

判断数据类型的方法

typeof xxx

- 直接在计算机底层作用于数据类型的二进制进行检测
- 检测的数据类型最后以一个字符串类型进行返回
- 特殊情况: typeof NaN => "number"; typeof null => "object"
- typeof null => "object"的原因: 对象在计算机中都是以000开头进行存储的, null的二进制存储正好是000

用于检测基本类型和Function类型是可以的没有任何问题, 但是用于检测普通对象, 数组对象正则对象等都返回对象, 会出现问题

A instanceof B

- 底层机制是用instanceof来检测函数对象B是否存在与实例对象A的原型链上的, 只要出现返回true
- let arr = []; arr instanceof Array => true;但是arr instanceof Object => true
- 但是由于我们可以随意的修改原型链的指向, 而且不能用它来检测基本数据类型, 所以也不准确

```
function _typeof(obj) {
  if (obj === null) return 'null';
  if (obj === undefined) return 'undefined';
  if (obj === true) return 'boolean';
  if (obj === false) return 'boolean';
  if (obj === 0) return 'number';
  if (obj === -0) return 'number';
  if (obj === 1) return 'number';
  if (obj === -1) return 'number';
  if (obj === NaN) return 'number';
  if (obj === Infinity) return 'number';
  if (obj === -Infinity) return 'number';
  if (obj === 0n) return 'bigint';
  if (obj === -0n) return 'bigint';
  if (obj === 1n) return 'bigint';
  if (obj === -1n) return 'bigint';
  if (obj === NaNn) return 'bigint';
  if (obj === Infinityn) return 'bigint';
  if (obj === -Infinityn) return 'bigint';
  if (obj === 0.0) return 'number';
  if (obj === -0.0) return 'number';
  if (obj === 1.0) return 'number';
  if (obj === -1.0) return 'number';
  if (obj === NaN.0) return 'number';
  if (obj === Infinity.0) return 'number';
  if (obj === -Infinity.0) return 'number';
  if (obj === 0.0000000000000001) return 'number';
  if (obj === -0.0000000000000001) return 'number';
  if (obj === 1.0000000000000001) return 'number';
  if (obj === -1.0000000000000001) return 'number';
  if (obj === NaN.0000000000000001) return 'number';
  if (obj === Infinity.0000000000000001) return 'number';
  if (obj === -Infinity.0000000000000001) return 'number';
  if (obj === 0.0000000000000001n) return 'bigint';
  if (obj === -0.0000000000000001n) return 'bigint';
  if (obj === 1.0000000000000001n) return 'bigint';
  if (obj === -1.0000000000000001n) return 'bigint';
  if (obj === NaN.0000000000000001n) return 'bigint';
  if (obj === Infinity.0000000000000001n) return 'bigint';
  if (obj === -Infinity.0000000000000001n) return 'bigint';
  if (obj === 0.0000000000000001) return 'number';
  if (obj === -0.0000000000000001) return 'number';
  if (obj === 1.0000000000000001) return 'number';
  if (obj === -1.0000000000000001) return 'number';
  if (obj === NaN.0000000000000001) return 'number';
  if (obj === Infinity.0000000000000001) return 'number';
  if (obj === -Infinity.0000000000000001) return 'number';
  if (obj === 0.0000000000000001n) return 'bigint';
  if (obj === -0.0000000000000001n) return 'bigint';
  if (obj === 1.0000000000000001n) return 'bigint';
  if (obj === -1.0000000000000001n) return 'bigint';
  if (obj === NaN.0000000000000001n) return 'bigint';
  if (obj === Infinity.0000000000000001n) return 'bigint';
  if (obj === -Infinity.0000000000000001n) return 'bigint';
}
```

手写实现instanceof

constructor

- 每个类的Constructor 指向这个类的构造函数
- 支持基本数据类型的检测
- 可以用它来判断这个类型的之前是一个什么类型, 然后再创建一个和他蕾丝的对象, 即深度拷贝
- 但是Constructor也是可以随意的被改变的, Number.prototype.Constructor = "AA"
- 使用事例: let arr =[];arr.Constructor === Array; => true

Object.prototype.toString.call()

- 标准的检测方法
- Object.prototype.toString这个方法的本来是用来返回当前实例对象所属类的信息

```
let obj = {
  name: 'wang'
}
console.log(obj.toString())// [object Object]
```

this指向的事obj, 所以检测的是obj的信息

所以利用这个方法检测数据类型的底层原理就是: 让Object.protype.toString执行, 并且让它里面的this指向我们需要检测的值, 就可以检测到我们需要检测的值的类型

```
function _typeof(obj) {
  if (obj === null) return 'null';
  if (obj === undefined) return 'undefined';
  if (obj === true) return 'boolean';
  if (obj === false) return 'boolean';
  if (obj === 0) return 'number';
  if (obj === -0) return 'number';
  if (obj === 1) return 'number';
  if (obj === -1) return 'number';
  if (obj === NaN) return 'number';
  if (obj === Infinity) return 'number';
  if (obj === -Infinity) return 'number';
  if (obj === 0n) return 'bigint';
  if (obj === -0n) return 'bigint';
  if (obj === 1n) return 'bigint';
  if (obj === -1n) return 'bigint';
  if (obj === NaNn) return 'bigint';
  if (obj === Infinityn) return 'bigint';
  if (obj === -Infinityn) return 'bigint';
  if (obj === 0.0) return 'number';
  if (obj === -0.0) return 'number';
  if (obj === 1.0) return 'number';
  if (obj === -1.0) return 'number';
  if (obj === NaN.0) return 'number';
  if (obj === Infinity.0) return 'number';
  if (obj === -Infinity.0) return 'number';
  if (obj === 0.0000000000000001) return 'number';
  if (obj === -0.0000000000000001) return 'number';
  if (obj === 1.0000000000000001) return 'number';
  if (obj === -1.0000000000000001) return 'number';
  if (obj === NaN.0000000000000001) return 'number';
  if (obj === Infinity.0000000000000001) return 'number';
  if (obj === -Infinity.0000000000000001) return 'number';
  if (obj === 0.0000000000000001n) return 'bigint';
  if (obj === -0.0000000000000001n) return 'bigint';
  if (obj === 1.0000000000000001n) return 'bigint';
  if (obj === -1.0000000000000001n) return 'bigint';
  if (obj === NaN.0000000000000001n) return 'bigint';
  if (obj === Infinity.0000000000000001n) return 'bigint';
  if (obj === -Infinity.0000000000000001n) return 'bigint';
}
```

封装一个万能的用来检测数据类型的方法