a + b运算操作中，a和b用“+”连接，使得a和b都要先进行隐式强制类型转换，再做“加”运算。

1）当“+”符号两端存在一个或两个不是数字类型的值时，需要把“+”符号两端都先进行“取原始值”操作（ToPrimitive）。

2）a和b都是数组类型(**或是引用类型**)，要进行取原始值操作的话，解析器会对数组先调用valueOf()方法，如果有该方法并且返回值是原始类型的话就是这个返回值；否则继续调用toString()方法，如果返回的是原始值，则利用这个返回值进行“加”运算；否则会抛出异常

**示例解析 ： []+{} 和 {} + []**

[]+{}，都不是简单类型，先用valueOf或toString取原始值，再拼接两头的值。结果是 ‘’+ “[object object]”-->“[object object]”

{} + []，涉及js解析原理，｛｝被当作代码块做运算，这里返回空，+[]就被当作一元运算，[]当作0，结果是 +0 –-> 0

显式强制类型转换是通过显而易见的、目的明确的代码将数据进行强制类型转换，如Number()就是将变量显式的强制转换为数字类型的值。

隐式强制类型转换往往是一些操作的附带产物，如if(){}中会将括号内的部分转换为布尔类型。

而关于”显式“和”隐式“是很主观的，如果+”123“对你来说，你一眼就能看出这是将字符串”123“转换为数字的操作，那么这对于你来说就是显式的强制类型转换。

**示例解析：String(a) 和 a+""**

var a = {

valueOf: ()=>"123",

toString: ()=>"456"

}

String(a); //"456"

a + ""; //"123"

通过现象可以看得出来，String()是直接调用了a的toString()方法，而a+""则是先调用了valueOf()方法。原因是加运算是隐式强制类型转换，会对两端操作数进行ToPrimitive操作。

**有哪些值强制转换成布尔类型时结果为false？**

undefined，null，-0，+0，NaN，false， ""

**将一个变量强制转换为数字类型时，都进行了哪些操作**

将变量强制转换为数字遵循的是ToNumber操作。

对于基本类型的话：

true → 1

false → 0

undefined → NaN

null → 0

对于字符串，遵循常量的相关规则语法，如果转化失败就返回NaN

对于对象来说：

会先进行去原始值操作ToPrimitive，即先检查该值是否有valueOf()方法，如果有并且返回的基本类型值，就使用该值进行转强制类型转换。如果不是就使用toString()的返回值进行强制类型转换。如果valueOf()和toString()均不返回基本类型值，会产生TypeError错误。

**宽松相等——“==”和严格相等——“===”有什么区别？**

==允许在相等比较中进行强制类型转换，而===不允许，转换规则如下

