```
Array.prototype.spliceCustom = function (start, deleteCount) {
    let result = [];

    // 如果一个参数都没有，直接返回空数组，即什么元素都不摘取
    // 不可以使用 start === undefined 来判断是否传参，经测试，splice() 与 splice(undefined) 结果并不相同
    if (arguments.length === 0) {
        return result;
    }

    // 经测试，任何非法的参数都将被重置为 0
    Number.isNaN(Number(start)) && (start = 0);

    // 如果参数为反向下标，进行一次转换
    start < 0 && (start = start + this.length);

    // 确定了参数为正向后，如果下标小于 0，那么下标重置为 0， 如果下标大于 length，那么下标重置为 length
    start < 0 && (start = 0);
    start > this.length && (start = this.length);

    // 不可以使用 deleteCount === undefined 来判断是否传参，经测试，splice(1) 与 splice(1, undefined) 结果并不相同
    arguments.length === 1 && (deleteCount = this.length - start);

    // 经测试，任何非法的参数都将被重置为 0
    Number.isNaN(Number(deleteCount)) && (deleteCount = 0);

    // 如果 deleteCount 小于 0，将被重置为 0
    deleteCount < 0 && (deleteCount = 0);

    // 如果 deleteCount 大于 start 下标之后元素的个数，将被重置为剩余参数个数
    deleteCount > this.length - start && (deleteCount = this.length - start);

    let insertElements = [];

    // 将待插入元素填充到 insertElements 中
    if (arguments.length > 2) {
        for (let i = 2; i < arguments.length; i++) {
            insertElements[i - 2] = arguments[i];
        }
    }

    // 原数组的长度
    let length = this.length;
    // 待插入的元素长度
    let insertLength = insertElements.length;
    // 最终的数组的长度
    let newLength = length + insertLength - deleteCount;

    let slotStart = start;
    let slotEnd = start + insertLength;
    let end = start + deleteCount;

    /**
     * splice 有点类似于拔萝卜填坑一样，分为如下几步
     * 1.将要挖出来的元素提出来 比如 [1, 2, 3, 4, 5]，[1, empty, empty, 4, 5] → [2, 3]
```

```
     * 2.挖完待提取元素后，我们可以看到三部分，(1) 为首，(empty, empty) 为坑，(4, 5) 为尾
     * 接下来就需要考虑坑的大小是否需要调整
     * 坑变大或者变小，就表示尾部向左或者向右移动，最后再用待插入元素放进坑中
     *
     * 比如: [1, 2, 3, 4, 5].splice(1, 2, 99)，该例子，挖走两个，但是只加入了一个元素99，坑变成了一个
     * [1, 2, 3, 4, 5] → [1, empty, empty, 4, 5] → [1, empty, 4, 4, 5] → [1, empty, 4, 5, 5] → [1, 99, 4, 5, 5] → [1, 99, 4, 5]
     *
     * 比如: [1, 2, 3, 4, 5].splice(1, 2, 97,  98,  99)，该例子，挖走两个，但是加入了三个元素，坑变成了三个
     * [1, 2, 3, 4, 5] → [1, empty, empty, 4, 5] → [1, empty, empty, 4, 5, 5] → [1, empty, empty, 4, 4, 5] → [1, 97,98,99, 4, 5]
     */


    for (let i = start; i < end; i++) {
        result[i - start] = this[i];
    }

    if (insertLength > deleteCount) {
        for (let i = newLength - 1; i > slotEnd - 1; i--) {
            this[i] = this[i - insertLength + deleteCount]
        }
    } else if (newLength < length) {
        for (let i = slotEnd - 1; i < newLength; i++) {
            this[i] = this[i + 1];
        }
    }

    for (let i = slotStart; i < slotEnd; i++) {
        this[i] = insertElements[i - slotStart];
    }

    this.length = newLength;

    return result;
}

Array.prototype.spliceCustom = function (start, deleteCount) {
    // 如果一个参数都没有，直接返回空数组，即什么元素都不摘取
    // 不可以使用 start === undefined 来判断是否传参，经测试，splice() 与 splice(undefined) 结果并不相同
    if (arguments.length === 0) {
        return result;
    }

    // 经测试，任何非法的参数都将被重置为 0
    Number.isNaN(Number(start)) && (start = 0);

    // 如果参数为反向下标，进行一次转换
    start < 0 && (start = start + this.length);

    // 确定了参数为正向后，如果下标小于 0，那么下标重置为 0， 如果下标大于 length，那么下标重置为 length
    start < 0 && (start = 0);
    start > this.length && (start = this.length);

    // 不可以使用 deleteCount === undefined 来判断是否传参，经测试，splice(1) 与 splice(1, undefined) 结果并不相同
```

```
    arguments.length === 1 && (deleteCount = this.length - start);

    // 经测试，任何非法的参数都将被重置为 0
    Number.isNaN(Number(deleteCount)) && (deleteCount = 0);

    // 如果 deleteCount 小于 0，将被重置为 0
    deleteCount < 0 && (deleteCount = 0);

    // 如果 deleteCount 大于 start 下标之后元素的个数，将被重置为剩余参数个数
    deleteCount > this.length - start && (deleteCount = this.length - start);

    // 这种方法比较好理解，把所有数据全部拿出来再重新装填

    let left = [];
    let center = [];
    let right = [];
    let insert = [];

    if (arguments.length > 2) {
        for (let i = 2; i < arguments.length; i++) {
            insert[i - 2] = arguments[i];
        }
    }

    for (let i = 0; i < this.length; i++) {
        if (i < start) {
            left[left.length] = this[i];
        } else if (i > start + deleteCount - 1) {
            right[right.length] = this[i];
        } else {
            center[center.length] = this[i];
        }
    }

    this.length = 0;

    for (let i = 0; i < left.length; i++) {
        this[this.length] = left[i];
    }

    for (let i = 0; i < insert.length; i++) {
        this[this.length] = insert[i];
    }

    for (let i = 0; i < right.length; i++) {
        this[this.length] = right[i];
    }

    return center;
}
```