* **=== 不做类型转换，类型不同的一定不等，返回false；**
* **两个string严格相等表示它们有相同的字符排列、相同的长度和每个位置的字符都相同；**
* **两个number严格相等表示它们有相同的数值，NaN和任何东西都不相等，包括NaN它自己；正负零彼此之间相等；**
* **两个boolean严格相等表示它们同时为true或者同时为false；**
* **两个不同的object在严格和概要比较中都不相等，返回false；**
* **两个object相等唯一的情况是他们引用了相同的object；**
* **== 在两边值类型不同的时候，会做如下的转换再严格比较：**
  + null == undefined 但是 null !== undefined；
  + 如果有一个操作数是一个数字或布尔值，如果可能，另一个操作数转换为数字；否则，如果其中一个操作数为字符串，如果可能，另一个操作数被转换为字符串；
  + 如果两个操作数都是对象，那会比较对象在内存中的引用是否相同。

根据上面的规则，我们知道：如果在比较时两个变量的类型很重要，就要使用严格比较（===）；否则可以使用一般比较（==）。

在JavaScript中，下面的值被当做假（false），除了下面列出的值，都被当做真（true）：

* false
* null
* undefined
* 空字符串 ”
* 数字 0
* NaN

类型不等进行==比较时，先进行类型转换，转换成同类型。

空数组[]当做对象时，为true

任意值与布尔值比较，都会将两边的值转化为Number

[**js 空数组是true还是false**](http://www.cnblogs.com/frostbelt/p/3425498.html)

|  |  |
| --- | --- |
| 1  2 | var arr = new Array();  // 或 var arr = []; |

我们知道，初始化后，即使数组arr中没有元素，也是一个object。

|  |  |
| --- | --- |
| 1  2 | typeof arr;  // "object" |

既然是object，用于判断条件时就会被转化为true

|  |  |
| --- | --- |
| 1  2 | if(arr)console.log("it's true");  // it's true |

但是，如果将arr与布尔值比较：

|  |  |
| --- | --- |
| 1  2  3  4 | arr == false;  // true  arr == true;  // false |

可是，如果把arr转化为Boolean，的确是true：

|  |  |
| --- | --- |
| 1  2 | Boolean(arr);  // true |

那arr与布尔值比较时，到底发生了什么？

原来，任意值与布尔值比较，都会将两边的值转化为Number。

如arr与false比较，false转化为0，而arr为空数组，也转化为0：

|  |  |
| --- | --- |
| 1  2  3  4 | Number(false)  // 0  Number(arr)  // 0 |

所以，当空数组作为判断条件时，相当于true。当空数组与布尔值直接比较时，相当于false。

也就出现了以下令人绕圈的现象：

|  |  |
| --- | --- |
| 1  2 | [] == ![];  // true |

如何避免数组与布尔值比较时出现的坑呢？可以先把数组转化为布尔值：

|  |  |
| --- | --- |
| 1  2 | Boolean(arr) == true;  // true |

有同学问：

|  |  |
| --- | --- |
| 1 | new Array() == []; |

必须是false，就和new Array() == new Array()一样。是不同的两个对象。

但有个问题很奇怪：

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7 | new Array(1) == false;  // true    // 或    [undefined] == false;  // true |

这个怎么解释呢？Number(new Array(1))为什么会是0呢？