包装对象

定义

对象是 JavaScript 语言最主要的数据类型，三种原始类型的值——数值、字符串、布尔值——在一定条件下，也会自动转为对象，也就是原始类型的“包装对象”（wrapper）。

所谓“包装对象”，指的是与数值、字符串、布尔值分别相对应的`Number`、`String`、`Boolean`三个原生对象。这三个原生对象可以把原始类型的值变成（包装成）对象。

```javascript
var v1 = new Number(123);
var v2 = new String('abc');
var v3 = new Boolean(true);

typeof v1 // "object"
typeof v2 // "object"
typeof v3 // "object"

v1 === 123 // false
v2 === 'abc' // false
v3 === true // false
```

上面代码中，基于原始类型的值，生成了三个对应的包装对象。可以看到，`v1`、`v2`、`v3`都是对象，且与对应的简单类型值不相等。

包装对象的设计目的，首先是使得“对象”这种类型可以覆盖 JavaScript 所有的值，整门语言有一个通用的数据模型，其次是使得原始类型的值也有办法调用自己的方法。

`Number`、`String`和`Boolean`这三个原生对象，如果不作为构造函数调用（即调用时不加`new`），而是作为普通函数调用，常常用于将任意类型的值转为数值、字符串和布尔值。

```javascript
// 字符串转为数值
Number('123') // 123

// 数值转为字符串
String(123) // "123"

// 数值转为布尔值
Boolean(123) // true
```

上面这种数据类型的转换，详见《数据类型转换》一节。

总结一下，这三个对象作为构造函数使用（带有`new`）时，可以将原始类型的值转为对象；作为普通函数使用时（不带有`new`），可以将任意类型的值，转为原始类型的值。

实例方法

三种包装对象各自提供了许多实例方法，详见后文。这里介绍两种它们共同具有、从`Object`对象继承的方法：`valueOf()`和`toString()`。

valueOf()

`valueOf()`方法返回包装对象实例对应的原始类型的值。

```javascript
new Number(123).valueOf()  // 123
new String('abc').valueOf() // "abc"
new Boolean(true).valueOf() // true
```

toString()

`toString()`方法返回对应的字符串形式。

```javascript
new Number(123).toString() // "123"
new String('abc').toString() // "abc"
new Boolean(true).toString() // "true"
```

原始类型与实例对象的自动转换

某些场合，原始类型的值会自动当作包装对象调用，即调用包装对象的属性和方法。这时，JavaScript 引擎会自动将原始类型的值转为包装对象实例，并在使用后立刻销毁实例。

比如，字符串可以调用`length`属性，返回字符串的长度。

```javascript
'abc'.length // 3
```

上面代码中，`abc`是一个字符串，本身不是对象，不能调用`length`属性。JavaScript 引擎自动将其转为包装对象，在这个对象上调用`length`属性。调用结束后，这个临时对象就会被销毁。这就叫原始类型与实例对象的自动转换。

```javascript
var str = 'abc';
str.length // 3

// 等同于
```

```
var strObj = new String(str)
// String {
//   0: "a", 1: "b", 2: "c", length: 3, [[PrimitiveValue]]: "abc"
// }
strObj.length // 3
```

上面代码中，字符串`abc`的包装对象提供了多个属性，`length`只是其中之一。

自动转换生成的包装对象是只读的，无法修改。所以，字符串无法添加新属性。

```javascript
var s = 'Hello World';
s.x = 123;
s.x // undefined
```

上面代码为字符串`s`添加了一个`x`属性，结果无效，总是返回`undefined`。

另一方面，调用结束后，包装对象实例会自动销毁。这意味着，下一次调用字符串的属性时，实际是调用一个新生成的对象，而不是上一次调用时生成的那个对象，所以取不到赋值在上一个对象的属性。如果要为字符串添加属性，只有在它的原型对象`String.prototype`上定义（参见《面向对象编程》章节）。

自定义方法

除了原生的实例方法，包装对象还可以自定义方法和属性，供原始类型的值直接调用。

比如，我们可以新增一个`double`方法，使得字符串和数字翻倍。

```javascript
String.prototype.double = function () {
  return this.valueOf() + this.valueOf();
};

'abc'.double()
// abcabc

Number.prototype.double = function () {
  return this.valueOf() + this.valueOf();
};

(123).double() // 246
```

上面代码在`String`和`Number`这两个对象的原型上面，分别自定义了一个方法，从而可以在所有实例对象上调用。注意，最后一张的`123`外面必须要加上圆括号，否则后面的点运算符（`.`）会被解释成小数点。