《前端高手进阶》

朱德龙 前中兴软创主任工程师

— 拉勾教育出品 —

第07讲：关于 JavaScript 的数据类型你知多少？

JavaScript 的数据类型

- 空（Null）
- 未定义（Undefined）
- 数字（Number）
- 字符串（String）
- 布尔值（Boolean）
- 符号（Symbol）
- 对象（Object）

JavaScript 的数据类型

基础类型的数据在被引用或拷贝时是值传递

也就是说会创建一个完全相等的变量

引用类型只是创建一个指针指向原有的变量

实际上两个变量是“共享”这个数据的

并没有重新创建一个新的数据

Undefined

只有一个值 **undefined**

- 引用已声明但未初始化的变量
- 引用未定义的对象属性
- 执行无返回值函数
- 执行 void 表达式
- 全局常量 window.undefined 或 undefined

Undefined


```
var a; // undefined
var o = {}
o.b // undefined
(() => {})() // undefined
void 0 // undefined
window.undefined //
undefined
```

Undefined


```
x>0 && x<5 ? fn() : void 0;
```

Undefined

如何判断一个变量的值是否为 undefined 呢？

- 直接通过**逻辑取非**操作来将变量 x 强制转换为布尔值进行判断
  - 通过 3 个等号将变量 x 与 undefined 做**真值比较**
- 通过 typeof 关键字获取变量 x 的类型，然后与 'undefined' 字符串做**真值比较**

Undefined

```
// 方式1
if(!x) {
  ...
}
// 方式2
if(x===undefined) {
  ...
}
// 方式2
if(typeof x === 'undefined')
{
  ...
}
```

Null

只有唯一的一个值 null，都可以表示空值

甚至我们通过 “==” 来比较它们是否相等的时候得到的结果都是 true

但 null 是 **JavaScript 保留关键字**

而 undefined 只是一个常量

我们可以声明名称为 undefined 的变量（虽然只能在老版本的 IE 浏览器中给它重新赋值）

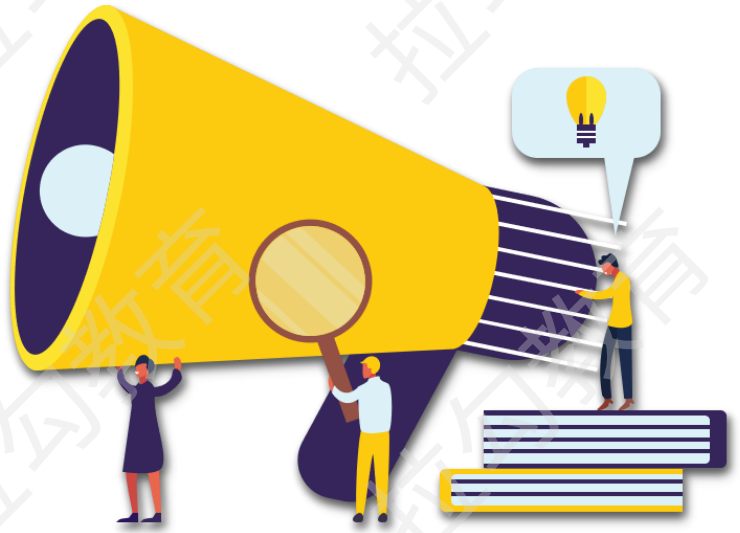
但将 null 作为变量使用时则会报错

Boolean

Boolean 数据类型有两个值：**true** 和 **false**

注意

不要将各种表达式和变量转换成 Boolean 数据类型来当作判断条件



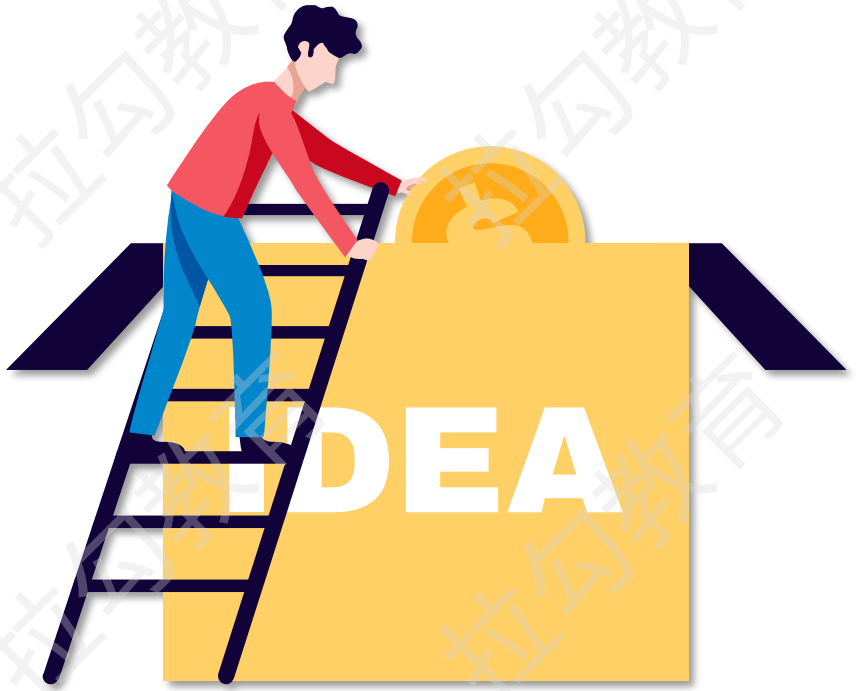
Boolean


```
function getWeek(week){
  const dict = ['日', '一', '二', '三', '四', '五', '六'];
  if(week) return `??${dict[week]}`;
}
```

Number

- **NaN（Not a Number）** 通常在计算失败的时候会得到该值要判断一个变量是否为 NaN 则可以通过 Number.isNaN 函数进行判断

- **Infinity** 是无穷大，加上负号 “-” 会变成无穷小
  在某些场景下比较有用
  比如通过数值来表示权重或者优先级
  Infinity 可以表示最高优先级或最大权重



Number

进制转换

```
['0', '1', '2'].map(parseInt) // [0, NaN, NaN]
```

Number

进制转换

```
(10).toString(2) // "1010"
```

Number

精度问题

```
0.1 + 0.2 // 0.30000000000000004
```

Number

精度问题

```
Math.pow(Math.pow(5, 1/2), 2) // 5.000000000000001
```

Number

精度问题

```
parseFloat((0.1 + 0.2).toPrecision(12)) // 0.3
```

String

笔试题

千位分隔符是指为了方便识别较大数字

每隔三位数会加入 1 个逗号，该逗号就是千位分隔符

如果要编写一个函数来为输入值的数字添加千分位分隔符

该怎么实现呢 ？

String


```
function sep(n){
  let [i, c] = n.toString().split(/(\.\d+)/)
  return i.split('').reverse().map((c, idx) => (idx+1) % 3 === 0 ?
',' + c : c).reverse().join('').replace(/^,/, '') + c
}
```

String


```
function sep2(n){
 let str = n.toString()
 str.indexOf('.') < 0 ? str+= '.' : void 0
 return str.replace(/(\d)(?=(\d{3})+\.)/g, '$1,').replace(/\.$/, '')
}
```

Symbol

Symbol 是 ES6 中引入的新数据类型，它表示一个唯一的常量

```
var a = Symbol('1')
var b = Symbol(1)
a.description === b.description // true
var c = Symbol({id: 1})
c.description // [object Object]
var _a = Symbol('1')
_a == a // false
```

Symbol

避免常量值重复

```
function getValue(key) {
 switch(key){
  case 'A':
   ...
   ...
  case 'B':
   ...
 }
}
getValue('B');
```

Symbol

避免常量值重复

```
const KEY = {
  alibaba: 'A',
  baidu: 'B',
  ...
}
function getValue(key) {
  switch(key){
    case KEY.alibaba:
      ...
    ...
    case KEY.baidu:
      ...
  }
}
getValue(KEY.baidu);
```

Symbol

避免常量值重复

```
const KEY = {
  alibaba: 'A',
  baidu: 'B',
  ...
  bytedance: 'B'
}
```

Symbol

避免常量值重复

```
getValue(KEY.baidu) // 等同于 getValue(KEY.bytedance)
```

Symbol

避免常量值重复

```
const KEY = {
  alibaba: Symbol(),
  baidu: Symbol(),
  ...
  bytedance: Symbol()
}
```

Symbol

避免对象属性覆盖

假设有这样一个函数 fn

需要对传入的对象参数添加一个临时属性 user

但可能该对象参数中已经有这个属性了

如果直接赋值就会覆盖之前的值

此时就可以使用 **Symbol** 来避免这个问题

Symbol

避免对象属性覆盖

```
function fn(o) { // {user: {id: xx, name: yy}}
  const s = Symbol()
  o[s] = 'zzz'
  ...
}
```

补充：类型转换

什么是类型转换？

JavaScript 这种弱类型的语言

相对于其他高级语言有一个特点

那就是在处理不同**数据类型运算**或**逻辑操作**时

会强制转换成同一数据类型

补充：类型转换

	String	Number	Boolean
undefined	"undefined"	NaN	false
null	"null"	0	false
String	-	对应的数值或 NaN	除空字符串外都为 true
Number	对应的字符串	-	除 0 外都为 true
Boolean	"true" 或 "false"	1 或 0	-
Symbol	不可转换	不可转换	true

- 把基本类型的数据换成对应的对象过程称之为“**装箱转换**”
- 把数据对象转换为基本类型的过程称之为“**拆箱转换**”

补充：类型转换


```
var n = 1
var o = new Number(n) //显式装箱
o.valueOf() //显式拆箱
n.toPrecision(3) // 隐式装箱, 实际操作：var tmp = new Number(n);tmp.toPrecision(3);tmp = null;
o + 2 // 隐式拆箱，实际操作:var tmp = o.valueOf();tmp + 2;tmp = null;
```

补充：类型转换

会触发隐式地类型转换的操作作

- 运算相关的操作符包括 +、-、+=、++、* 、/、%、<<、& 等
- 数据比较相关的操作符包括 >、<、== 、<=、>=、===
- 逻辑判断相关的操作符包括 &&、!、||、三目运算符

Object

- 简单地说，Object 类型数据就是**键值对的集合**

  键是一个字符串（或者 Symbol），值可以是任意类型的值

- 复杂地说，Object 又**包括很多子类型**

  比如 Date、Array、Set、RegExp

Object

- 由于引用类型在赋值时只传递指针，这种拷贝方式称为**浅拷贝**
- 而创建一个新的与之相同的引用类型数据的过程称之为**深拷贝**

通过等号“=”赋值只是浅拷贝

要实现真正的拷贝操作则需要通过遍历键来赋值对应的值

这个过程中如果遇到 Object 类型还需要再次进行遍历

Object


```
[undefined, null, true, '', 0, Symbol(), {}].map(it => typeof it)
// ["undefined", "object", "boolean", "string", "number", "symbol", "object"]
```

Object

```
function clone(data) {
  let result = {}
  const keys =
[...Object.getOwnPropertyNames(data),...Object.getOwnPropertySymbols(data)]
  if(!keys.length) return data
  keys.forEach(key => {
    let item = data[key]
```

Object

```
  let item = data[key]
  if (typeof item === 'object' && item) {
    result[key] = clone(item)
  } else {
    result[key] = item
  }
})
```

Object


```
    result[key] = clone(item)
  } else {
    result[key] = item
  }
 })
 return result
}
```

Object

在 clone 函数中有没有可能出现无限递归调用呢？

当对象数据嵌套的时候可能出现

Object


```
var a = {}
var b = {}
a.b = b
b.a = a
```

Object

把已添加的对象记录下来

这样下次碰到相同的对象引用时

直接指向记录中的对象即可

要实现这个记录功能

我们可以借助 ES6 推出的 **WeakMap** 对象

该对象是一组键/值对的集合

其中的键是弱引用的

其键必须是对象

而值可以是任意的

Object


```
function clone(obj) {
  let map = new WeakMap()
  function deep(data) {
    let result = {}
    const keys =
[...Object.getOwnPropertyNames(data), ...Object.getOwnPropertySymbols(data)]
    if(!keys.length) return data
    const exist = map.get(data)
    if (exist) return exist
```

Object

```
if (exist) return exist
map.set(data, result)
keys.forEach(key => {
 let item = data[key]
 if (typeof item === 'object' && item) {
  result[key] = deep(item)
 } else {
  result[key] = item
 }
```

Object

```
      result[key] = deep(item)
    } else {
      result[key] = item
    }
  })
  return result
}
return deep(obj)
}
```

总结

深入理解了 JavaScript 的 6 种**基础数据类型**

和 1 种**引用数据类型**

熟知6 种基础数据类型之间的转换关系

引用类型则重点讲了如何深拷贝一个对象

总结

拉勾教育
互联网人实战大学

你能否写出一个函数来判断两个变量是否相等 ？

Next：第07讲《关于 JavaScript 的数据类型，你知多少？》

拉勾教育

—互联网人实战大学—

下载「拉勾教育App」
获取更多内容

L / A / G / O / U