过的值无需再次计算。**

**自定义模式**

**以新的主体重写本身，以使得在第二次或以后调用时仅需执行更少的工作。**

**第六章 代码复用模式**

**类式继承模式  
1.默认模式**

**2.借用构造函数**

**3.借用和设置原型**

**4.共享原型**

**5.临时构造函数**