**limit无new实例化**

const limit = function(val){

if( !(this instanceof limit) ){

return new limit(val);

};

this.\_\_value\_\_ = [val];

};

使用：limit(‘aa’)。先通过this instanceof判断是否为实例化出来的对象。如果不是，，则实例化，调用构造函数将val传入this.\_\_value\_\_

limit.each(limit, (val, key) => {

if( limit.isFunction(val) ){

limit.prototype[key] = function(...args){

this.\_\_value\_\_.push( limit[key]( ...[limit.last(this.\_\_value\_\_), ...args] ) );

return this;

};

}else{

limit.prototype[key] = val;

};

});

首先将运行limit[key]并返回 limit对象的函数传入limit.prototype[key]，以实现链式操作。该函数中，将limit[key]的运行结果推入this.\_\_value\_\_。

将limit上一次的运算结果传入下一个函数中，需要确定每一次操作的返回值类型是下一次操作的第一个参数。如：limit([11,22]).each(function(val){console.log(val)});

**Helper函数**

**检查参数**

在format中使用，对参数 (target, …args) / ( target, iterator, ...args)做初始化处理

1. checkTargetNoEqualNull 若 target 为null，改为 {}
2. checkTargetWithArray 控制 target 为 数组
3. checkTargetWithString 控制target 为 字符串
4. checkTargetWithNumber 控制 target 为 数字
5. checkObjFunction 控制 target 为 对象， iterator为函数
6. checkArrFunction 控制 target 为 数组， iterator为函数
7. checkTrueIndex( length, index ) 若index<0，从右往左获取索引
8. checkPositive( num ) 返回正整数
9. checkStringNumber( str, num )

确定 str为字符串，num为正整数。以数组返回 [bool, bool]

10、checkPadArgs ( str, leg, arg ) 字符串，正整数，空白填充字符，以数组返回

11、checkFiniteNum(…args) 判断参数是否均为有限数

12、checkFlattenArgs 将参数平铺为数组

**limit属性：**

1、getProp(obj = O, key = '', deVal)

设置默认属性 obj[key] === void 0 ? deVal : obj[key]

2、log(type, content)

自定义console.log显示

3、limit['T.T'] 相当于 log(error, ...args)

limit[ '!!!'] 相当于 log(warn, ...args)

limit['...'] 相当于 log(log, ...args)

* **判定方法**

4、isElement(o) //o: document.body

是否DOM元素

5、isDocument(o) //o: document

6、isWin

7、isUndefined、isDefined

8、isNull

9、isFunction

10、isBoolean

11、isString, Number, Array, Date, RegExp, Error, Math

12、isObject // typeof null === ‘object’，通过 !!null 排除

函数也作为对象

13、isArguments //通过是否包含 ‘callee’ 属性判断

是否是函数参数

14、isArrayLike

含length属性的对象

15、isNaN => priority: Number.isNaN ( ES6 )

是否为NaN，与传统isNaN区别（传统isNaN字符串均为true）：

isNaN( ‘aa’ ) == true, Number.isNaN( ‘aa’ ) == false

16、isFinite => priority: Number.isFinite ( ES6 )

是否有限，与传统isFinite区别（传统isFinite字符串均转换为数值后判断）：

isFinite(‘123’) == true, Number.isFinite(‘123’) == false

17、isInteger => priority: Number. isInteger ( ES6 )

是否为整数，有限的数值，且 Math.floor(num) == num

18、isSafeInteger => priority: Number. isInteger ( ES6 )

limit.isInteger(n) && -9007199254740992 < n && n < 9007199254740992

19、isEmpty

null，undefined，NaN，[]， {}，数字，‘’ 均为true

有内容的字符串、数组、对象 均为 false

bug：{a: 1, length:1} 带length属性的对象，在判断isArrayLike时有分歧。

导致toArray结果为[]，从而判断为true。

toArray已通过limit.from修正

* **工具方法**

20、getUid() 设置版本号 0.0.0 每次调用自动加1

21、has(key) hasOwnProperty

22、is(a, b) 判断两个参数全等于，兼容NaN

23、keys 兼容ES5：Object.keys()，keysFixed => forin 循环将可枚举key置入数组

24、keysSuper 优先ES5/6：Object.getOwnPropertyNames 获取可枚举及不可枚举key

兼容方法同样调用了keysFixed ，功能与limit.keys 相同

25、forin(obj, iterator, context)

for(let key in obj){ iterator.call(context, obj[key], key, obj) }

功能同\_loop，没有break

26、\_getLoopKey(obj) （需调整优化）

取键值，调用了keys方法，对于 isArrayLike的对象，去掉length属性后取键值

27、\_loop(obj, iterator, context, isBreak, begin)

通过\_getLoopKey从keys数组的begin位置开始循环调用 iterator.call(context, obj[key], key, obj)，根据返回值true/false及isBreak，break。

begin未定义的情况下，~~undefined，~~null == 0

调用 \_loop 的方法，均支持object对象

28、each(obj, iterator, context)

与forin区别，因为调用\_loop，故没有遍历类数组对象的 length 属性

29、cb(callback)

判断callback是函数则返回，不是则返回function(){}

30、assign(target, …args)

兼容 es6: Object.assign()，对象合并。相同的属性名，后面对象的属性会覆盖前面的

调用两层each 忽略对象中的length属性

应用 assign({}, {a: 1, b: 2}, {c: 3, d: 4})，将 {a: 1, b: 2}, {c: 3, d: 4} 合并到 {} 中

31、assignSuper

与assing区别：仅将已定义的属性，合并到新的对象

应用 assign({}, {a: undefined, b: 2}, {c: 3, d: 4})，将 {b: 2}, {c: 3, d: 4} 合并到 {} 中

32、extend

与assign区别：调用each=>forin。可将length属性合并到新对象

33、extendSuper

与extend区别：仅将已定义的属性，合并到新的对象

34、values(target)

兼容es6: Object.values()，返回对象value的数组

35、entries(target)

兼容es6: Object. entries()，将对象可枚举的属性，转换为键值对的数组

如 {a:1, b:2} 转换为 [[‘a’, 1], [‘b’, 2]]

36、getValueInObject(obj, …arg)

获取对象中的值，应用：limit.getValueInObject({a:{c:3}}, 'a', 'c') 返回 3

37、size(obj)

根据\_getLoopKey获取对象的长度。

注：该对象存在一些问题，是由于\_getLooKey中调用toArray方法引起

说明：{0: 'a1', 1: 'b1', length:4}

对toArray未修复前，转换为 ['a1', 'b1']，返回2

按照下面对toArray修复后，转换为 ['a1', 'b1', undefined, undefined]，返回4

**PS：个人认为size的使用应该返回原始对象属性个数，不依赖于toArray 方法。**

**或者可将现有的size命名为length**

* **数组**

38、toArray

通过Array.prototype.slice.call将类数组对象，转换为数组。

只能转换格式为：{0: ‘a1’, 1: ’b1’, length:2}

当前对 {a: ‘a1’, b: ’b1’, length:2} 的转换会报错，返回 [ ] 空数组

可通过添加 limit.from() 纠正，返回[undefined, undefined]。该方法有兼容问题，因为在limit.from的fixed中又调用了limit.toArray。尚需优化

let arr;  
try{  
      arr = slice.call(obj);  
}catch(e){  
      arr = sliceFix(obj);  
};  
return limit.from(arr);

39、flatten

将参数平铺为一个数组

flatten(1,2,[3,4],[5,[6,[7,8]]]) 转换为 [1, 2, 3, 4, 5, 6, 7, 8]

40、contains(arr, target)

通过some循环，limit.is函数处理 NaN， +-0 等问题。

支持对象，如 limit. contains({a:'a1', b:'b1', c:'c1'}, 'b1') 返回true

注：因为is方法中 +0 != -0。故limit.contains([+0], -0) 返回 false

41、includes (arr, target, index)

兼容：ES6 [].includes

与contains区别：

① arr可以是字符串

② limit.includes([+0], -0) 返回true。

因为fixed中非NaN的情况下通过indexOf来判断是否包含元素。

42、indexOf(arr, ele, formIndex)

兼容：ES5 [].indexOf

通过 \_loop 循环判断 val 与 ele是否相等，并返回 key。

注：

① 因为 \_loop 中调用了\_getLoopKey将类数组对象或普通对象转换成了keys数组来做循环处理，故该方法中的arr支持使用对象

② when方法中 indexOf && arr.indexOf === indexOf 做了两件事情：

1. 判断了Array.prototype.indexOf 是否存在
2. 判断了当前的arr对象是否为数组，若不是数组而是对象，则调用 fixed方法

43、indexOfSuper(arr, ele, formIndex)

与limit.indexOf区别：没有设置priority，直接执行fixed方法。用于arr为对象的情况

44、union(arr)

数组元素去重，如 [+0, -0, 1, 2, 1, 5, NaN, 'NaN', NaN, 3] 返回 [0, -0, 1, 2, 5, NaN, "NaN", 3]

因为调用了 contains 方法，故 +0和 -0 被作为不同的元素

45、unionSuper(arr)

先做排序，再遍历数组，前后两个数比较是否相等，避免了循环嵌套

该方法在排序的时候有问题。NaN, ‘NaN’, NaN以及+-0无法正确排序，故需慎重使用

46、whiteList(arr, ...args)、blackList

带limit.flatten，感觉与 whiteBlack方法有冲突，即无法执行两次数组循环。

或者需要修正whiteBlack方法

以下为去掉limit.flatten(args)后的用法。

limit.whiteList([[1,2],[2,3]], [1,2],[3,4],[5,6])，返回 [1,2]。

即arr为白名单数组，对后面的数组参数进行过滤

当中使用了whiteBlack判断是否包含指定数组，用法为whiteBlack([[1,2],[3,4]], [3,4])返回true

**PS：白名单的使用应该为 limit.whiteList([1,2,3], 1,2,5) 返回 1,2。嵌套数组过于繁琐**

47、difference(arr, …args)

用法：limit.difference([1,2,3], 1,4,5) 返回 [2, 3]

即对arr进行过滤，返回args中不包含的元素

48、remove (arr, tar, formIndex)

支持remove([1,2,3], 2) 或 remove({a:'a1', b:'b1', c:'c1'}, 'b1')。仅删出第一个等于tar的值

49、removeAll(arr, tar)

将arr中所有等于tar的值都删除

如 limit.removeAll({a:'a1', b:'b1', c:'c1', d: 'b1'}, 'b1') arr变为{ a:'a1', c:'c1'}

50、first(obj)

获取对象的第一个值

51、last(obj)

返回数组或对象中的最后一个元素

52、forEach (arr, iterator, context)

兼容：ES5 forEach。调用了limit.each 方法

支持写法

limit.forEach([1,2,3], function(val, index){})

limit.forEach({0:'a1', 1:'b1', 2:'c1', length:3}, function(val, index){})

limit.forEach({a:'a1', b:'b1', c:'c1'}, function(val, key){})

53、map (arr, iterator, context)

支持写法同上，返回重新设置的数组或对象

54、filter (arr, iterator, context)

55、some (arr, iterator, context)

56、every (arr, iterator, context)

57、lastIndexOf(arr, ele, fromIndex)

先通过arr.reverse()将元素顺序颠倒。然后获取index，再做 length - index处理

58、reduce(arr, iterator, init)

判断是否有init参数，没有的话将初始值设置为arr[0]，同时从index=1开始\_loop遍历循环。

使用：limit.reduce([1,2,3], function(pre, cur, index, arr){})

arr 可使用数组 或 类数组对象

59、reduceRight(...args)

将args[0] 数组反转，将args[1] 函数中的index参数做maxIndex – index处理。

再将新参数args传入reduce

60、from(obj, iterator, context)

兼容ES6: Array.from()

将带length属性的类数组对象通过toArray转换为数组。再返回改数组的map方法

61、of(…args)

兼容ES6：Array.of ——将参数转换为数组

fixed：[].slice.call(args)

62、find (arr, iterator, context) 【支持object】

兼容ES6：Array.find ——返回第一个匹配的数组元素

fixed：通过some循环遍历，当iterator返回true时，返回当前的value值，并退出循环

63、findIndex (arr, iterator, context) 【支持object】

兼容ES6：Array.findIndex ——返回第一个匹配的数组元素的索引

fixed：通过some循环遍历，当iterator返回true时，返回当前的key值，并退出循环

64、fill (arr, target, start, end)

兼容ES6：Array.fill

fixed：先对start、end格式化，len=end-start。

再通过unshift生成用于splice的参数，即 [start, len, target\*len]

65、copyWithin (arr, target, start, end)

兼容ES6：Array.copyWithin

fixed：先通过slice复制arr中start到end的元素，再通过unshift添加target、length

生成用于splice的参数，即[target, len, sliceItem]

**函数**

66、bind

兼容ES6：bind。

原生bind的使用：fn.bind(context)(args) 或者 fn.bind(context, args)()

fixed：原理先返回return Function.call.apply(fn, [context, args])的函数，即在bind后不马上执行，通过调用返回的函数再执行

使用：var fn2 = limit.bind(fn, context) 返回this指向context的函数，

fn2(…args) args为fn所需的参数

或者limit.bind(fn, context, …args)()

67、compose

目的：compose 组合 a(b(c())) => (a(), b(), c())，即将原本需要根据返回值作为参数的嵌套写法展开书写，使代码更清晰

使用：limit.compose(a(), b(), c())(…args)，args为c()的参数

优化：c().b().a()，更清晰直观

**68、promise**

使用：new Promise(function(resolve, reject));

在构造函数中将resolve函数和reject函数作为参数传入以上实例化的第一个function参数中。resolve（reject）中，循环读取this.promiseList，若当中包含promise对象，则修改promise对象的状态为resolved（rejected），value值为传入的参数。

然后调用\_clean方法，在该方法中判断this.Stack中是否有值，并执行this.Stack中的方法。this.Stack用于保存success、fail执行的函数。

因为resolve指向的success函数依赖Stack中是否有值，而Stack的元素是在then中置入，故promise实例化时的resolve方法具体实现即 success ，会在then之后调用

执行过程：

1. 在实例化的过程中，通过构造函数会执行function(resolve, reject)中的方法
2. 实例化时resolve/reject的执行有两种情况：
3. 构造函数的中resolve方法立即执行

①在resolve方法中会修改promisList队列中的promise对象 [this] 的状态为resolved。该this. PromiseStatus在then中使用，故需要each([this].concat())

②调用 \_clean方法，当前this.Stack为空，故跳出。

③执行then方法，在this.promisList队列中通过this. PromiseStatus添加一个状态为resolved的新promise对象，将(success, fail)置入this.Stack中。因为当前状态为resolved，可执行\_clean。

④调用 \_clean方法，this.Stack(shift) 执行success/fail方法。this.Stack为空时，

delete me.cleanStatus

1. resolve方法通过回调或延迟异步执行

①实例化执行到resolve方法之前，会先执行到then方法。在this.promisList队列中添加一个 ('pedding', undefined) 的新promise对象。将 (success, fail) 置入新promise独享的Stack。

②延迟执行到resolve方法。each([this].concat(this.promiseList), function(){})，在执行this时，因为Stack.length == 0，跳过。执行then中添加的promise，对promiseValue 赋值即resolve(value)中的参数，调用该promise中的 \_clean循环执行当中Stack里的success方法。Stack为空时，delete me.cleanStatus

Promise.all(promisesArr).then(vals=>{}).catch(e=>{})

promisesArr中的所有promise都执行通过时，将其所有返回值组成数组，传入并调用then中的success方法。

**字符串**

69、toString(obj)

参数为obj返回警告，null返回空字符串

70、trim(str)

正则匹配头尾空格，替换成 ''

80、repeat(str, num)

兼容ES6：str重复num次

fixed：先生成包含num个空元素的数组，通过limit.from将当中的元素转换为str

81、padStart/padEnd (str, len, padStr)

兼容ES6：用padStr填充str，使str长度达到len

fixed：如果str.length>len，则返回原字符串。否则先计算填充padStr的次数，然后通过repeat填充后，再通过slice截取len长度

82、startsWith(str, argStr, index)

兼容ES6：判断是index位置的字符串是否为argStr

fixed：判断str.indexOf(argStr, index) === index。

原代码中需添加index = index || 0; 或index = ~~index

83、endsWith(str, argStr, index)

兼容ES6：判断到index位置的字符串，最后一位是否为argStr

fixed：当index>str.length时原方法返回结果与es6不一致。修改如下：

index = arguments.length === 3 ? ~ ~index : str.length;

var newStr = str.slice(0, index),

endStr = newStr.slice(-arg.length);

return endStr === arg;

84、codePointAt(str, index)

兼容ES6：能够正确处理4个字节储存的字符，返回一个字符的码点

fixed：

(1) 通过charAt获取str前后两个字节的字符码点。

(2) 转换为二进制字符串后分别取最后10位拼在一起。

(3) 转换为数字加上 0x10000

85、fromCodePoint(code)

兼容ES6：将4个字节的码点转换为字符串

fixed：

(1) 原始码点转换为数字，减去 0x10000。

(2) 高位originH通过右移10位获取，低位originL用 & 1023获取。

(3) 然后分别通过 | 相应的数字返回最后16位的高位和低位，转换为16位字符串后返回。

(4) 将高低位拼成一个字符串，通过new Function('return "' + code + '"')() 返回

**数字**

86、toNumber(obj)

null转为0，date转为date.getTime()，boolean转为0/1，数字字符串转为数字

87、random(from, to)

返回大于from，小于to的随机数

88、toFixed (num, scale)

修改小数点的位数为scale。原理：先小数点右移scale位做四舍五入，再左移scale位，以保证小数位数。

movePoint (num, scale)：通过num.split(‘.’) 获取到leftStr，rightStr

movePointLeft(sign, leftStr, rightStr, scale)：

scale<leftStr.length，则leftStr.slice(0, -scale) + '.' + leftStr.slice(-scale)+ rightStr。

否则，'0.' + limit.padStart(leftStr, scale, '0') + rightStr

movePointRight (sign, leftStr, rightStr, scale)：

scale< rightStr.length，则leftStr+rightStr.slice(0, scale) + '.' + rightStr.slice(-scale)。

否则，lefttStr+ limit.padStart(rightStr, scale, '0')

89、thousandSeparator (num, med = 2)

先通过toFixed对num添加小数点，然后通过正则replace对整数部分添加逗号

正则：/(\d{1,3})(?=(\d{3})+$)/g;

89、unThousandSeparator(str)

去掉数字字符串中的逗号，然后转换为数字

90、plus,+(…args)

多个带小数的数字相加，先将小数点右移作为整数递归相加，再左移恢复为小数。以修复浮点数运算可能出现的误差。

getMaxScale 获取参数中，最大的小数位数，如getMaxScale (111.111, 22.22) 返回3

bug：let maxScale = getMaxScale(args); 需改为 let maxScale = getMaxScale(...args); 即参数数组展开后传入getMaxScale。因为getMaxScale的入参为(...args)，又包装了一次参数数组。

91、minus,- (…args)

多个带小数的数字相减，先将小数点右移作为整数递归相减，再左移恢复为小数。

92、multiply,\* (…args) except,/

小数乘法、除法。args传入getNeedNum后，对前两个参数进行操作返回num，然后splice掉args的前两个数用num代替。再判断args长度，大于1的话，递归。

getNeedNum：先取args的前两个元素，及这两个元素的小数长度。然后对两个元素去掉小数点后相乘，再移动小数位数。（乘法：向左移动两个元素小数位数相加。除法：移动 除数小数位数-被除数小数位数）

93、express,? (exp)

使用：limit.express('-11.11 - 22.22 - 33.33 ') 或 limit.express('-11.11 \* 22.22 \* 33.33 ')

先通过replace 正则匹配，计算前两个数 return limit.express('-33.33 - 33.33')，递归计算

计算顺序，乘除、括号、加减。如果不匹配则直接返回数字

**日期**

94、formatDate (timestamp = new Date().getTime(), formatStr = 'yyyy-MM-dd HH:mm:ss')

通过formatStr.replace(reg, function(…args){}) 对获取到的匹配项循环遍历，执行FUN\_DATAS 中对应的方法。返回替换后的字符串

slice.call(args, 1, -2)，因为args为匹配到的完整字符串，括号中的匹配项，匹配字符串的索引，被匹配字符串formatStr。需要将第一个和最后两个去掉，只取括号中的匹配项来遍历。