前言

`reduce(...)`方法对数组中的每个元素执行一个由您提供的**reducer**函数(升序执行)，将其结果汇总为单个返回值（累计作用）。此方法接受两个参数：`callback(...)`(必选)、initialValue(可选)。`callback(...)`接受4个参数：Accumulator (acc) (累计器)、Current Value (cur) (当前值)、Current Index (idx) (当前索引)、Source Array (src) (源数组)。

注意点：
1、`callback(...)`一般需要返回值；
2、不会改变原数组；
3、在回调函数中修改源数组在遍历数组时会影响值的输出（**内部使用浅拷贝**）；

实现思路

1、先获取初始累计的值（分成两种情况：有提供initialValue || 未提供initialValue）；
2、遍历数组并执行`callback(...)`；
3、返回累计值；

源码实现

```
Array.prototype.myReduce = function(callback, initialValue) {
  if(this === null) {
    throw new TypeError( 'Array.prototype.reduce called on null or undefined' );
  }
  if (typeof callback !== 'function') {
    throw new TypeError( callback + ' is not a function');
  }
  const O = Object(this);
  const lenValue = O.length;
  const len = lenValue >>> 0;
  if(len === 0 && !initialValue) {
    throw new TypeError('the array contains no elements and initialValue is not
provided');
  }
  let k = 0;
  let accumulator;
  // 分成两种情况来获取accumulator
  // 有提供initialValue accumulator=initialValue
  // 没有提供initialValue accumulator=数组的第一个有效元素
  if(initialValue) {
    accumulator = initialValue;
  } else {
    let kPressent = false;
    while(!kPressent && k < len) {
      const pK = String(k);
      kPressent = O.hasOwnProperty(pK);
      if(kPressent) {
        accumulator = O[pK];
      };
      k++;
    }
    if(!kPressent) {
```

```
      throw new TypeError('the array contains error elements');
    }
  }
  // 当accumulator=initialValue时 k=0
  // accumulator=数组的第一个有效元素时 k=1
  while(k < len) {
    if(k in O) {
      // callback一般需要返回值
      accumulator = callback(accumulator, O[k], k, O);
    }
    k++;
  }
  return accumulator;
}
let r = [1,2,3].myReduce(function (prevValue, currentValue, currentIndex, array)
{
  return prevValue + currentValue;
}, 22);
console.log(r);
```

参考链接：

- reduce-mdn
- 官方规范