项目说明文档

操作系统

——内存管理之动态内存分配

作 者 姓 名： 崔宸睿

学 号： 2152614

指 导 教 师： 张惠娟

学院、 专业： 软件学院 软件工程

同济大学

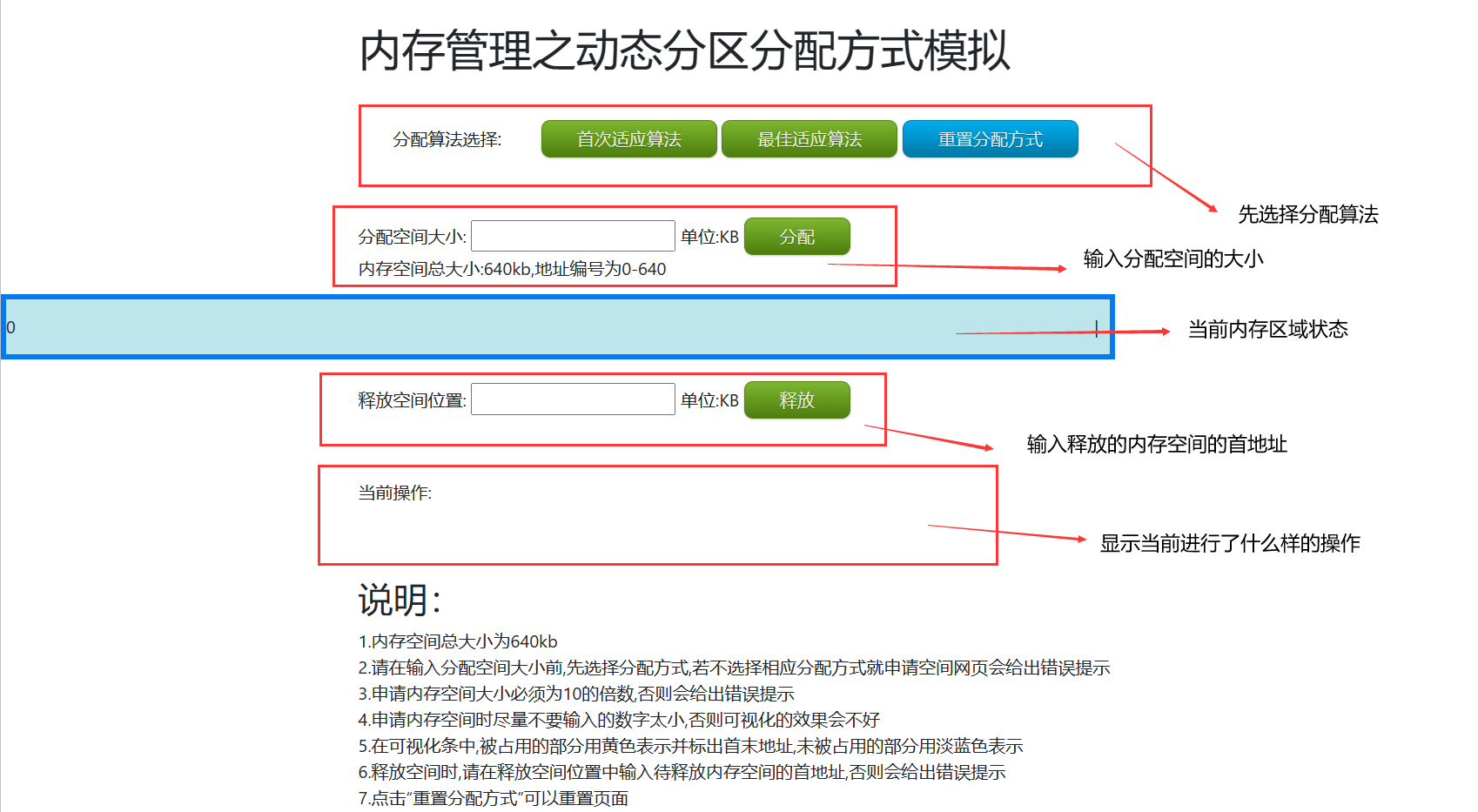
Tongji University

1 项目说明

本项目的开发技术为html+css+javascript+jQuery，使用VSCode+Edge作为开发工具。

启动项目直接双击打开index.html即可，打开界面后可以看到可视化界面和使用说明。效果展示视频如下：<效果展示.mp4>.

2 项目界面展示



一些说明：

1.内存空间总大小为640kb；

2.请在输入分配空间大小前,先选择分配方式,若不选择相应分配方式就申请空间网页会给出错误提示；

3.申请内存空间大小必须为10的倍数,否则会给出错误提示；

4.申请内存空间时尽量不要输入的数字太小,否则可视化的效果会不好；

5.在可视化条中,被占用的部分用黄色表示并标出首末地址,未被占用的部分用淡蓝色表示；

6.释放空间时,请在释放空间位置中输入待释放内存空间的首地址,否则会给出错误提示；

7.点击“重置分配方式”可以重置页面；

3 项目文件结构及类设计

项目主要由三个文件组成：

|  |  |
| --- | --- |
| 文件名 | 作用 |
| index.html | 项目前端界面 |
| style.css | 前端界面样式 |
| allocAlogorithm.js | 后端动态分配算法 |

为了更加方便的表示内存当前的状态，我采用了双向链表的数据结构。链表的一个节点就是一段已被占用或未被占用的内存。为了实现双向链表，我在allocAlogorithm.js文件中封装了链表类和链表节点类，下面具体介绍allocAlogorithm.js文件中的这两个类。

3.1节点类：Node

|  |  |
| --- | --- |
| 类成员名称 | 作用 |
| startPos | 某段内存的起始地址 |
| length | 某段内存的长度 |
| statu | 某段内存的占用情况，true表示未被占用，false表示被占用 |
| former | 该节点在链表中的前驱节点 |
| next | 该节点在链表中的后继节点 |

**3.2双向链表类：List**

|  |  |
| --- | --- |
| 类成员名称 | 作用 |
| size | 链表长度 |
| head | 链表空置头结点 |
| tail | 链表空置尾结点 |

此外，双向链表类中还封装了很多类方法，简要说明如下：

|  |  |
| --- | --- |
| 类方法名称 | 作用 |
| findNode | 按照先后位置寻找并返回节点 |
| findStartPosFree | 按照起始地址寻找被占用节点 |
| insertBack | 尾插法插入节点 |
| remove | 删除某个位置处的节点 |
| clearList | 在重置链表时清空当前表 |
| firstFit | 首次适应算法实现 |
| bestFit | 最佳适应算法实现 |
| FreeNode | 释放被占有的节点并和前后的节点合并 |
| renewInterface | 更新链表的页面显示 |

4算法设计

4.1 首次适应算法 List:firstFit函数

firstFit函数有1个参数blockSize，这个参数记录待分配的内存空间的大小。算法的思想就是从低地址向高地址遍历，一旦找到长度大于blockSize且未被占用的内存空间就在这个空间的起始地址处申请blockSize大小的内存空间。具体的做法是将未被占用空间的首地址向后偏移blockSize大小，同时申请一个新的节点用于记录新申请的blockSize块，新申请的块的起始地址是原来未被占用的块的起始地址，长度为blockSize。然后在链表结构中将申请的块插入到未被占用的块之前；若未被占用块的大小等于blockSize，则只需将未被占用块的statu属性由未被占用（true）改为已被占用（false）即可。若未找到足够大的空闲空间，则以弹窗的方式提醒用户空间不足，主要代码如下：

 let tempPtr=this.head.next;

        for(let i=1;i<=this.size;i++){

            if(tempPtr.statu===true && tempPtr.length>=blockSize){

                let startPosition=tempPtr.startPos;

                let newNode=new Node(startPosition,blockSize,false);

                if(tempPtr.length>blockSize){

                    tempPtr.startPos+=blockSize;

                    tempPtr.length-=blockSize;

                    tempPtr.former.next=newNode;

                    newNode.former=tempPtr.former;

                    newNode.next=tempPtr;

                    tempPtr.former=newNode;

                    this.size+=1;

                }

                else if(tempPtr.length===blockSize){

                    tempPtr.statu=false;

                }

                return;

            }

            tempPtr=tempPtr.next;

    }

**4.2最佳适应算法List:bestFit函数**

bestFit函数有1个参数blockSize，记录待分配的内存块的大小。该算法同样是从低地址向高地址遍历，当找到长度大于等于blockSize的空块的时候先用变量tempPtr记录下这个块，继续向后遍历，当找到长度更小的空块且这个空块的长度也大于等于blockSize的时候更新tempPtr，当遍历完整个链表后tempPtr存的一定是长度最小且长度大于等于blockSize的空块。然后在tempPtr块前添加长度为blockSize大小的空间再修改tempPtr块的起始地址即可。主要代码如下：

let tempFitNode=new Node();   //暂时记录最适应块

        tempFitNode=null;

        let tempPtr=this.head.next;

        for(let i=1;i<=this.size;i++){

            if(tempPtr.statu===true && tempPtr.length>=blockSize){  //找到可以合适的块

                if(tempFitNode===null){    //第一次找到

                    tempFitNode=tempPtr;

                }

                else if(tempPtr.length<=tempFitNode.length){    //找到更小的合适的块

                    tempFitNode=tempPtr;

                }

            }

            tempPtr=tempPtr.next;

        }

        //未找到合适的块

        if(tempFitNode===null){

            alert("空间不足,请释放一部分空间");

            return;

        }

        //新建块

        if(tempFitNode.length>blockSize){   //最小长度大于块大小

            let newNode=new Node(tempFitNode.startPos,blockSize,false);

            tempFitNode.startPos+=blockSize;

            tempFitNode.length-=blockSize;

            tempFitNode.former.next=newNode;

            newNode.former=tempFitNode.former;

            newNode.next=tempFitNode;

            tempFitNode.former=newNode;

            this.size+=1;

        }

        else if(tempFitNode.length===blockSize){   //最小长度等于块大小

            tempFitNode.statu=false;

        }

**4.3更新页面显示 List:renewInterface函数**

renewInterface函数使用了jQuery更新前端内存界面可视化显示。在前端页面中，内存空间是由64个横向排列的表格组成的，该函数用于在每次有新空间申请或有旧空间释放或重置界面后根据链表的情况更新前端界面显示。函数从前向后遍历链表，若当前该节点的statu为空闲（true），则将该节点所对应空间的单元格置为蓝色；若该节点的statu为被占用（false），则将该节点所对应空间的单元格置为黄色。最后标出每个内存空间的起始地址，并在每个内存空间的结束处加上符号“|”。

**4.4释放空间FreeBlock函数**

FreeBlock函数有一个参数startPosition，用于记录待释放空间的内存起始地址。在该函数中，首先调用findStartPosFree函数找到待释放内存空间的节点索引，若未找到则弹窗提示“不可释放”。然后调用FreeNode函数更新链表将节点释放，最后调用renewInterface函数更新前端页面显示。

5 总结

通过这次项目，我自己动手实现了最先适配算法和最佳适应算法，加深了对两种算法的理解，也对两者各自的优势有了更深的认识。

最先适配算法是指在空闲内存块链表中，按照地址从小到大的顺序查找第一个能够满足需要的空闲块，并将其分配给请求者。这种算法的优点是实现简单，但是可能会导致大量的内存碎片，影响内存利用率。

最佳适应算法是指在空闲内存块链表中，查找最小的能够满足请求的空闲块，并将其分配给请求者。这种算法相对于最先适配算法可以避免一定的内存碎片。

此外，这次项目还锻炼了我使用前端三件套的技术。