data-structures-algorithms

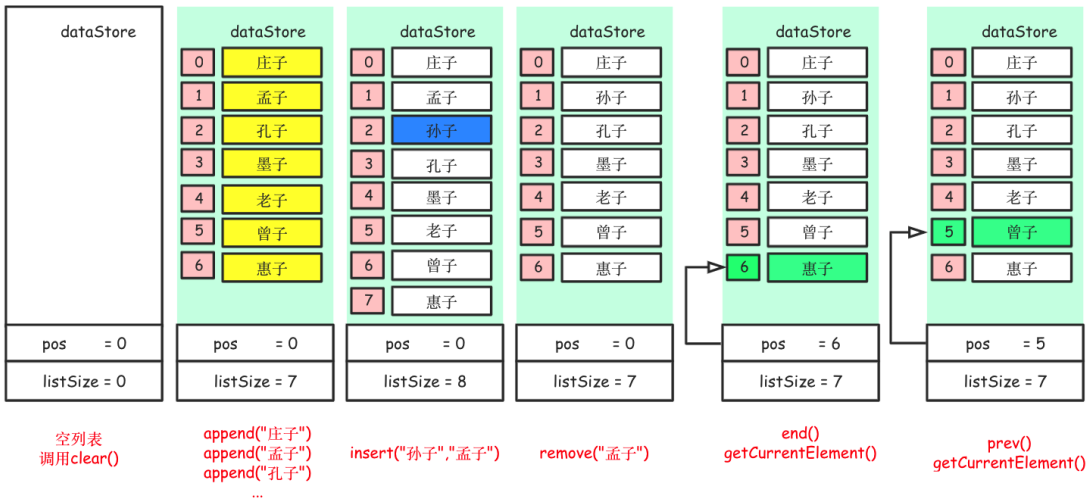
Introduce common data structures and algorithms in JavaScript

常用基础数据结构

List

> 列表(List)的抽象数据类型定义。

数据结构	内部属性	属性说明	核心方法	功能介绍
List 列表	dataStore	内部数据	clear()	清空列表重置指针
	listSize	标识列表的长度	toString()	转换为字符串显示
	pos	标识位置的指针	length()	获取当前列表长度
			start( )	移动指针列表开头
			end( )	移动指针列表末尾
			next( )	操作指针向后移动
			prev( )	操作指针向前移动
			showList( )	显示当前列表信息
			currentPos( )	获取当前的指针位置
			moveTo(position)	移动指针到特定位置
			remove(element)	删除列表中指定元素
			append(element)	在列表末尾追加元素
			insert(element,after)	往列表中间插入元素
			getCurrentElement( )	获取当前指向的元素



列表(List)是一组有序的数据，列表中的数据项称为元素，数据项可以是任意类型。

列表(List)有长度，长度为零的列表称为空列表，列表适用于简单的待办事项列表、购物清单和排行榜等场景，在下面给出列表的 JavaScript 语言实现。

```
class List {
    constructor() {
        this.listSize = 0;
        this.pos = 0;
        this.dataStore = [];
    }
    length() {
        return this.listSize;
    }
    toString() {
        return this.dataStore.join("\n");
    }
    contains(element) {
        let findIndex = this.find(element);
        return findIndex == -1 ? false : true;
    }
    start() {
        this.pos = 0;
    }
    end() {
        this.pos = this.listSize - 1;
    }
    prev() {
        if (this.pos > 0) {
            --this.pos;
        }
    }
    next() {
        if (this.pos < this.listSize - 1) {
            ++this.pos;
        }
    }
    insert(element, after) {
        let insertIndex = this.find(element);
        if (insertIndex != -1) {
            this.dataStore.splice(insertIndex + 1, 0, element);
            ++this.listSize;
            return true;
        }
        return false;
    }
    append(element) {
        this.dataStore[this.listSize++] = element;
    }
    find(element) {
        for (let index = 0; index < this.dataStore.length; index++) {
            const currentElement = this.dataStore[index];
```

```
            if (currentElement == element) {
                return index;
            }
            return -1;
        }
    }
    remove(element) {
        let findIndex = this.find(element);
        if (findIndex != -1) {
            this.dataStore.splice(findIndex, 1);
            --this.listSize;
            return true;
        }
        return false;
    }
    clear() {
        this.dataStore = [];
        this.listSize = this.pos = 0;
    }
    currentPos() {
        return this.pos;
    }
    moveTo(position) {
        if (position >= 0 && position <= this.listSize - 1) {
            this.pos = position;
        }
    }
    getCurrentElement() {
        return this.dataStore[this.pos];
    }
    showList() {
        for (let i = 0; i < this.dataStore.length; i++) {
            console.log(`${this.dataStore[i]}`)
        }
    }
}
```

Queue

> 队列(Queue)的抽象数据类型定义。

数据结构	内部属性	属性说明	核心方法	功能介绍
Queue 队列	dataStore	内部数据	start( )	获取队首元素
	queueSize	队列长度	back( )	获取队尾元素
			clear( )	清理队列内容
			length( )	获取队列长度
			enqueue( )	队尾添加元素

数据结构	内部属性	属性说明	核心方法	功能介绍
			dequeue( )	队首移出元素
			isEmpty( )	队列是否为空
			toString( )	显示队列元素

队列(Queue)是一种列表，是一种先进先出(FIFO)的数据结构。

队列(Queue)用于存储按顺序排序的数据，在操作队列的时候遵循的是先进先出的原则，主要操作是入队(新增元素)和出队(删除元素)，队列也有长度，队列适用于处理多个动画任务的执行、打印任务池等场景，下面给出队列的 JavaScript 语言实现。

```
class Queue {
    constructor(queue) {
        this.dataStore = queue || [];
        this.queueSize = queue.length || 0;
    }
    start() {
        return this.dataStore[0];
    }
    end() {
        return this.dataStore[this.dataStore.length - 1];
    }
    toString() {
        return this.dataStore.join("\n");
    }
    isEmpty() {
        return this.dataStore.length == 0 ? true : false;
    }
    enqueue(element) {
        this.dataStore.push(element);
        ++this.queueSize;
    }
    dequeue() {
        this.dataStore.shift();
        --this.queueSize;
    }
    length() {
        return this.queueSize;
    }
}
```

Stack

栈(Stack)的抽象数据类型定义

数据结构	内部属性	属性说明	核心方法	功能介绍

数据结构	内部属性	属性说明	核心方法	功能介绍
Stack 栈	dataStore	内部数据	push( )	元素入栈操作
	top	栈顶指针	pop( )	元素出栈操作
			peek( )	获取栈顶元素
			clear( )	清理栈的内容
			length( )	获取栈的长度

```
class Stack {
    constructor() {
        this.top = 0;
        this.dataStore = [];
    }
    pop() {
        this.dataStore[--this.top];
    }
    push(element) {
        this.dataStore[this.top++] = element;
    }
    peek() {
        return this.dataStore[this.top - 1];
    }
    clear() {
        this.top = 0;
        this.dataStore = [];
    }
    length() {
        return this.top;
    }
}
```

Set

Dictionary

Tree

博客文章

数据结构与算法 [01]-数组、链表和选择排序

数据结构与算法 [02]-栈、队列和递归

数据结构与算法 [03]-栈的实现和应用