**ECMAScript 预定义引用类型。**

引用类型通常叫做类（class），也就是说，遇到引用值，所处理的就是对象。

注意：从传统意义上来说，ECMAScript 并不真正具有类。事实上，除了说明不存在类，在 ECMA-262 中根本没有出现“类”这个词。ECMAScript 定义了“对象定义”，逻辑上等价于其他程序设计语言中的类。前端一般使用术语“对象”。

**Object 对象**

Object 对象自身用处不大，不过在了解其他类之前，还是应该了解它。因为 ECMAScript 中的 Object 对象与 Java 中的 java.lang.Object 相似，ECMAScript 中的所有对象都由这个对象继承而来，Object 对象中的所有属性和方法都会出现在其他对象中，所以理解了 Object 对象，就可以更好地理解其他对象。

**Object 对象具有下列属性：**

constructor

对创建对象的函数的引用（指针）。对于 Object 对象，该指针指向原始的 Object() 函数。

Prototype

对该对象的对象原型的引用。对于所有的对象，它默认返回 Object 对象的一个实例。

**Object 对象还具有几个方法：**

hasOwnProperty(property)

判断对象是否有某个特定的属性。必须用字符串指定该属性。（例如，o.hasOwnProperty("name")）

IsPrototypeOf(object)

判断该对象是否为另一个对象的原型。

PropertyIsEnumerable

判断给定的属性是否可以用 for...in 语句进行枚举。

ToString()

返回对象的原始字符串表示。对于 Object 对象，ECMA-262 没有定义这个值，所以不同的 ECMAScript 实现具有不同的值。

ValueOf()

返回最适合该对象的原始值。对于许多对象，该方法返回的值都与 ToString() 的返回值相同。

注释：上面列出的每种属性和方法都会被其他对象覆盖。

**Boolean 对象**

Boolean 对象是 Boolean 原始类型的引用类型。

要创建 Boolean 对象，只需要传递 Boolean 值作为参数：

var oBooleanObject = new Boolean(true);

Boolean 对象将覆盖 Object 对象的 ValueOf() 方法，返回原始值，即 true 和 false。ToString() 方法也会被覆盖，返回字符串 "true" 或 "false"。

遗憾的是，在 ECMAScript 中很少使用 Boolean 对象，即使使用，也不易理解。

问题通常出现在 Boolean 表达式中使用 Boolean 对象时。例如：

var oFalseObject = new Boolean(false);

var bResult = oFalseObject && true; //输出 true

在这段代码中，用 false 值创建 Boolean 对象。然后用这个值与原始值 true 进行 AND 操作。在 Boolean 运算中，false 和 true 进行 AND 操作的结果是 false。不过，在这行代码中，计算的是 oFalseObject，而不是它的值 false。

正如前面讨论过的，在 Boolean 表达式中，所有对象都会被自动转换为 true，所以 oFalseObject 的值是 true。然后 true 再与 true 进行 AND 操作，结果为 true。

注意：虽然你应该了解 Boolean 对象的可用性，不过最好还是使用 Boolean 原始值，避免发生这一节提到的问题。

**Number 对象**

正如你可能想到的，Number 对象是 Number 原始类型的引用类型。要创建 Number 对象，采用下列代码：

var oNumberObject = new Number(68);

您应该已认出本章前面小节中讨论特殊值（如 Number.MAX\_VALUE）时提到的 Number 对象。所有特殊值都是 Number 对象的静态属性。

要得到数字对象的 Number 原始值，只需要使用 valueOf() 方法：

var iNumber = oNumberObject.valueOf();

当然，Number 类也有 toString() 方法，在讨论类型转换的小节中已经详细讨论过该方法。

除了从 Object 对象继承的标准方法外，Number 对象还有几个处理数值的专用方法。

**toFixed() 方法**

toFixed() 方法返回的是具有指定位数小数的数字的字符串表示。例如：

var oNumberObject = new Number(68);

alert(oNumberObject.toFixed(2)); //输出 "68.00"

在这里，toFixed() 方法的参数是 2，说明应该显示两位小数。该方法返回 "68.00"，空的字符串位由 0 来补充。对于处理货币的应用程序，该方法非常有用。toFixed() 方法能表示具有 0 到 20 位小数的数字，超过这个范围的值会引发错误。

**toExponential() 方法**

与格式化数字相关的另一个方法是 toExponential()，它返回的是用科学计数法表示的数字的字符串形式。

与 toFixed() 方法相似，toExponential() 方法也有一个参数，指定要输出的小数的位数。例如：

var oNumberObject = new Number(68);

alert(oNumberObject.toExponential(1)); //输出 "6.8e+1"

这段代码的结果是 "6.8e+1"，前面解释过，它表示 6.8x101。问题是，如果不知道要用哪种形式（预定形式或指数形式）表示数字怎么办？可以用 toPrecision() 方法。

**toPrecision() 方法**

toPrecision() 方法根据最有意义的形式来返回数字的预定形式或指数形式。它有一个参数，即用于表示数的数字总数（不包括指数）。例如，

var oNumberObject = new Number(68);

alert(oNumberObject.toPrecision(1)); //输出 "7e+1"

这段代码的任务是用一位数字表示数字 68，结果为 "7e+1"，以另外的形式表示即 70。的确，toPrecision() 方法会对数进行舍入。不过，如果用 2 位数字表示 68，就容易多了：

var oNumberObject = new Number(68);

alert(oNumberObject.toPrecision(2)); //输出 "68"

当然，输出的是 "68"，因为这正是该数的准确表示。不过，如果指定的位数多于需要的位数又如何呢？

var oNumberObject = new Number(68);

alert(oNumberObject.toPrecision(3)); //输出 "68.0"

在这种情况下，toPrecision(3) 等价于 toFixed(1)，输出的是 "68.0"。

toFixed()、toExponential() 和 toPrecision() 方法都会进行舍入操作，以便用正确的小数位数正确地表示一个数。

提示：与 Boolean 对象相似，Number 对象也很重要，不过应该少用这种对象，以避免潜在的问题。只要可能，都使用数字的原始表示法。

**String 对象**

String 对象是 String 原始类型的对象表示法，它是以下方式创建的：

var oStringObject = new String("hello world");

String 对象的 valueOf() 方法和 toString() 方法都会返回 String 类型的原始值：

alert(oStringObject.valueOf() == oStringObject.toString()); //输出 "true"

如果运行这段代码，输出是 "true"，说明这些值真的相等。

**length 属性**

String 对象具有属性 length，它是字符串中的字符个数：

var oStringObject = new String("hello world");

alert(oStringObject.length); //输出 "11"

这个例子输出的是 "11"，即 "hello world" 中的字符个数。注意，即使字符串包含双字节的字符（与 ASCII 字符相对，ASCII 字符只占用一个字节），每个字符也只算一个字符。

**charAt() 和 charCodeAt() 方法**

String 对象还拥有大量的方法。

首先，两个方法 charAt() 和 charCodeAt() 访问的是字符串中的单个字符。这两个方法都有一个参数，即要操作的字符的位置。

charAt() 方法返回的是包含指定位置处的字符的字符串：

var oStringObject = new String("hello world");

alert(oStringObject.charAt(1)); //输出 "e"

在字符串 "hello world" 中，位置 1 处的字符是 "e"。在“ECMAScript 原始类型”这一节中我们讲过，第一个字符的位置是 0，第二个字符的位置是 1，依此类推。因此，调用 charAt(1) 返回的是 "e"。

如果想得到的不是字符，而是字符代码，那么可以调用 charCodeAt() 方法：

var oStringObject = new String("hello world");

alert(oStringObject.charCodeAt(1)); //输出 "101"

这个例子输出 "101"，即小写字母 "e" 的字符代码。

**concat() 方法**

接下来是 concat() 方法，用于把一个或多个字符串连接到 String 对象的原始值上。该方法返回的是 String 原始值，保持原始的 String 对象不变：

var oStringObject = new String("hello ");

var sResult = oStringObject.concat("world");

alert(sResult); //输出 "hello world"

alert(oStringObject); //输出 "hello "

在上面这段代码中，调用 concat() 方法返回的是 "hello world"，而 String 对象存放的仍然是 "hello "。出于这种原因，较常见的是用加号（+）连接字符串，因为这种形式从逻辑上表明了真正的行为：

var oStringObject = new String("hello ");

var sResult = oStringObject + "world";

alert(sResult); //输出 "hello world"

alert(oStringObject); //输出 "hello "

**indexOf() 和 lastIndexOf() 方法**

迄今为止，已讨论过连接字符串的方法，访问字符串中的单个字符的方法。不过如果无法确定在某个字符串中是否确实存在一个字符，应该调用什么方法呢？这时，可调用 indexOf() 和 lastIndexOf() 方法。

indexOf() 和 lastIndexOf() 方法返回的都是指定的子串在另一个字符串中的位置，如果没有找不到子串，则返回 -1。

这两个方法的不同之处在于，indexOf() 方法是从字符串的开头（位置 0）开始检索字符串，而 lastIndexOf() 方法则是从字符串的结尾开始检索子串。例如：

var oStringObject = new String("hello world!");

alert(oStringObject.indexOf("o")); 输出 "4"

alert(oStringObject.lastIndexOf("o")); 输出 "7"

在这里，第一个 "o" 字符串出现在位置 4，即 "hello" 中的 "o"；最后一个 "o" 出现在位置 7，即 "world" 中的 "o"。如果该字符串中只有一个 "o" 字符串，那么 indexOf() 和 lastIndexOf() 方法返回的位置相同。

**localeCompare() 方法**

下一个方法是 localeCompare()，对字符串进行排序。该方法有一个参数 - 要进行比较的字符串，返回的是下列三个值之一：

* 如果 String 对象按照字母顺序排在参数中的字符串之前，返回负数。
* 如果 String 对象等于参数中的字符串，返回 0
* 如果 String 对象按照字母顺序排在参数中的字符串之后，返回正数。

注释：如果返回负数，那么最常见的是 -1，不过真正返回的是由实现决定的。如果返回正数，那么同样的，最常见的是 1，不过真正返回的是由实现决定的。

示例如下：

var oStringObject = new String("yellow");

alert(oStringObject.localeCompare("brick")); //输出 "1"

alert(oStringObject.localeCompare("yellow")); //输出 "0"

alert(oStringObject.localeCompare("zoo")); //输出 "-1"

在这段代码中，字符串 "yellow" 与 3 个值进行了对比，即 "brick"、"yellow" 和 "zoo"。由于按照字母顺序排列，"yellow" 位于 "brick" 之后，所以 localeCompare() 返回 1；"yellow" 等于 "yellow"，所以 localeCompare() 返回 0；"zoo" 位于 "yellow" 之后，localeCompare() 返回 -1。再强调一次，由于返回的值是由实现决定的，所以最好以下面的方式调用 localeCompare() 方法：

var oStringObject1 = new String("yellow");

var oStringObject2 = new String("brick");

var iResult = oStringObject1.localeCompare(oStringObject2);

if(iResult < 0) {

alert(oStringObject1 + " comes before " + oStringObject2);

} else if (iResult > 0) {

alert(oStringObject1 + " comes after " + oStringObject2);

} else {

alert("The two strings are equal");

}

采用这种结构，可以确保这段代码在所有实现中都能正确运行。

localeCompare() 方法的独特之处在于，实现所处的区域（locale，兼指国家/地区和语言）确切说明了这种方法运行的方式。在美国，英语是 ECMAScript 实现的标准语言，localeCompare() 是区分大小写的，大写字母在字母顺序上排在小写字母之后。不过，在其他区域，情况可能并非如此。

**slice() 和 substring()**

ECMAScript 提供了两种方法从子串创建字符串值，即 slice() 和 substring()。这两种方法返回的都是要处理的字符串的子串，都接受一个或两个参数。第一个参数是要获取的子串的起始位置，第二个参数（如果使用 的话）是要获取子串终止前的位置（也就是说，获取终止位置处的字符不包括在返回的值内）。如果省略第二个参数，终止位就默认为字符串的长度。

与 concat() 方法一样，slice() 和 substring() 方法都不改变 String 对象自身的值。它们只返回原始的 String 值，保持 String 对象不变。

var oStringObject = new String("hello world");

alert(oStringObject.slice("3")); //输出 "lo world"

alert(oStringObject.substring("3")); //输出 "lo world"

alert(oStringObject.slice("3", "7")); //输出 "lo w"

alert(oStringObject.substring("3", "7")); //输出 "lo w"

在这个例子中，slice() 和 substring() 的用法相同，返回值也一样。当只有参数 3 时，两个方法返回的都是 "lo world"，因为 "hello" 中的第二个 "l" 位于位置 3 上。当有两个参数 "3" 和 "7" 时，两个方法返回的值都是 "lo w"（"world" 中的字母 "o" 位于位置 7 上，所以它不包括在结果中）。

为什么有两个功能完全相同的方法呢？事实上，这两个方法并不完全相同，不过只在参数为负数时，它们处理参数的方式才稍有不同。

对于负数参数，slice() 方法会用字符串的长度加上参数，substring() 方法则将其作为 0 处理（也就是说将忽略它）。例如：

var oStringObject = new String("hello world");

alert(oStringObject.slice("-3")); //输出 "rld"

alert(oStringObject.substring("-3")); //输出 "hello world"

alert(oStringObject.slice("3, -4")); //输出 "lo w"

alert(oStringObject.substring("3, -4")); //输出 "hel"

这样即可看出 slice() 和 substring() 方法的主要不同。

当只有参数 -3 时，slice() 返回 "rld"，substring() 则返回 "hello world"。这是因为对于字符串 "hello world"，slice("-3") 将被转换成 slice("8")，而 substring("-3") 将被转换成 substring("0")。

同样，使用参数 3 和 -4 时，差别也很明显。slice() 将被转换成 slice(3, 7)，与前面的例子相同，返回 "lo w"。而 substring() 方法则将两个参数解释为 substring(3, 0)，实际上即 substring(0, 3)，因为 substring() 总把较小的数字作为起始位，较大的数字作为终止位。因此，substring("3, -4") 返回的是 "hel"。这里的最后一行代码用来说明如何使用这些方法。

**toLowerCase()、toLocaleLowerCase()、toUpperCase() 和 toLocaleUpperCase()**

最后一套要讨论的方法涉及大小写转换。有 4 种方法用于执行大小写转换，即

* toLowerCase()
* toLocaleLowerCase()
* toUpperCase()
* toLocaleUpperCase()

从名字上可以看出它们的用途，前两种方法用于把字符串转换成全小写的，后两种方法用于把字符串转换成全大写的。

toLowerCase() 和 toUpperCase() 方法是原始的，是以 java.lang.String 中相同方法为原型实现的。

toLocaleLowerCase() 和 toLocaleUpperCase() 方法是基于特定的区域实现的（与 localeCompare() 方法相同）。在许多区域中，区域特定的方法都与通用的方法完全相同。不过，有几种语言对 Unicode 大小写转换应用了特定的规则（例如土耳其语），因此必须使用区域特定的方法才能进行正确的转换。

var oStringObject = new String("Hello World");

alert(oStringObject.toLocaleUpperCase()); //输出 "HELLO WORLD"

alert(oStringObject.toUpperCase()); //输出 "HELLO WORLD"

alert(oStringObject.toLocaleLowerCase()); //输出 "hello world"

alert(oStringObject.toLowerCase()); //输出 "hello world"

这段代码中，toUpperCase() 和 toLocaleUpperCase() 输出的都是 "HELLO WORLD"，toLowerCase() 和 toLocaleLowerCase() 输出的都是 "hello world"。一般来说，如果不知道在以哪种编码运行一种语言，则使用区域特定的方法比较安全。

提示：记住，String 对象的所有属性和方法都可应用于 String 原始值上，因为它们是伪对象。

**instanceof 运算符**

在使用 typeof 运算符时采用引用类型存储值会出现一个问题，无论引用的是什么类型的对象，它都返回 "object"。ECMAScript 引入了另一个 Java 运算符 instanceof 来解决这个问题。

instanceof 运算符与 typeof 运算符相似，用于识别正在处理的对象的类型。与 typeof 方法不同的是，instanceof 方法要求开发者明确地确认对象为某特定类型。例如：

var oStringObject = new String("hello world");

alert(oStringObject instanceof String); //输出 "true"

这段代码问的是“变量 oStringObject 是否为 String 对象的实例？”oStringObject 的确是 String 对象的实例，因此结果是 "true"。尽管不像 typeof 方法那样灵活，但是在 typeof 方法返回 "object" 的情况下，instanceof 方法还是很有用的。