## 第4条：原始类型优于封装对象

比如，你可以创建一个String对象，该对象封装了一个字符串的值；

var str = **new** *String*("hello word"); // 封装对象

var str2 = "hello word"; // 原始类型

原始类型的字符串你和封装类型的字符串基本上可以在编程的时候相同对待。

但是，需要注意的是，原始类型是一个真真正正的对象。也就是说两个内容相容的封装类型字符串是不相等的，因为他们两个根本就是两个人，只不过叫的名字是一样的而已。就像一个员工，每个员工对象都由同一个构造函数构造而来，但是员工和员工依旧是两个人。即使他们两个是克隆人，信息完全相同！这里的字符串也是如此。

var str = **new** *String*("hello word"); // 封装对象

var str2 = "hello word"; // 原始类型

var str3 = **new** *String*("hello word"); // 内容相同的封装对象

*console*.log(str == str2); // true

*console*.log(str == str3); // false

对于字符串而言，封装对象明显更麻烦，所以用处不大。原始类型的字符串在需要的时候会经过js的隐式调用，这后面经历了一系列的调用操作，结果导致，你可以很方便的对原始字符串进行各种操作（使用封装类型中的方法和属性），而不会发生错误（我的理解是原始字符串天生就没有经过对象的包装，因此天生也不具备字符串对象的那些属性和方法，他就是单纯的字符串而已，只不过为了方便，在对这些原始字符串使用属性和方法的时候，js背后对这些原始字符串进行了暂时的封装）。

*console*.log("hello word".length); // 10

因为是对原始字符串进行操作，js会对原始的东西进行隐式的封装，因此，会导致一种奇怪的现象，这就是可以对原始值设置属性，但是丝毫没有任何影响。

"hello".someProperty = 17;

*console*.log("hello".someProperty) ; // undefined

还有一个问题就是，不知道为什么，原始字符串竟然会和内容相同的封装类型所得到的字符串对象相等?

var str = **new** *String*("hello word"); // 封装对象！！！

var str2 = "hello word"; // 原始类型!!!

var str3 = **new** *String*("hello word"); // 内容相同的封装对象

*console*.log(str == str2); // true!!!

*console*.log(str == str3); // false

对于这一点我的想法是，在进行比较的时候，原始类型因为经过了隐式封装，调用了valueOf(),读取了经过隐式封装后的值，然后与封装对象的valueOf()进行比较，因为两个valueOf的结果都是长得一样的原始字符串，因此相等。但是如果改为“===”就又不相等了，按照我的想法，应该在“===”的时候也相等才对，因为两个长得一样的原始字符串作严格比较依旧相等，具体细节仍然有待考证。

具体js是怎么隐式封装操作的，高程的笔记中会有解答。