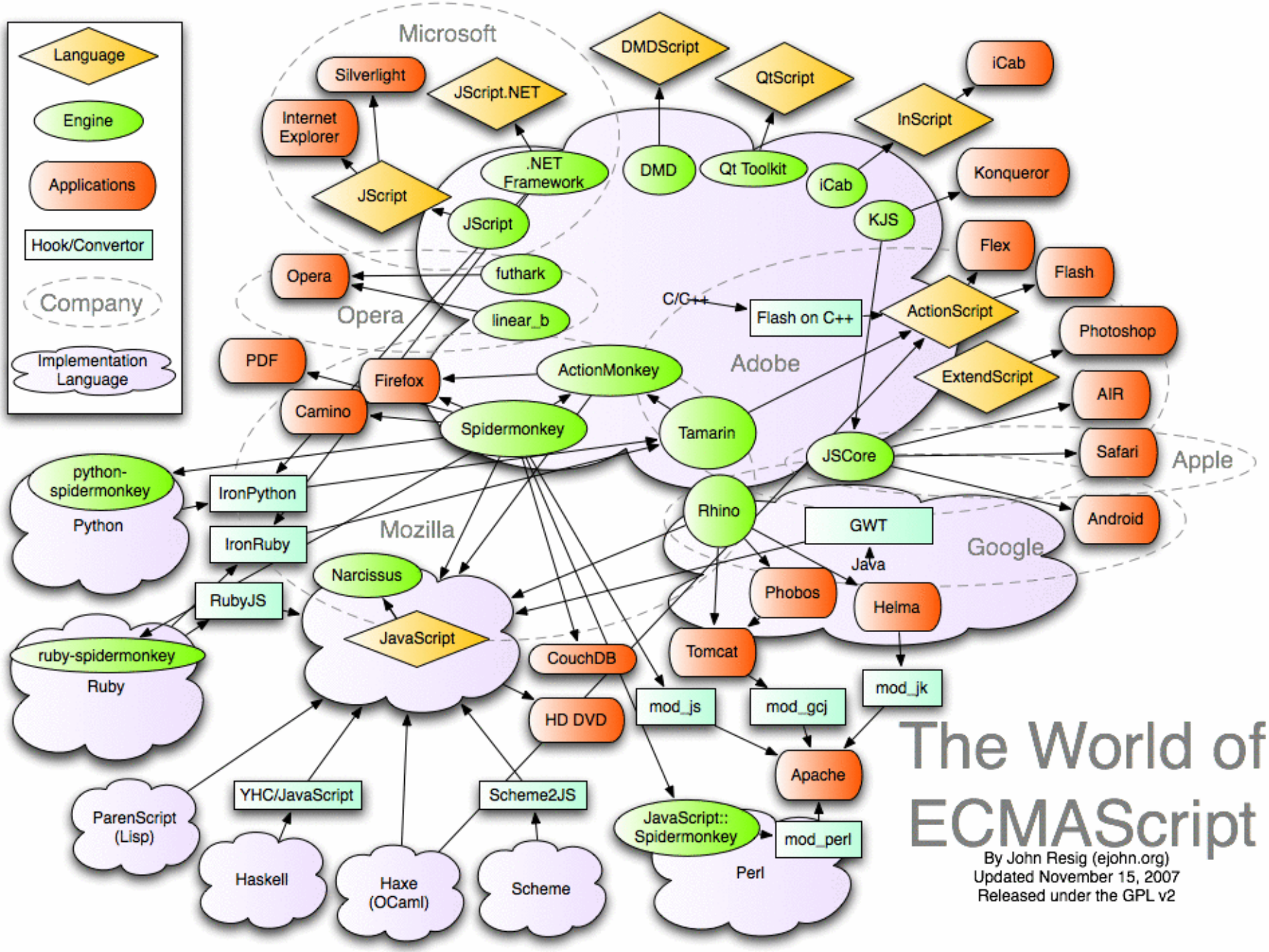
新业务新人培训课程计划

ECMAscript基础

分享人：地极


Language
Engine
Applications
Hook/Convertor
Company
Implementation Language
Microsoft
Silverlight
JScript.NET
Internet Explorer
JScript
.NET Framework
JScript
DMDScript
QtScript
DMD
Qt Toolkit
iCab
InScript
iCab
Konqueror
KJS
Opera
Opera
futhark
linear_b
PDF
Firefox
Camino
ActionMonkey
Spidermonkey
Tamarin
C/C++
Flash on C++
Adobe
Flex
Flash
ActionScript
Photoshop
ExtendScript
AIR
Safari
Apple
JSCore
Android
Rhino
GWT
Java
Google
Phobos
Helma
Tomcat
mod_jk
mod_gcj
mod_js
HD DVD
CouchDB
Apache
mod_perl
JavaScript:: Spidermonkey
Perl
python-spidermonkey
Python
IronPython
IronRuby
Mozilla
RubyJS
Narcissus
JavaScript
ruby-spidermonkey
Ruby
ParenScript (Lisp)
YHC/JavaScript
Scheme2JS
Haskell
Haxe (OCaml)
Scheme
The World of ECMAScript
By John Resig (ejohn.org)
Updated November 15, 2007
Released under the GPL v2

Javascript 历史

Javascript 实现

语法

变量

运算符

基本语句

函数

对象

Javascript 历史

1992年（起源）

- 早期美国Nombas公司开发的ScriptEase（最初命名为Cmm，即C--语言）

1995年

- Javascript的第一个正式版本，即Javascript 1.0是由Netscape公司和SUN公司联合开发，在Netscape Navigator 2.0中搭载发布

1996年

- Javascript 1.1在Netscape Navigator 3.0中发布

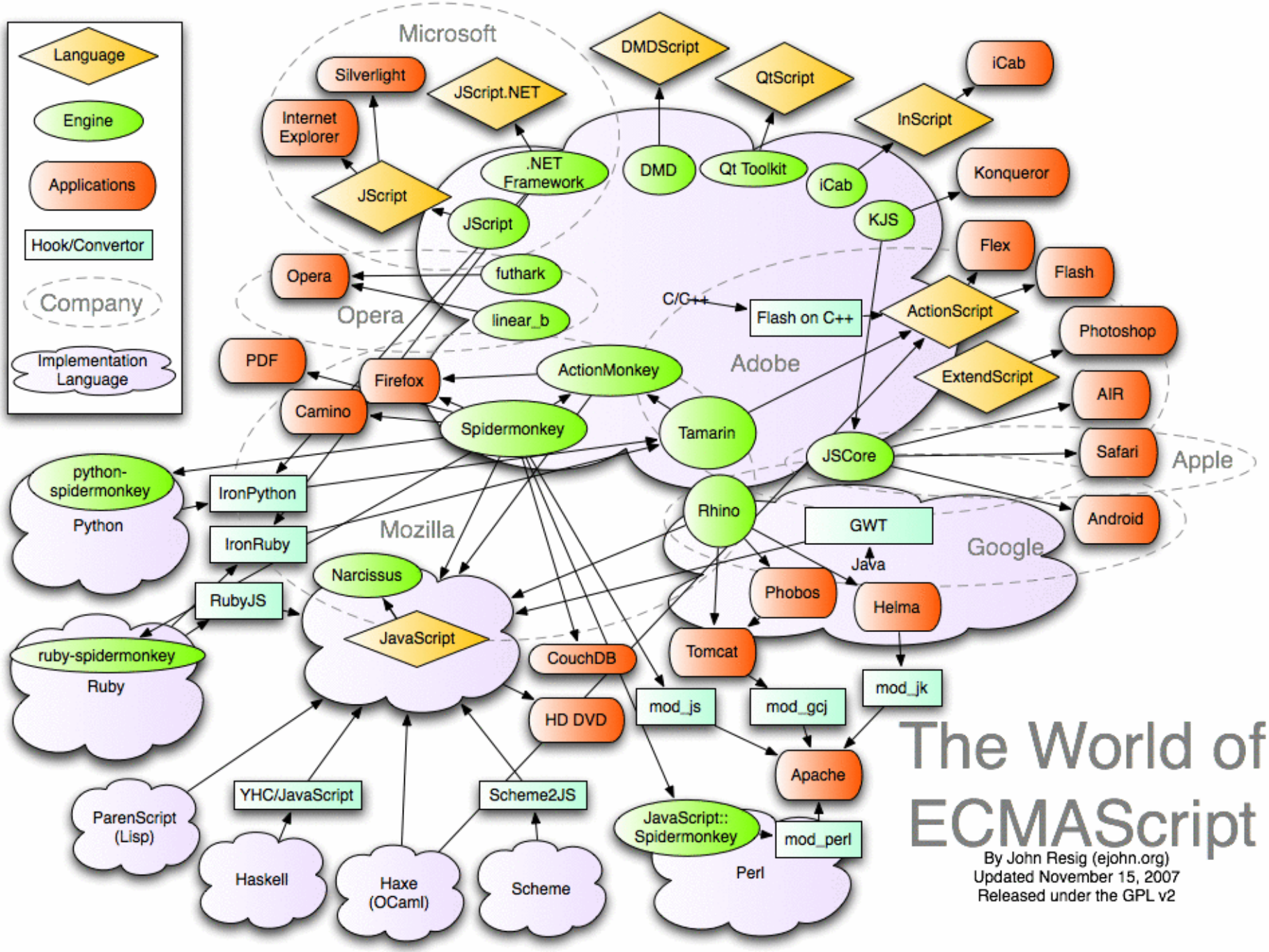
1997年

- Javascript 1.2搭载在1997年发布的Netscape Navigator 4.0中，但不符合在之后发布的ECMA-262规范第一版。

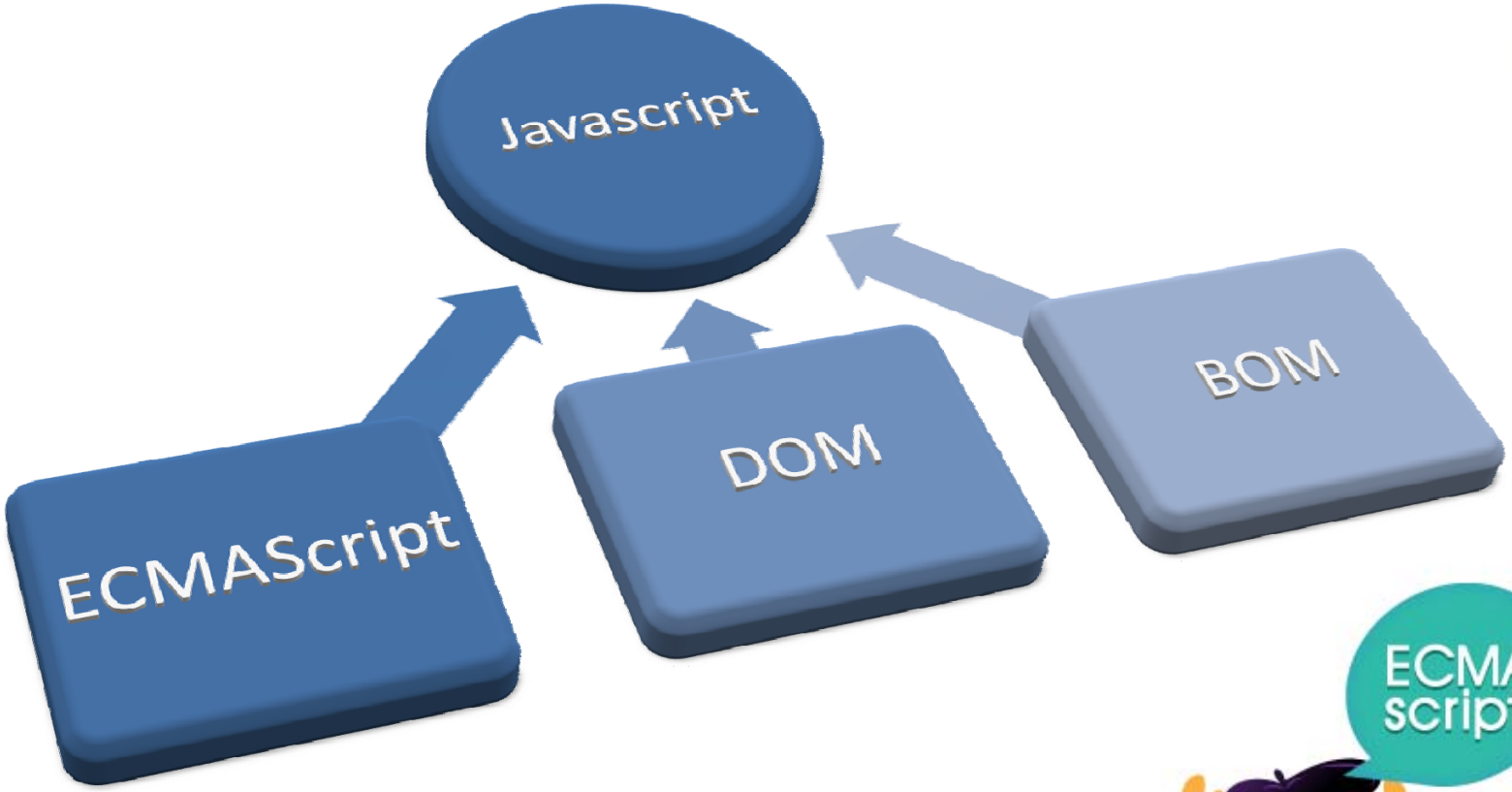
Javascript 历史

- Javascript 1.3
  - 搭载在随后发布的Netscape Navigator 4.06中，是Netscape第一个真正符合ECMA-262规范第一版的脚本语言实现，但这已经落后于Microsoft公司，因为在这之前微软公司发布的IE4.0中搭载的JScript3.0已经做到了这一点。
- Javascript 1.4
  - 搭载在Netscape Enterprise Server(Web Server)中发布，而没有出现在客户端的浏览器中。主要原因是当时Netscape公司决定全部重写浏览器的代码
- Javascript 1.5
  - 搭载在Netscape Navigator 6.0和Firefox 1.0中发布，这是一个相对稳定而全面的版本，完全符合ECMA-262规范第三版的要求。目前大多数浏览器支持的Javascript也是这个版本的实现。
- Javascript 1.6
  - 搭载在Firefox 1.5中发布，增加了对E4X规范的部分实现。
- Javascript 1.7
  - 搭载在Firefox 2.0中发布，仅引入了少量的新特性。
- Javascript 1.8
  - 搭载在Firefox 3.0中，引入了部分满足ECMAScript 4/JavaScript 2的特性。

Javascript ? ECMAScript


Language
Engine
Applications
Hook/Convertor
Company
Implementation Language
Microsoft
Silverlight
JScript.NET
Internet Explorer
JScript
.NET Framework
JScript
DMDScript
QtScript
DMD
Qt Toolkit
iCab
InScript
iCab
Konqueror
KJS
Opera
Opera
futhark
linear_b
PDF
Firefox
Camino
ActionMonkey
Spidermonkey
Tamarin
C/C++
Flash on C++
Adobe
Flex
Flash
ActionScript
Photoshop
ExtendScript
AIR
Safari
Apple
JSCore
Android
Rhino
GWT
Java
Google
Phobos
Helma
Tomcat
mod_jk
mod_gcj
mod_js
HD DVD
CouchDB
Apache
mod_perl
JavaScript:: Spidermonkey
Perl
python-spidermonkey
Python
IronPython
IronRuby
Mozilla
RubyJS
Narcissus
JavaScript
ruby-spidermonkey
Ruby
ParenScript (Lisp)
YHC/JavaScript
Scheme2JS
Haskell
Haxe (OCaml)
Scheme
The World of ECMAScript
By John Resig (ejohn.org)
Updated November 15, 2007
Released under the GPL v2

Javascript 实现



核心 - ECMAScript

- 与宿主环境（浏览器等）无关
- 描述了以下内容

语法 | 类型 | 语句 | 关键字

保留字 | 运算符 | 对象

ECMAScript 版本

1997年6月

- ECMA-262 第一版

1998年6月

- ECMA-262 第二版

强大的正则表达式，更好的文字链处理，新的控制指令，异常处理，错误定义更加明确，数输出的格式化及其它改变

1999年

- ECMA-262 第三版

DOM & BOM

- DOM
  - DOM（文档对象模型）是 HTML 和 XML 的应用程序接口（API）。DOM 将把整个页面规划成由节点层级构成的文档。
- BOM
  - BOM（浏览器对象模型），可以对浏览器窗口进行访问和操作。

ECMAScript 语法

- 区分大小写
  - ECMAScript中的一切，比如变量、函数名、操作符都区分大小写。
- 注释
  - //单行注释
  - /* javascript code */多行注释
- 语句
  - 语句以分号结尾，如果没有分号则由解析器确定结尾
  - {}表示代码块，if else语句在多行的时候才要求使用代码块(我们推荐任何时候都使用代码块)

ECMAScript 变量

- 使用var来声明变量，变量是弱类型的
  var test = "hi";
  var test = “hi”, test2 = “hello”;

- 命名变量
  - 第一个字符必须是字母、下划线（_）或美元符号（$）
  - 余下的字符可以是下划线、美元符号或任何字母或数字字符
  - 关键字不能做为变量名

- 变量声明不是必须的
  - 建议大家使用变量前都进行声明
  - 否则会污染全局变量

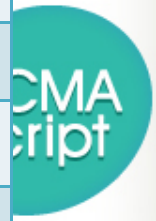
ECMAScript 关键字&保留字

- 关键字

break	else	new	var	case
finally	return	void	catch	for
switch	while	continue	function	this
with	default	if	throw	delete
in	try	do	instanceof	typeof

- 保留字：不推荐用作变量名和函数名

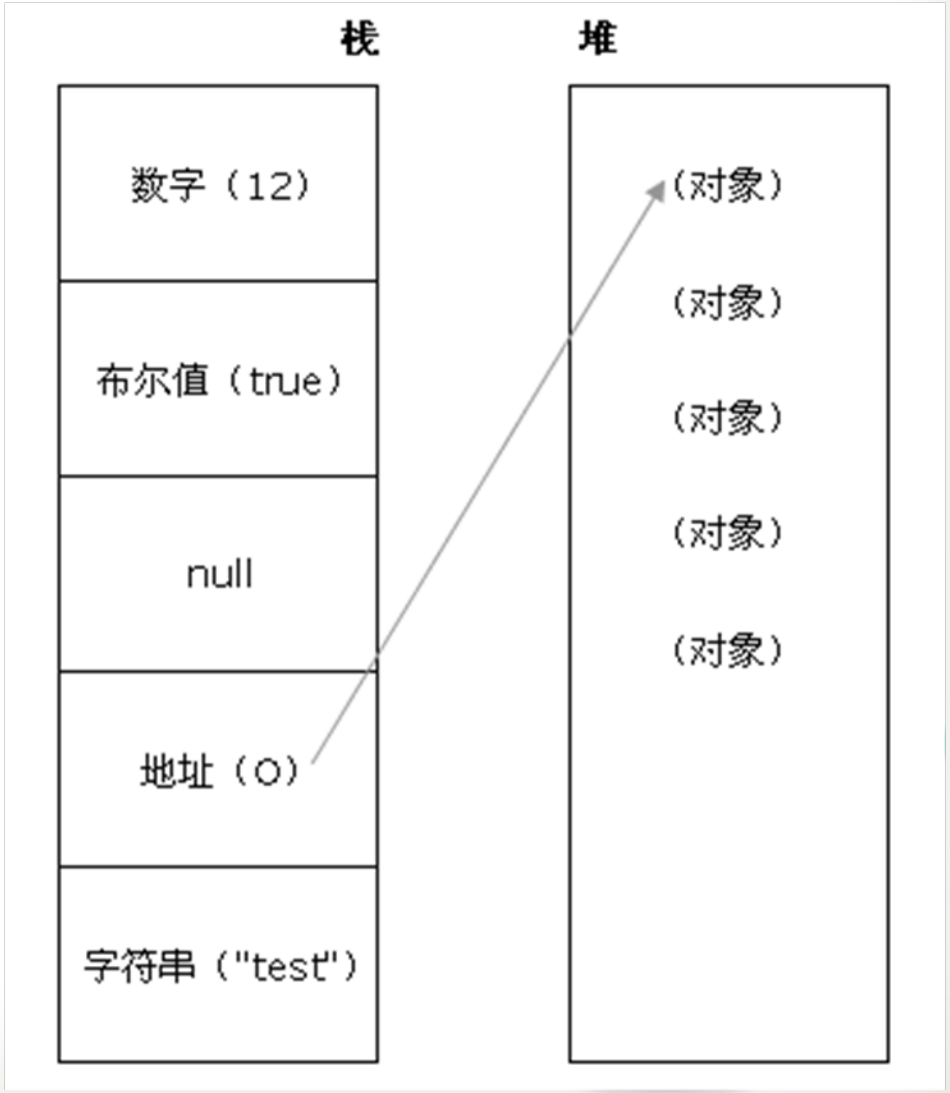
abstract	enum	synchronized	short	boolean	export
interface	static	implements	extends	long	super
char	native	debuggerv	int	class	float
package	throws	volvatile	goto	private	transient
final	byte	protected	const	double	import
public					



ECMAScript 数据类型

ECMAScript 原始值和引用值

- 原始值
  - 值直接存在变量中
- 引用值
  - 存储在变量处的值是一个指针（point），指向存储对象的内存处。



ECMAScript 原始值

- 原始值
  - Undefined
  - Null
  - Boolean
  - Number
  - String

- 使用typeof运算符获取原始值类型

- 如果变量是一种引用类型或Null类型的，返回“object”

```
var sTemp = "test string";
alert (typeof sTemp);     //输出 "string"
alert (typeof 86);     //输出 "number"
```

Undefined 类型

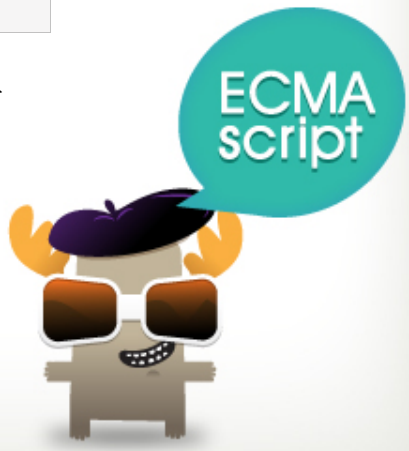
- Undefined 类型只有一个值，即 undefined。当声明的变量未初始化时，该变量的默认值是 undefined。

```
var oTemp;

alert(typeof oTemp);   //输出 "undefined"
alert(typeof oTemp2);  //输出 "undefined"
```

值 undefined 并不同于未定义的值。但是，typeof 运算符并不真正区分这两种值。

```
var oTemp;
alert(oTemp2 == undefined);
```



Null 类型

- 另一种只有一个值的类型是 Null，它只有一个专用值 null，即它的字面量。
- 值 undefined 实际上是从值 null 派生来的，因此 ECMAScript 把它们定义为相等的。

```
alert(null == undefined);   //输出 "true"
```

- 但它们的含义不同。undefined 是声明了变量但未对其初始化时赋予该变量的值，null 则用于表示尚未存在的对象。如果函数或方法要返回的是对象，那么找不到该对象时，返回的通常是 null。

Boolean 类型

- Boolean类型有两个值true和false

- 如果发生类型转换

  true -> 1 || “true”

  false -> 0 || “false”

Number 类型

- 这种类型既可以表示 32 位的整数，还可以表示 64 位的浮点数。
- 整数：可以用十进制、十六进制和八进制定义

```
var iNum = 0x1f;   //0x1f 等于十进制的 31
var iNum = 0xAB;   //0xAB 等于十进制的 171
```

- 浮点数：必须包括小数点和小数点后的一位数字（例如，用 1.0 而不是 1）

String 类型

- String 类[illegible]在于，它是唯一没有固定大[illegible]可以用字符串存储 0 或更多的 Unicode 字符

Unicode:用四位16进制数字表示一个字符

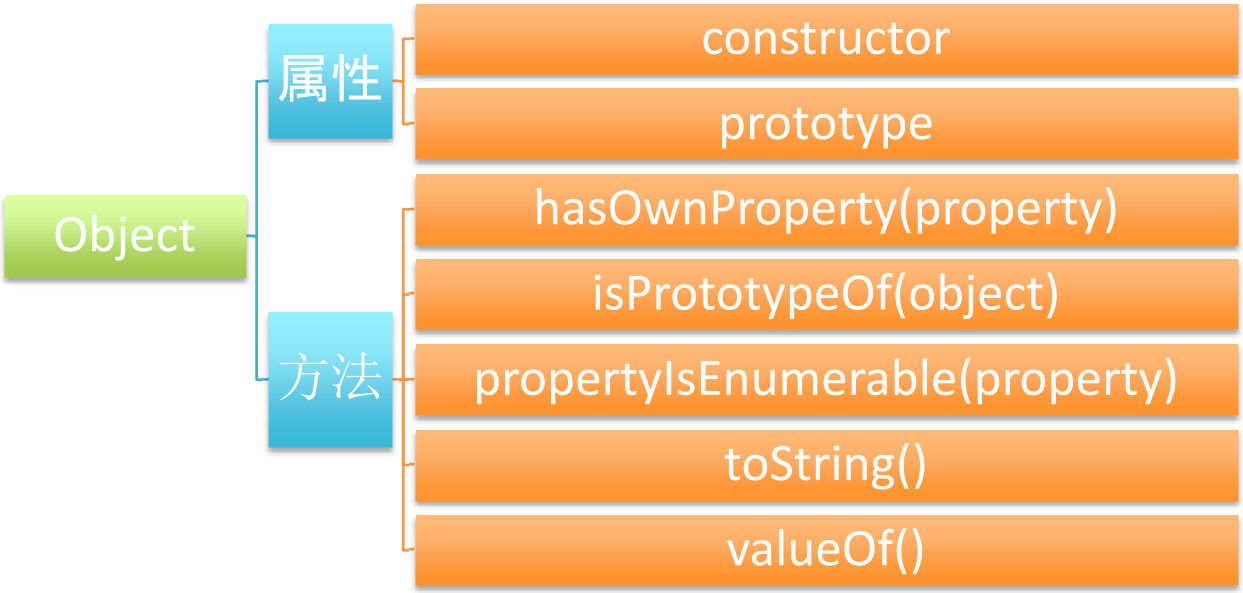

```
var sColor1 = "red";
var sColor2 = 'red';
```

ECMAScript 转义字符

字面量	含义
\n	换行
\t	制表符
\b	空格
\r	回车
\f	换页符
\\	反斜杠
\'	单引号
\"	双引号
\0*nnn*	八进制代码 *nnn* 表示的字符（*n* 是 0 到 7 中的一个八进制数字）
\x*nn*	十六进制代码 *nn* 表示的字符（*n* 是 0 到 F 中的一个十六进制数字）
\u*nnnn*	十六进制代码 *nnnn* 表示的 Unicode 字符（*n* 是 0 到 F 中的一个十六进制数字）

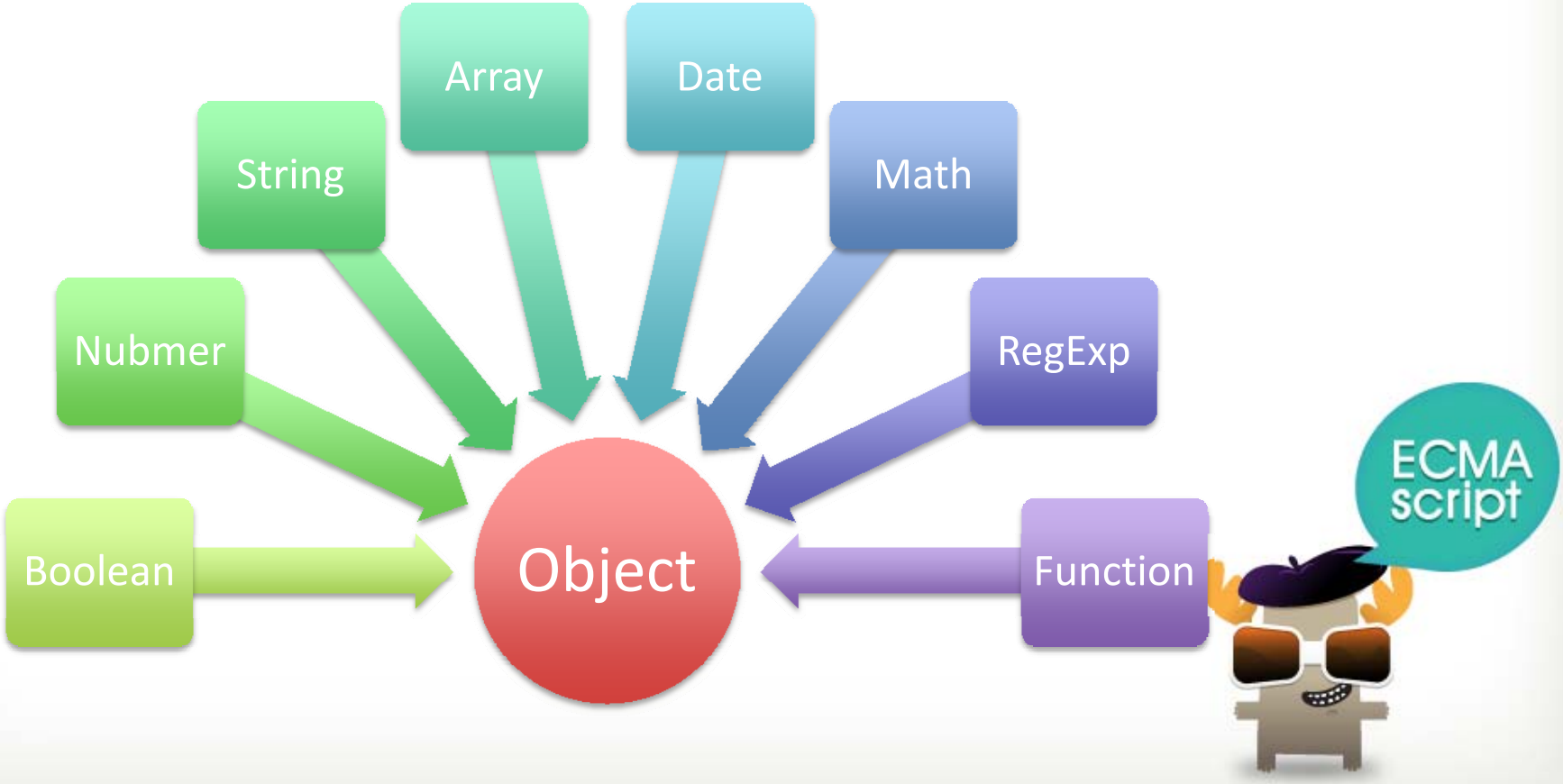
ECMAScript 引用类型

- 引用类型，通常指的是对象。

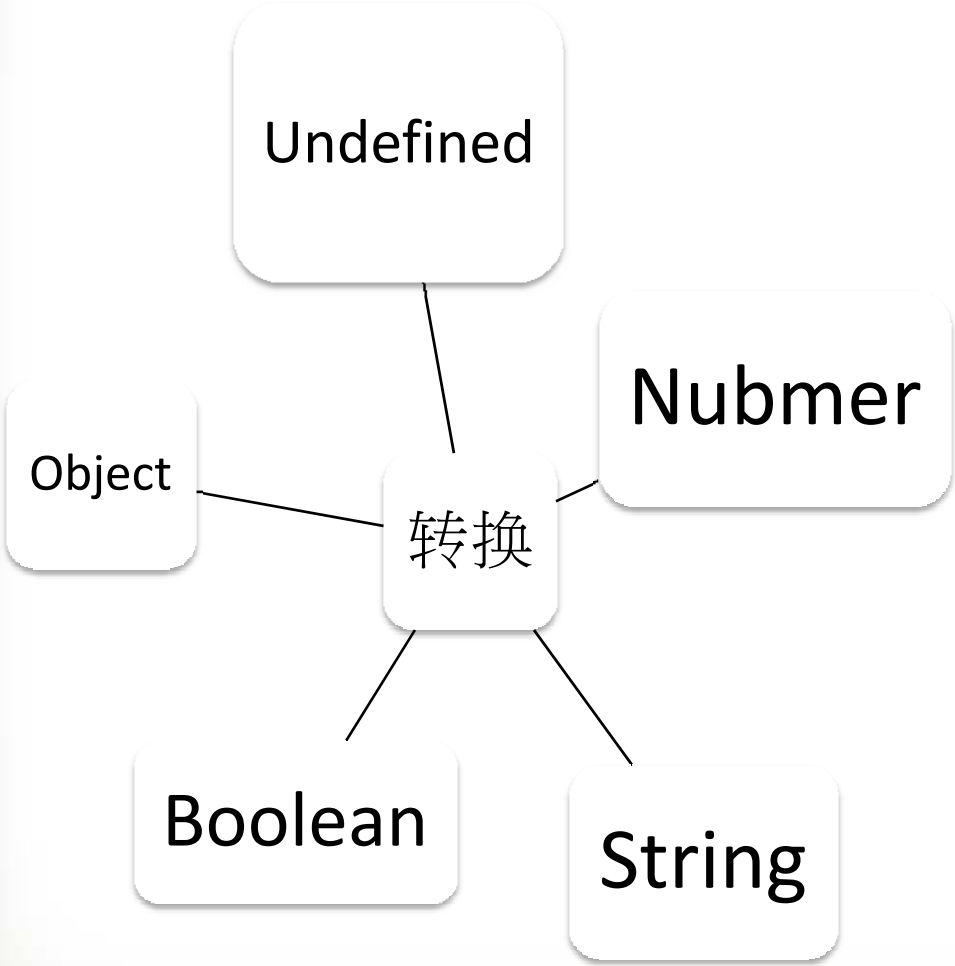


ECMAScript 引用类型

- ECMAScript包含以下几种预定义的引用类型



类型转换



- 通过运算符
- 通过if
- parseInt(), parseFloat()
- toString()
- 有些方法的参数会被强制进行类型转换，如：alert()

ECMAScript 运算符

- 一元运算符
- 位运算符
- 逻辑运算符
- 乘性运算符
- 加性运算符
- 关系运算符
- 等性运算符
- 条件运算符
- 赋值运算符
- 逗号运算符

一元运算符

- delete
  - 只能删除开发者自己定义的属性和方法
- void
  - 对任何值返回undefined
- 前增量、前减量运算
  - ++i --I
- 后增量、后减量运算
  - i++ i—
- typeof
- instanceof

ECMAScript 逻辑运算符

- NOT (!)
- AND (&&)
- OR (||)

ECMAScript 加性运算符

加号： +

1+1=?

1+ “1” =?

1+ “a” =?

减号： -

1-1=?

1- “1” =?

1- “a” =?

ECMAScript 乘性运算符

- 乘法运算符：*
- 除法运算符：/
- 取模（求余数）运算符：%

ECMAScript 关系运算符

- **关系运算符执行的是比较运算。每个关系运算符都返回一个布尔值。**

  >　　>=

  <　　<=

- **下面这几个分别都返回什么结果？**

```
var bResult = "25" < "3";
alert(bResult); //输出 "true"
```

```
var bResult = "25" < 3;
alert(bResult); //输出 "false"
```

```
var bResult = "a" < 3;
alert(bResult);
```

```
var bResult = "a" >= 3;
alert(bResult);
```

ECMAScript 等性运算符

- == !=
- ===（不进行类型转换）
- 执行类型转换的规则如下：
  - 如果一个运算数是 Boolean 值，在检查相等性之前，把它转换成数字值。false 转换成 0，true 为 1。
  - 如果一个运算数是字符串，另一个是数字，在检查相等性之前，要尝试把字符串转换成数字。
  - 如果一个运算数是对象，另一个是字符串，在检查相等性之前，要尝试把对象转换成字符串。
  - 如果一个运算数是对象，另一个是数字，在检查相等性之前，要尝试把对象转换成数字。
- 在比较时，该运算符还遵守下列规则：
  - 值 null 和 undefined 相等。
  - 在检查相等性时，不能把 null 和 undefined 转换成其他值。
  - 如果某个运算数是 NaN，等号将返回 false，非等号将返回 true。
  - 如果两个运算数都是对象，那么比较的是它们的引用值。如果两个运算数指向同一对象，那么等号返回 true，否则两个运算数不等。

ECMAScript 条件运算符

- variable = boolean_expression ? true_value : false_value;
- 例子：

```
var result;
if(confirm("I'm tall?")){
    result="I'm tall";
}else{
    result="I'm short";
}
```

- 等同于：

```
var result = confirm("I'm tall?")?"I'm tall":"I'm short";
```

ECMAScript 赋值运算符

- 乘法/赋值（*=）
- 除法/赋值（/=）
- 取模/赋值（%=）
- 加法/赋值（+=）
- 减法/赋值（-=）
- 左移/赋值（<<=）
- 有符号右移/赋值（>>=）
- 无符号右移/赋值（>>>=）

ECMAScript 语句

- 条件：

```
if (condition) statement1 else statement2
```

- Switch语句：

```
switch (expression)
  case value: statement;
        break;
 ...
case value: statement;
   break;
default: statement;
```

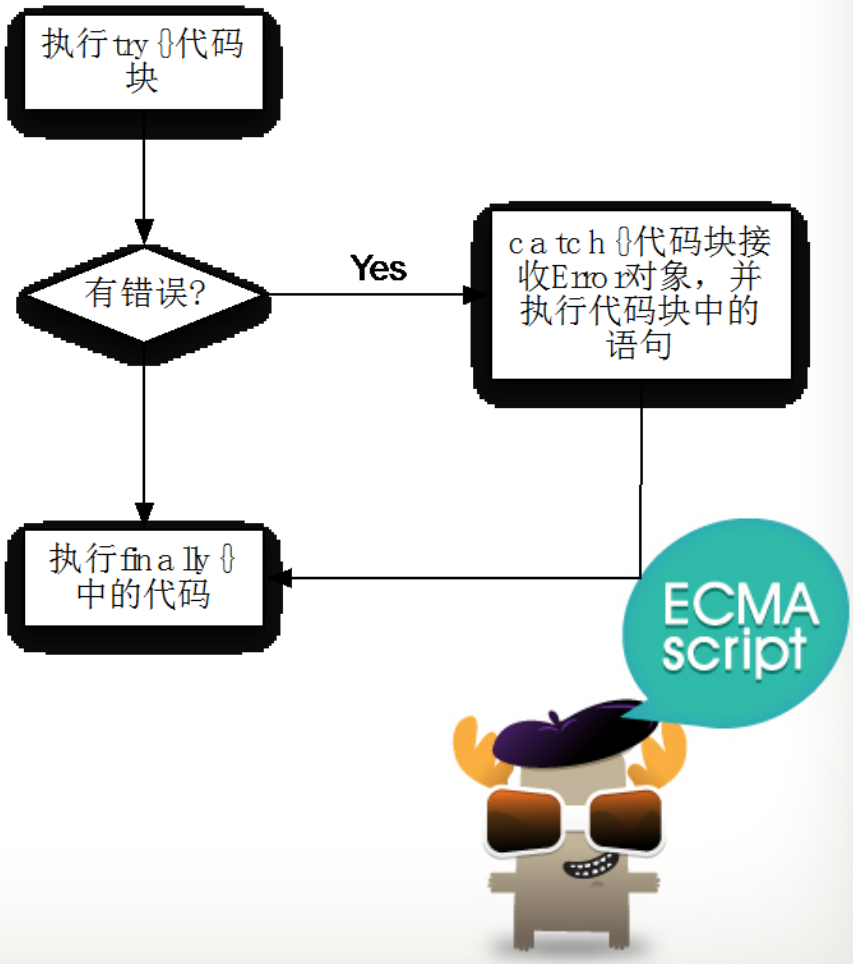
ECMAScript 语句

- 循环：
  do {statement} while (expression);
  while (expression) statement
  for (initialization; expression; post-loop-expression) statement
  for (property in object) statement
- 使用continue跳出一次循环
- 使用break跳出一层循环

ECMAScript 语句

- 结构化错误处理
- try {} catch(err) {} finally {}
- 也通过throw()抛出异常

- 使用场景：
  1. 屏蔽代码块中的错误
  2. 实现模块中的错误处理机制



ECMAScript 函数

- 定义,两种方式有何不同？
  - var fun1 = function(arg1){}
  - function fun1(arg1){}
- 调用 fun1(arg1)
- 参数对象：arguments
- 返回值：return

arguments.callee

ECMAScript 函数

- 闭包：**函数可以使用函数之外定义的变量。**

```
var sMessage = "hello world";

function sayHelloWorld() {
  alert(sMessage);
}

sayHelloWorld();
```

```
(function(){

})()
```

ECMAScript 函数

- this 指向函数运行的上下文环境，也就是调用该函数的对象

  function test(){alert(this)}

  test() this -> window

  var obj1 = {}; obj1.test=function(){alert(this)}

  obj1.test() this->obj1

ECMAScript 函数

- 改变this的指向
- 定义一个函数Fun,一个对象obj
- Fun.call(obj,arg1,arg2)
- Fun.apply(obj,[arg1,arg2])

ECMAScript 面向对象

- 对象的定义：属性的无序集合，每个属性存放一个原始值、对象或函数

- ECMAScript支持封装，继承，多态，可以被看作是面向对象的

类

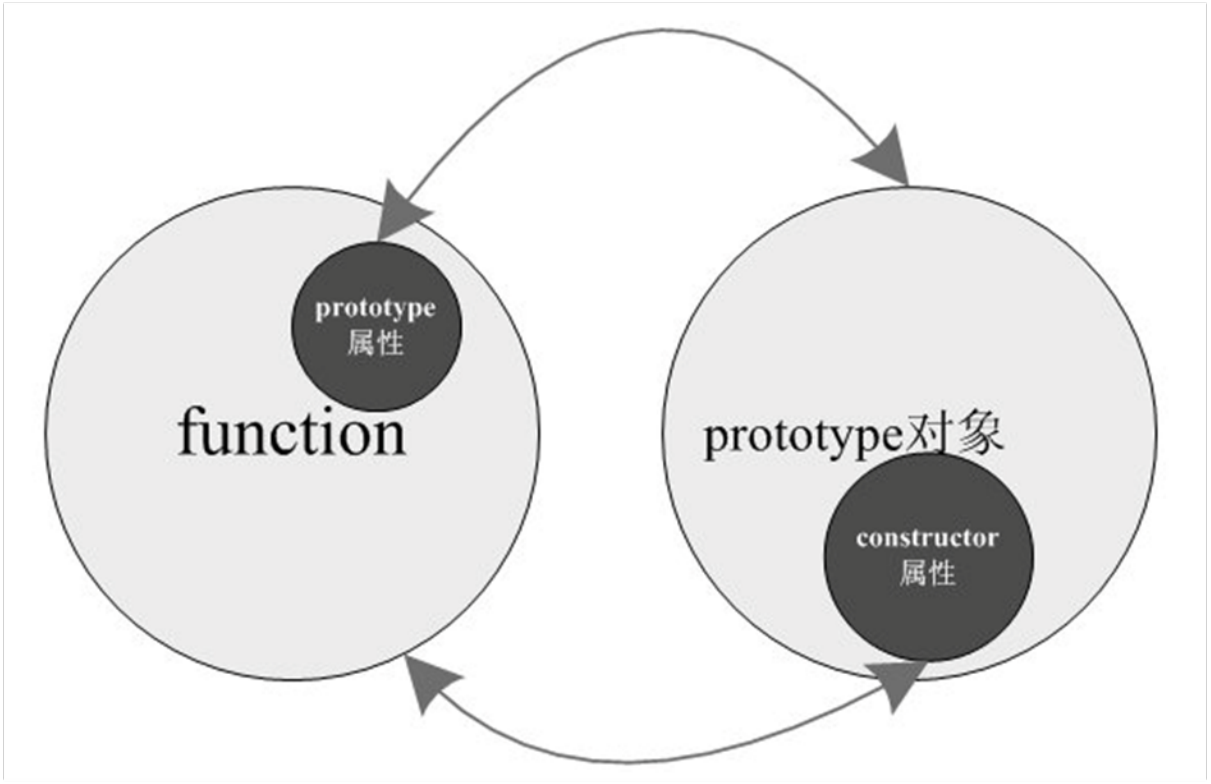
- 在ECMAScript中并没有真正的类，它使用函数来描述一个对象，解释器可以根据一个函数的描述来构建对象。因此在功能上讲函数和类是等价的。

```
function Foo1(){
    alert(“hello!”);
}
var obj = new Foo1();
```

结果会是啥？

```
function Foo1(){
    alert( “hello!” )
}
alert(Foo1.prototype.constructor)
```

prototype & constructor



创建类的方法

```
function myFunc(par1){
    this.prop1=par1;
    this.prop2=xxx;
}
myFunc.prototype.method1 = function(){}
myFunc.prototype.method2 = function(){}
```

```
var obj=new myFunc(par1)
```

创建一个对象（实例）

obj1 = new MyFunc()

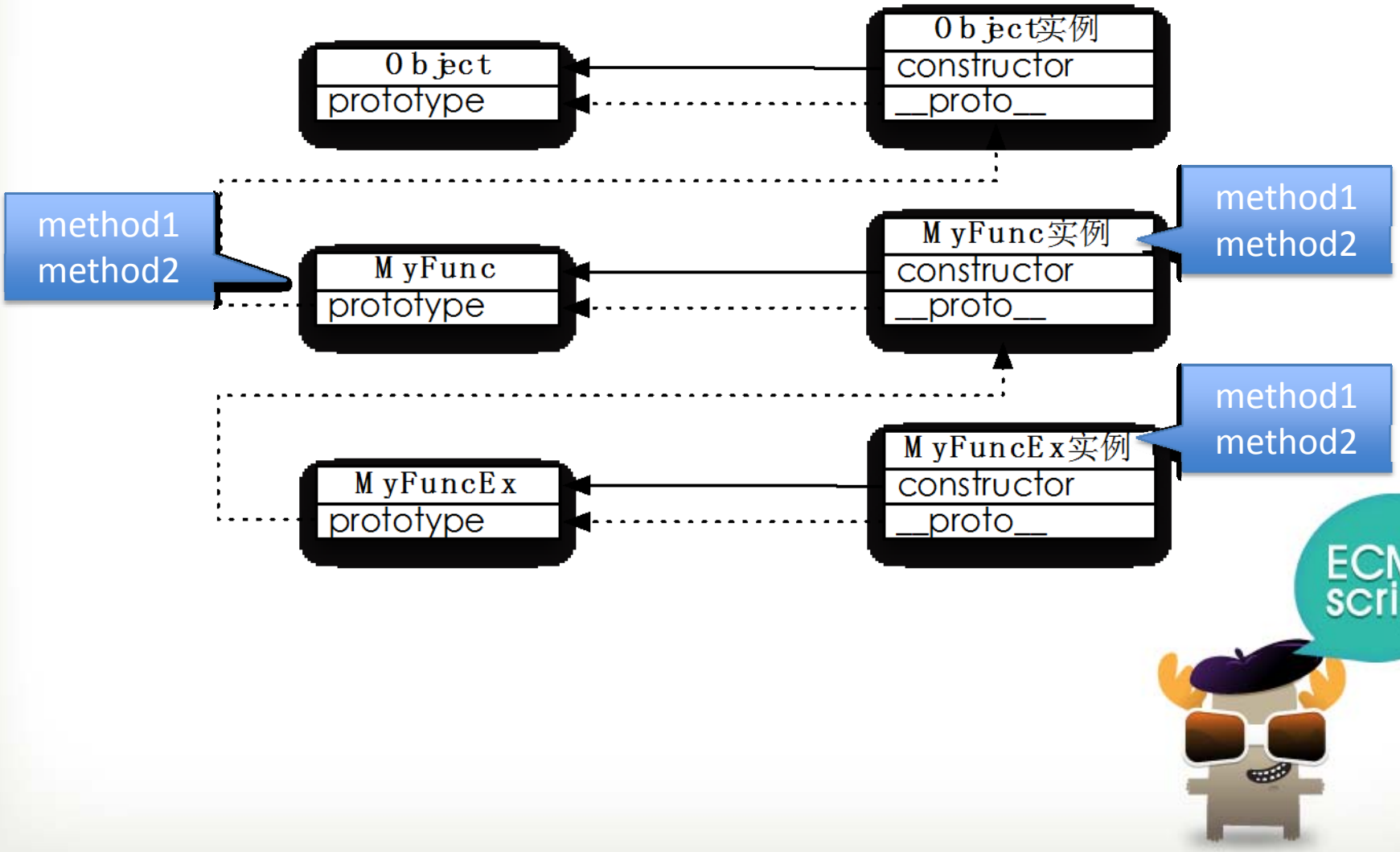
obj1指向一个新的对象

MyFunc的prototype如果是对象，则将它复制给obj1的内部属性_proto_

执行（构造函数）
MyFunc.call(obj1,arg1,arg2...)

- 使用new关键字

ECMAScript的原型链继承



继承

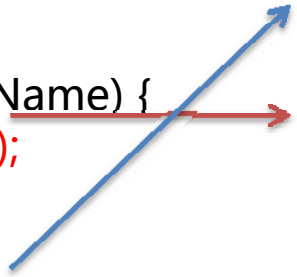
```
function ClassA(sColor) {
    this.color = sColor;
}

ClassA.prototype.sayColor = function ()
{
    alert(this.color);
};

function ClassB(sColor, sName) {
    ClassA.call(this, sColor);
    this.name = sName;
}

ClassB.prototype = new ClassA();
ClassB.prototype.constructor=ClassB;

ClassB.prototype.sayName = function
() {
    alert(this.name);
};
```



obj1 = new MyFunc()

obj1指向一个新的对象

MyFunc的prototype如果是对象，则将它复制给obj1的内部属性_proto_

执行（构造函数）
MyFunc.call(obj1,arg1,arg2...)

```
var objA = new ClassA("blue");
var objB = new ClassB("red", "John");
objA.sayColor();     //输出 "blue"
objB.sayColor();     //输出 "red"
objB.sayName();      //输出 "John"
```

关于在JS中使用面向对象？

- 维护
- 复用
- 模块
- 组件

Q & A


Thanks
Bye