# 内存管理 - 动态分区分配方式模拟

# 目录



# 项目需求

假设初始态下，可用内存空间为640K，并有下列请求序列，请分别用首次适应算法和最佳适应算法进行内存块的分配和回收，并显示出每次分配和回收后的空闲分区链的情况来。

**作业1申请130K**
作业2申请 60K
作业3申请100k
作业2释放 60K
作业4申请200K
作业3释放100K
作业1释放130K

**作业1申请130K**
作业5申请140K
作业6申请 60K
作业7申请 50K
作业6释放 60K

# 项目目的

- 数据结构、分配算法
- 加深对动态分区存储管理方式及其实现过程的理解

# 开发环境

- **开发环境:** Windows 11
- **开发软件:**
    1. **Visual Studio Code** *1.81.1*
    2. **WebStorm** *2023.2*
- **开发语言:** html, javascript, css, jQuery
- **主要引用块内容:**

```
<script src="http://libs.baidu.com/jquery/2.0.0/jquery.min.js"></script>
```

# 项目结构

```
|  动态分区分配方式模拟_设计方案报告.md
|
├─Resource
|      memory.png
|
└─src
    |  Dynamic partition allocation.html
    |
    └─static
        ├─css
        |      range.css
        |      style.css
        |
        └─js
                clear.js
                nextAssingment.js
                randColor.js
```

```
18                  RangeSlider.js
19                  select.js
```

# 操作说明

- 双击目录src下的Dynamic partition allocation.html文件, 并在浏览器中打开
- 选择希望进行模拟的动态分区分配算法*(首次适应算法/最佳适应算法)*
- 调节滑动条改变当前内存大小*(上方模拟链式空间长度会随着滑动条滑动而动态变化)*
- 点击**下一步**进行作业调度
- 上方的模拟内存会显示每次分配和回收后的空闲分区链的情况*(不同作业的颜色不同, 以区分不同作业在内存中的位置分布情况)*
- 下方的日志信息会显示作业*申请/释放*等信息
- 点击**清空内存**会清空内存中的作业以及日志信息全部内容, 此时可再次调整内存空间大小, 并再次进行动态分区分配方式模拟

# 系统分析

## 首次适应算法

- 算法逻辑: 记录当前内存中被使用的空间, 同时记录当前内存中可以使用的空间(并将其按照物理位置顺序列出)

  如果当前作业需要申请内存空间 => 顺序查找第一个空闲块大小大于所需空间 => 将占用的内存空标记为被使用 => 空闲块大小和位置做相应的调整

## 最佳适应算法

- 算法逻辑: 同样记录当前内存中被使用的空间, 同时记录当前内存中可以使用的空间(并将其按照物理容量大小列出)

  如果当前作业需要申请内存空间 => 找出当前容量最小并且满足当前申请需求的物理块 => 将占用的内存空标记为被使用 => 空闲块大小和位置做相应的调整

# 系统设计

# 界面设计

1. **整体设计**
2. **内存模型**:

```
<!--模拟内存(可以动态调节大小 & 动态增加作业)-->
<div class="container">
    <center>
        <div class="memory" id="memory">

        </div>
    </center>
</div>
```

```
/*中部链式内存样式*/
.memory {
    margin-top: 5%;
    height: 100px;
    width: 200px;
    border-top: 4px solid black;
    border-bottom: 4px solid black;
    display: flex;
    position: relative;
}
```

3. **作业块:**

```
/*内存中的每一个作业*/
.memory-proj {
    margin-top: 1px;
    background-color: rgb(111, 120, 172);
    height: 98px;
    width: 100px;
    position: absolute;
}
```

4. **选择算法(下拉框)**:

```
<!--下拉框(选择分区算法)-->
<div class="algorithm"">
    <div>请选择分区分配算法</div>
        <div class=" model-box" style="width:180px; margin-top: 5px;">
            <div id="box" class="model-select-text" data-value="首次适应算法">首次适应算法</div>

            <ul class="model-select-option">
                <li data-option="首次适应算法">首次适应算法</li>
                <li data-option="最佳适应算法">最佳适应算法</li>
            </ul>
        </div>
</div>
```

```
/*下拉框*/
.model-box {
    position: relative;
    width: 200px;
    height: 30px;
    line-height: 30px;
```

```
    background-color: #fff;
    border: 1px solid #e4e4e4;
    border-radius: 3px;
    text-indent: 5px;
}

/*下拉框中待选文字*/
.model-box .model-select-text {
    position: relative;
    width: 100%;
    height: 28px;
    color: #666;
    text-indent: 10px;
    font-size: 14px;
    cursor: pointer;
    user-select: none;
}

.model-box .model-select-text:after {
    position: absolute;
    top: 10px;
    right: 10px;
    content: '';
    width: 0;
    height: 0;
    border-width: 10px 8px 0;
    border-style: solid;
    border-color: #666 transparent transparent;
}

.model-box .model-select-option {
    position: absolute;
    top: 30px;
    left: -1px;
    display: none;
    list-style: none;
    border: 1px solid #e4e4e4;
    border-top: none;
    padding: 0;
    margin: 0;
    width: 100%;
    z-index: 99;
    background-color: #fff;
}

.model-box .model-select-option li {
    height: 28px;
    line-height: 28px;
    color: #333;
    font-size: 14px;
    margin: 0;
    padding: 0;
    cursor: pointer;
}

.model-box .model-select-option li:hover {
    background-color: #f3f3f3;
}
```

```
65
66  .model-box .model-select-option li.seleced {
67      background-color: #f3f3f3;
68  }
```

5. **选择内存大小(滑动条)**:

```
1   <!--滑动条(动态调节内存大小)-->
2   <div class="memory-size" id="memory-size" style="margin-left: 100px;">
3       <div style="display: inline-flex;">请选择内存空间大小
           
4           <div id="current-size">200</div>
5           K</div>
6
7       <div class="range-slider" style="margin-top: 10px;">
8           <input id="slider" type="range" value="200" min="200" max="1000"
    step="20">
9       </div>
10  </div>
```

```
1   /*去除系统默认的样式*/
2   input[type=range] {
3       -webkit-appearance: none;
4       width: 300px;
5       border-radius: 10px;
6       /*这个属性设置使填充进度条时的图形为圆角*/
7   }
8
9   input[type=range]::-webkit-slider-thumb {
10      -webkit-appearance: none;
11  }
12
13  /*自定义滑动控件的轨道*/
14  input[type=range]::-webkit-slider-runnable-track {
15      height: 15px;
16      border-radius: 10px;
17      /*将轨道设为圆角的*/
18      box-shadow: 0 1px 1px #def3f8, inset 0 .125em .125em #0d1112;
19      /*轨道内置阴影效果*/
20  }
21  /*原始的控件获取到焦点时，会显示包裹整个控件的边框，所以还需要把边框取消。*/
22  input[type=range]:focus {
23      outline: none;
24  }
25
26  input[type=range]::-webkit-slider-thumb {
27      -webkit-appearance: none;
28      height: 25px;
29      width: 25px;
30      margin-top: -5px; /*使滑块超出轨道部分的偏移量相等*/
31      background: #ffffff;
32      border-radius: 50%; /*外观设置为圆形*/
33      border: solid 0.125em rgba(205, 224, 230, 0.5); /*设置边框*/
34      box-shadow: 0 .125em .125em #3b4547; /*添加底部阴影*/
35  }
```

6. **下一步/清空内存(按钮)**:

```
<!--执行下一指令-->
<div class="next-btn" style="margin-left: 100px">
    <button style="border-color: rgb(50, 196, 233);background-color: rgb(50, 196, 233);
            height: 45px; width: 70px;" onclick="nextAssignment()">
        下一步
    </button>
</div>

<!--清空内存-->
<div class="clear-btn">
    <button style="border-color: rgb(252, 0, 0);background-color: rgb(252, 0, 0);
                height: 45px; width: 70px;" onclick="clearbtnClick()">
        清空内存
    </button>
</div>
```

7. **日志信息(滚动div)**:

```
<!--日志信息滚动窗口-->
<div class="container" style="margin-top: 3%;">
    <center>
        <div>日志信息</div>
        <div id="board" style="height: 150px;width: 500px;background-color: #C1BDA3;
        overflow:auto; ">
        </div>
    </center>
</div>
```

# 类设计

1. **作业类**: 作业实体

```
/**
 * 作业类
 * @param {作业名称} name
 * @param {申请/释放的空间大小} data
 */
function Proj(name, data) {
    this.name = name;
    this.data = data;
    this.getname = function () {
        return this.name
    }
    this.getdata = function () {
        return this.data
    }
}
```

2. **标记类**: 标记作业在内存中的起始地址和长度

```
/**
 * 标记类
 * @param {起始地址} start
 * @param {长度} last
```

```
 */
function Mark(start, last) {
    this.start = start;
    this.last = last;
    this.getstart = function () {
        return this.start;
    }
    this.getlast = function () {
        return this.last;
    }
}
```

## 实体设计

1. **作业列表:**

```
/**
 * 作业列表(后期可考虑动态扩充)
 */
const projList = [
    new Proj("作业1", 130),
    new Proj("作业2", 60),
    new Proj("作业3", 100),
    new Proj("作业2", -60),
    new Proj("作业4", 200),
    new Proj("作业3", -100),
    new Proj("作业1", -130),
    new Proj("作业5", 140),
    new Proj("作业6", 60),
    new Proj("作业7", 50),
    new Proj("作业6", -60)
]
```

2. **被占用标记表:** `occupyMem = []`
3. **空闲空间标记表:** `useableMem = []`

## 状态设计

1. 添加作业成功: `const ADDSUCCESS = 0;`
2. 添加作业失败: `const ADDFAILED = 1;`
3. 释放作业成功: `const REMOVESUCCESS = 2;`

## 系统实现

# 下拉框选择算法

- 单击某个下拉列表，显示当前下拉列表的下拉列表框，并隐藏页面中其他下拉列表
- 点击选择，关闭其他下拉框
- 点击文档隐藏所有下拉框

```
$(function () {
    selectModel();
});

/*下拉列表选择*/
function selectModel() {
    var $box = $('div.model-box');
    var $option = $('ul.model-select-option', $box);
    var $txt = $('div.model-select-text', $box);
    var speed = 10;
    /**
     * 单击某个下拉列表时，显示当前下拉列表的下拉列表框
     * 并隐藏页面中其他下拉列表
     */
    $txt.on('click', function () {
        var $self = $(this);
        $option.not($self).siblings('ul.model-select-option').slideUp(speed, function () {
            init($self);
        });
        $self.siblings('ul.model-select-option').slideToggle(speed, function () {
            init($self);
        });
        return false;
    });

    // 点击选择，关闭其他下拉框
    /**
     * 为每个下拉列表框中的选项设置默认选中标识 data-selected
     * 点击下拉列表框中的选项时，将选项的 data-option 属性的属性值赋给下拉列表的 data-value 属性，并改变默认选中标识 data-selected
     * 为选项添加 mouseover 事件
     */
    $option.find('li').each(function (index, element) {
        var $self = $(this);
        if ($self.hasClass('selected')) {
            $self.addClass('data-selected');
        }
    }).mousedown(function () {
        $(this).parent().siblings('div.model-select-text').text($(this).text()).attr('data-value', $(this).attr('data-option'));

        $option.slideUp(speed, function () {
            init($(this));
        });
        $(this).addClass('selected data-selected').siblings('li').removeClass('selected data-selected');

```

```
        //输出选择的算法
        console.log($("#box").attr("data-value"))

        return false;
    }).mouseover(function () {

 $(this).addClass('selected').siblings('li').removeClass('selected');
    });

    // 点击文档隐藏所有下拉框
    $(document).on('click', function () {
        var $self = $(this);
        $option.slideUp(speed, function () {
            init($self);
        })
    });

    /**
     * 初始化默认选择
     */
    function init(obj) {
        obj.find('li.data-
selected').addClass('selected').siblings('li').removeClass('selected');
    }
}
```

## 滑动条动态调节内存大小

- 内存设置为200K~1000K可动态改变
- 初始化滚动条处于最左端(内存容量为最小值), 最小移动步长为20K
- 使用html5提供的<input type="range">创建滚动条并绑定change()事件

```
//滑动条改变触发事件
var change = function ($input) {
    console.log($input.value)
    //将slider的值进行函数映射
    document.getElementById('current-size').innerHTML = $input.value;
    document.getElementById('memory').style.width = String($input.value) +
"px";
}

$('input').RangeSlider({
    min: 200,
    max: 1000,
    step: 20,
    callback: change
});
```

- 根据滑动条当前value进行函数映射, 对应到内存的当前大小, 并填充色块代表选择的范围

```
$.fn.RangeSlider = function (cfg) {
    this.sliderCfg = {
        min: cfg && !isNaN(parseFloat(cfg.min)) ? Number(cfg.min) : null,
        max: cfg && !isNaN(parseFloat(cfg.max)) ? Number(cfg.max) : null,
        step: cfg && Number(cfg.step) ? cfg.step : 1,
```

```
        callback: cfg && cfg.callback ? cfg.callback : null
    };

    var $input = $(this);
    var min = this.sliderCfg.min;
    var max = this.sliderCfg.max;
    var step = this.sliderCfg.step;
    var callback = this.sliderCfg.callback;

    $input.attr('min', min)
        .attr('max', max)
        .attr('step', step);

    $input.bind("input", function (e) {
        $input.attr('value', this.value);
        $input.css('background', 'linear-gradient(to right, #059CFA, white
' + (this.value - 200) / 8 + '%, white)');
                            //其中要将this.value进行从[200,1000]到[0,100]的
函数映射

        if ($.isFunction(callback)) {
            callback(this);
        }
    });
};
```

- 将用户选择的当前内存容量显示到滑动条的上方便于用户账务内存容量信息
- 用户点击下一步后滑动条失效(不可以再更改内存空间大小)
- 用户点击清空内存后滑动条恢复

# 下一步进行作业调度

- 设定**全局定位器**now = 0用于标识当前执行任务列表的第几条
- 设定**被占用标记表**occupyMem和**空闲空间标记表**useableMem
- 依据now值进行分支处理:
  - 如果now值为0 => 内存中没有任何作业 => 全部内存空间标记为可使用 => 起始位置为0, 长度为当前内存空间大小

    ```
    if (occupyMem == false) { //内存没有任何作业
        useableMem.push(new Mark(0, memSize));
    }
    ```

  - 如果now值小于任务列表的长度 => 继续执行下一条指令
  - 如果now值大于等于任务列表的长度 => 所有任务已经执行完毕 => 输出相应日志信息

    ```
    var mess = document.createElement("mess");
    mess.type = "div";
    mess.innerText = "作业已全部完成!\n";
    mess.style.color = "red";
    Board.appendChild(mess);
    ```

- 执行下一条指令时:
  - 如果可以装入作业 => now指向下一条待执行的指令

# 分区分配算法

- 如果该指令是要申请空间:
  - 首次适应算法:
    - 依照Mark的起始位置将空闲空间表进行排序
    - 寻找第一个能放下该作业的位置
  - 最佳适应算法:
    - 依照Mark的大小将空闲空间表进行排序
    - 寻找能放下该作业的最小内存位置

```
if(algorithm == "首次适应算法"){
    useableMem.sort(compareStart)
}else if(algorithm == "最佳适应算法"){
    useableMem.sort(compareLast);
}
```

  - 将这段空间记录在被占用标记表中 => 将被占用标记表重新排序
  - 将该段的空闲空间表起始位置加上作业申请的内存空间大小
  - 将该段的空闲空间表大小减去作业申请的内存空间大小
  - 在模拟的内存中添加一个作业块
  - 在日志信息中增加一行申请成功的日志

```
for (var i = 0; i < useableMem.length; ++i) {
    if (useableMem[i].getlast() > data) { //第一个能放下的位置
        start = useableMem[i].getstart();
        last = useableMem[i].getlast();

        occupyMem.push(new Mark(start, data));
        occupyMem.sort(compareStart)

        useableMem[i].start += data;
        useableMem[i].last -= data;

        addProj(Mem, name, start, data);
        addMess(Board, name, data, start, ADDSUCCESS);

        return true;
    }
}
```

- 如果该指令是要释放空间:
  - 在模拟内存中清除该作业块
  - 获取该作业的起始位置
  - 清除占用表中该作业的项
  - 在空闲空间表末尾添加一块新的可以使用的空间
  - 对可用内存重新整理
  - 添加一条释放成功的日志信息

```
var proj = document.getElementById("proj" + name[2]);
document.getElementById("memory").removeChild(proj); //清除作业块

/*获取起始位置 */
var start = proj.style.marginLeft;
start = Number(start.slice(0, -2));
console.log(start)

for (var i = 0; i < occupyMem.length; ++i) {
    if (occupyMem[i].getstart() == start) { //清除占用表项
        last = occupyMem[i].getlast();
        occupyMem.splice(i, 1);
        break;
    }
}

useableMem.push(new Mark(start, last)); //先在末尾添加一块新的可以使用的空间
update(); //对可用内存重新整理

addMess(Board, name, data, start, REMOVESUCCESS);

return true;
```

- 如果申请失败:
  - 在日志信息中添加一条申请失败的日志信息

```
addMess(Board, name, data, -1, ADDFAILED);
return false;
```

## 更新空闲空间标记表

- 每次释放一个作业之后要对空闲空间表进行更新
- 先将其依照Mark的起始位置进行排序
- 当前空闲块和后面的空闲块可以合并, 持续循环合并 => 合并后删除后面的空闲块
- 直到标记表被整个遍历完

```
/**重新整理useableMem */
function update() {
    useableMem.sort(compareStart)
    console.log("before update", useableMem)

    var i = 0;
    while (i < useableMem.length) {
        while (i + 1 < useableMem.length &&             //当前空闲块和后面的空闲块可以合并,持续循环合并
            (useableMem[i].getstart() + useableMem[i].getlast() == useableMem[i + 1].getstart())) {
            useableMem[i].last += useableMem[i + 1].getlast();
            useableMem.splice(i + 1, 1);    //合并后删除后面的空闲块
        }
        ++i;
    }
    console.log("after update", useableMem)
}
```

## 添加一个作业块

```
/**
 * 添加一个作业块(不检测)
 * @param {内存实体} Mem
 * @param {作业名称} name
 * @param {起始位置} start
 * @param {作业数据信息} data
 */
function addProj(Mem, name, start, data) {
    var proj = document.createElement("proj");
    proj.type = "div";
    proj.classname = "memory-proj";
    proj.id = "proj" + name[2];
    proj.innerText = "\n" + name + "\n" + data + "K" + "\n"; //作业块内部显示作业名和作业大小
    proj.style.marginTop = "1px";
    proj.style.background = randomHexColor(); //随机配色
    proj.style.height = "98px";
    proj.style.width = String(data) + "px"; //作业块的宽度为作业大小

    /*实现以内存左端点为基准定位 */
    proj.style.position = "absolute";
    proj.style.marginLeft = String(start) + "px";

    Mem.appendChild(proj);
}

```

## 添加一行日志信息

- 作业申请内存成功: 输出作业编号, 释放内存大小, 内存中的起始位置
- 作业申请内存失败: 输出作业编号, 释放内存大小, 当前内存空间不足
- 作业释放内存成功: 输出作业编号, 释放内存大小

```
/**
 * 添加一行日志信息
 * @param {告示板实体} Board
 * @param {作业名称} name
 * @param {作业数据信息} data
 * @param {存放的起始位置} start
 */
function addMess(Board, name, data, start, flag) {
    var mess = document.createElement("mess");
    mess.type = "div";
    if (flag == ADDSUCCESS) {
        mess.innerText = name + "申请" + String(data) + "K内存空间成功\n" + "起始位置是" + String(start) + "\n\n";
        mess.style.color = "black";
    } else if (flag == ADDFAILED) {
        mess.innerText = name + "要申请" + String(data) + "K内存空间\n当前内存空间不足!\n\n";

```

```
        mess.style.color = "red";
    } else if (flag == REMOVESUCCESS) {
        mess.innerText = name + "释放" + String(-data) + "K内存空间成功\n\n";
        mess.style.color = "blue";
    }
    Board.appendChild(mess);
}
```

## 清空内存

- 用户点击清空内存按钮 => 清除模拟内存中的所有作业块 => 清空日志信息中的所有日志 => 重置作业列表(从头开始作业调度) => 清空占用列表 => 清空可用列表 => 重置滑动条为可用

```
function clearbtnClick(){
    console.log('clear...')
    var Mem = document.getElementById('memory');
    while(Mem.hasChildNodes()){
        Mem.removeChild(Mem.firstChild)
    }

    var Board = document.getElementById('board');
    while(Board.hasChildNodes()){
        Board.removeChild(Board.firstChild)
    }

    now = 0;          //重置作业列表(从头开始作业调度)
    occupyMem.length = 0;    //清空占用列表
    useableMem.length = 0;   //清空可用列表
    $("#memory-size").removeClass("disable")     //重置滑动条为可用
}
```

## 添加的作业随机颜色标识

- 添加到模拟内存中的作业会被随机配色以示区分

```
function randomHexColor() { //随机生成十六进制颜色
    var hex = Math.floor(Math.random() * 16777216).toString(16); //生成ffffff以内16进制数
    while (hex.length < 6) { //while循环判断hex位数，少于6位前面加0凑够6位
        hex = '0' + hex;
    }
    return '#' + hex; //返回'#'开头16进制颜色
}
```