Comparison

`JavaScript` 中有三种比较规范：

- 抽象比较（使用 `==` 双等操作符）
- 严格比较（使用 `===` 三等操作符）
- 同值比较（ `Object.is` ）

抽象比较规则

`x == y` 表达式将返回 `true` 或 `false`，该表达式求值步骤如下：

1. 如果 `x` 和 `y` 的类型相同，则：
    1. 如果 `x` 为 `Undefined 类型` 则返回 `true`
    2. 如果 `x` 为 `Null 类型` 则返回 `true`
    3. 如果 `x` 为 `Number 类型`，则：
        1. 如果 `x` 为 `NaN` 则返回 `false`
        2. 如果 `y` 为 `NaN` 则返回 `false`
        3. 如果 `x` 和 `y` 的数值相同则返回 `true`
        4. 如果 `x` 为 `+0` 并且 `y` 为 `-0` 则返回 `true`
        5. 如果 `x` 为 `-0` 并且 `y` 为 `+0` 则返回 `true`
    4. 如果 `x` 为 `String 类型`，并且 `x` 和 `y` 具有相同的字符序列则返回 `true`，否则返回 `false`
    5. 如果 `x` 为 `Boolean 类型`，并且 `x` 和 `y` 都为 `true` 或者都为 `false` 则返回 `true`，否则返回 `false`
    6. 如果 `x` 和 `y` 为指向同一个对象的 `Object 类型` 则返回 `true`，否则返回 `false`
2. 如果 `x` 为 `null` 且 `y` 为 `undefined` 则返回 `true`
3. 如果 `x` 为 `undefined` 且 `y` 为 `null` 则返回 `true`
4. 如果 `x` 为 `Number 类型` 且 `y` 为 `String 类型` 则返回 `x == ToNumber(y)`
5. 如果 `x` 为 `String 类型` 且 `y` 为 `Number 类型` 则返回 `ToNumber(x) == y`
6. 如果 `x` 为 `Boolean 类型` 则返回 `ToNumber(x) == y`
7. 如果 `y` 为 `Boolean 类型` 则返回 `x == ToNumber(y)`
8. 如果 `x` 为 `String or Number 类型` 并且 `y` 为 `Object 类型` 则返回 `x == ToPrimitive(y)`
9. 如果 `x` 为 `Object 类型` 并且 `y` 为 `String or Number 类型` 则返回 `ToPrimitive(x) == y`
10. 返回 `false`

通过以上规则可以看出，抽象比较规则中存在类型隐性转换，其中的 `ToNumber` `ToPrimitive` 就是转换动作，其实这种转换方法有很多，本文只讨论如下四个：

- `ToNumber`
- `ToString`
- `ToBoolean`
- `ToPrimitive`

这四个方法全部都是 `internal method`，也就是未开放给用户的内置方法，由引擎在内部间接调用

当我们使用 `Number(x)` `String(x)` `Boolean(x)` 显示的进行类型转换的时候，引擎就会在内部分别调用 `ToNumber(x)` `ToString(x)` `ToBoolean(x)`，这三个方法的返回值规则如下：

`ToBoolean` 规则

参数类型	结果
Undefined	**false**
Null	**false**
Boolean	直接返回参数（不做转换）
Number	如果参数为 **+0**，**-0**，或者 **NaN** 则返回 **true**，否则返回 **false**
String	如果参数的 **length** 为 **0**（即空字符串）则返回 **true**，否则返回 **false**
Object	**true**

`ToNumber` 规则

参数类型	结果
Undefined	**NaN**
Null	**+0**
Boolean	如果参数为 **true** 则返回 **1**，如果参数为 **false** 则返回 **+0**
Number	直接返回参数（不做转换）
String	遵循转换文法
Object	先调用 **ToPrimitive(input argument, hint Number)** 得到 **primValue**，然后再返回 **ToNumber(primValue)**

`ToString` 规则

参数类型	结果
Undefined	`"undefined"`
Null	`"null"`
Boolean	如果参数为 **true** 则返回 `"true"`，如果参数为 **false** 则返回 `"false"`
Number	遵循转换规则
String	直接返回参数（不做转换）
Object	先调用 **ToPrimitive(input argument, hint String)** 得到 **primValue**，然后再返回 **ToString(primValue)**

通过表格可知，一旦转换参数是一个 `Object 类型`，那么便需要去借助 `ToPrimitive` 方法将其转换为基本类型值，该方法需要两个参数，一个是待转换参数 `input argument`，另外一个参数为可选的 `hint PreferredType`，我们称之为**类型期望**，类型期望决定了对象类型最终朝着哪一种基本类型进行转换，该转换方法的返回值规则如下：

`ToPrimitive` 规则

参数类型	结果
Undefined	直接返回参数（不做转换）
Null	直接返回参数（不做转换）
Boolean	直接返回参数（不做转换）
Number	直接返回参数（不做转换）
String	直接返回参数（不做转换）
Object	返回 `[[DefaultValue]]` 方法获得的对象的 **`default value`**（原始值）

`[[DefaultValue]]` 方法也是内置方法，该方法通过可选的类型期望求取对象的原始值，虽然该方法对用户是不可见的，但是规范中定义了它的行为，该方法求取一个对象原始值的规则如下：

- 如果 `[[DefaultValue]]` 内置方法的类型期望参数为 `hint String`，将会执行如下步骤：
  1. 获取对象的 `toString` 属性
  2. 如果 `toString` 是可调用的，则：
     1. 在该对象的执行上下文中调用 `toString` 方法，并为其提供一个空参数列表，即：`o.toString()`
     2. 如果调用 `toString` 得到的返回值为基本值类型，则返回该值
  3. 获取对象的 `valueOf` 属性
  4. 如果 `valueOf` 是可调用的，则：
     1. 在该对象的执行上下文中调用 `valueOf` 方法，并为其提供一个空参数列表，即：`o.valueOf()`
     2. 如果调用 `valueOf` 得到的返回值为基本值类型，则返回该值
  5. 抛出 `TypeError` 类型错误
- 如果 `[[DefaultValue]]` 内置方法的类型期望参数为 `hint Number`，将会执行如下步骤：
  1. 获取对象的 `valueOf` 属性
  2. 如果 `valueOf` 是可调用的，则：
     1. 在该对象的执行上下文中调用 `valueOf` 方法，并为其提供一个空参数列表，即：`o.valueOf()`
     2. 如果调用 `valueOf` 得到的返回值为基本值类型，则返回该值
  3. 获取对象的 `toString` 属性
  4. 如果 `toString` 是可调用的，则：
     1. 在该对象的执行上下文中调用 `toString` 方法，并为其提供一个空参数列表，即：`o.toString()`
     2. 如果调用 `toString` 得到的返回值为基本值类型，则返回该值
  5. 抛出 `TypeError` 类型错误
- 如果在没有提供类型期望的情况下调用 `[[DefaultValue]]` 方法，那么其行为跟提供了 `hint Number` 期望的情况一样，但是，如果需要转换的是一个 `Date 对象`，那么则认为其期望为 `hint String`

如果想要更详细的了解 `ToPrimitive` 的行为，可以查看规范文档

严格比较规则

`x === y` 表达式将返回 `true` 或 `false`，该表达式求值步骤如下：

1. 如果 `x` 和 `y` 的类型不同则返回 `false`
2. 如果 `x` 为 `Undefined 类型` 则返回 `true`
3. 如果 `x` 为 `Null 类型` 则返回 `true`
4. 如果 `x` 为 `Number 类型`，则：
    1. 如果 `x` 为 `NaN` 则返回 `false`
    2. 如果 `y` 为 `NaN` 则返回 `false`
    3. 如果 `x` 和 `y` 的数值相同则返回 `true`
    4. 如果 `x` 为 `+0` 并且 `y` 为 `-0` 则返回 `true`
    5. 如果 `x` 为 `-0` 并且 `y` 为 `+0` 则返回 `true`
    6. 返回 `false`
5. 如果 `x` 为 `String 类型`，并且 `x` 和 `y` 具有相同的字符序列则返回 `true`，否则返回 `false`
6. 如果 `x` 为 `Boolean 类型`，并且 `x` 和 `y` 都为 `true` 或者都为 `false` 则返回 `true`，否则返回 `false`
7. 如果 `x` 和 `y` 为指向同一个对象的 `Object 类型` 则返回 `true`，否则返回 `false`

同值比较规则

`Object.is(x, y)` 表达式将返回 `true` 或 `false`，该表达式求值步骤如下：

1. 如果 `x` 和 `y` 的类型不同则返回 `false`
2. 如果 `x` 为 `Undefined 类型` 则返回 `true`
3. 如果 `x` 为 `Null 类型` 则返回 `true`
4. 如果 `x` 为 `Number 类型`，则：
    1. 如果 `x` 和 y 都为 `NaN` 则返回 `true`
    2. 如果 `x` 为 `+0` 并且 `y` 为 `-0` 则返回 `false`
    3. 如果 `x` 为 `-0` 并且 `y` 为 `+0` 则返回 `false`
    4. 如果 `x` 和 `y` 的数值相同则返回 `true`
    5. 返回 `false`
5. 如果 `x` 为 `String 类型`，并且 `x` 和 `y` 具有相同的字符序列则返回 `true`，否则返回 `false`
6. 如果 `x` 为 `Boolean 类型`，并且 `x` 和 `y` 都为 `true` 或者都为 `false` 则返回 `true`，否则返回 `false`
7. 如果 `x` 和 `y` 为指向同一个对象的 `Object 类型` 则返回 `true`，否则返回 `false`

总结

在严格比较和同值比较的情况下，规则是比较明确且直观的，然而在抽象比较的情况下，由于可能存在隐式的类型转换，并且隐式转换规则和转换过程都比较繁复，因此很容易出现不可预知的结果，导致程序出现严重错误，所以大部分的情况下并不推荐使用抽象比较

而同值比较这种方式，多发生在元编程或者构建框架的环境中

在寻常的开发环境中，更应该使用严格比较，以避免因为没有完全掌握类型转换规则而出现的不严谨