OS-电梯调度

姓名 刘客 **学号** 1651573

一、系统概述

1.1需求分析

运用操作系统进程管理的知识，结合操作系统调度算法，模拟电梯调度的功能。本电梯模拟系统有5个电梯，共20层楼。每个电梯在每一层有一个操作面板，往上和往下按钮以及显示当前楼层。每次按下按钮，五部电梯同时响应，将执行调度算法，选择最优电梯进行调度，调度过程可变，总是选择最优电梯进行调度，尽可能模拟真实电梯情况。

1.2开发语言

使用**javascript**语言，使用**原生html+css**做界面展示，使用**jquery**来进行点击交互设计 建议页面缩放比例为**100%**，这样页面能够得到很好地展示，否则会出现图标错位等情况

1.3系统简述

1.3.1功能简述

共5个电梯，20层楼，每个电梯在每层楼有两个按钮，表示往上和往下请求，显示当前楼层的面板。电梯内部有一个控制面板，用户进入后可以选择楼层。另设开门关门键，提供手动电梯开关门操作。在到达指定楼层时，电梯会自动执行开关门操作。

1.3.2 电梯实现

由于javascript是**单线程语言**，所以使用**setInterval**来设立定时函数,模拟多线程实现 本电梯系统除了主线程之外，还模拟了6个线程 主线程主要用来处理点击操作，和页面动态变化。剩下6个线程中，5个用来模拟电梯操作，最后一个用来调度分配任务(用来处理外部调度)。

1.3.3 调度算法简述

- 内部调度
  - 当用户选定楼层与电梯运行方向处于一致时 将该按钮高亮并将该楼层加入该电梯的经停列表里，当到达该楼层时，电梯会执行停留操作。
  - 当用户选定楼层与电梯运行方向不一致时 为了便捷操作，默认该按钮不可选定。即使点击也不会触发高亮操作。
- 外部调度

  当用户点击外部楼层按键时，该按键高亮，同时将该指令加入指令序列。当指令序列不为空时，调度线程即会将该指令寻找合适的电梯进行分配。该处理过程按入指令序列的顺序进行处理，但有一定的分配规则用来规避“先到先服务”带来的资源损失。

  分配优先级(分配规则)

> **>if**

> **>1** 恰好有停留在该层的电梯

> **>2** 有与该请求同向的电梯，且双方楼层间隔小于5

> **>else** 该操作遗留到下次进行处理

具体实现后面会结合代码进行讲述

二、代码设计

2.1 类和变量

Elevator类

运行调度的主体

> _Cstatus与_ Tstatus解释 e.p. 用户想从18楼下去，但所有的电梯均位于18楼以下，此时电梯需要先上再下，_Cstatus就是上，而_Tstatus是下

> _SLayer元素的添加 每次向_SLayer添加楼层时，总会对其进行一次排序操作。确保从前往后是电梯运行方向上依次到达的楼层。

> _TFloor转换 当用户_Cstatus与_Tstatus不同时，此时_TFloor为电梯要到达的楼层，按上面的例子即是18楼。当电梯到达18楼时，_TFloor将被转换成 _SLayer的尾元素

变量	含义
_Cstatus	当前电梯的运行状态
_Tstatus	调度操作实际要求的电梯的运行状态
_CFloorl	当前的floor
_SLayer	(set类型) 电梯将要进行停滞的楼层
_TFloor	调度操作实际要求的floor
_CanOpen	是否可以开门
_Interupt	是否阻断了自动开关门操作

Operation类

指令类

变量	含义
_Floor	请求电梯到来的楼层
_status	上行还是下行

全局元素

- 全局常量

变量	含义
INTERVAL	执行调度优先级分配(2)的间隔选择,当前选定为5
STATUS_FREE	电梯状态----空闲状态
STATUS_UP	电梯状态----上行状态
STATUS_DOWN	电梯状态----下行状态
RUNNING_ON	该电梯被调度
RUNNING_OFF	该电梯未被调度

- 全局变量

变量	含义
_minFloor	最低楼层
_maxFloor	最高楼层
_elevatorNum	电梯数量
_timer	定时器数组，用来存储模拟电梯线程的定时器
_running	用来判断当前电梯是否已经被调度
_elevatorArray	电梯数组，用来存储每个电梯实例
_operationArray	用来存储进入的操作实例

2.2 函数列表

> jquery 点击事件此处不予赘述，可去elevator-dispath.js详细查看。

函数名称	作用
init()	初始化电梯数组，并设立定时器
sortDesc(x,y)	降序排列
sortAesc(x,y)	升序排列
openDoor(n,sign=false)	模拟开门动画，并且如果该操作打断了正常的定时器逻辑，则重新设立逻辑
closeDoor(n,sign=false)	模拟关门动画，并且如果该操作打断了正常的定时器逻辑，则重新设立逻辑
selectElevator(floorIndex,isUp)	选择电梯
computeDistance(floor,CFloor)	计算发出请求楼层和电梯当前楼层的距离的绝对值
addLayerToUp(n,floorIndex,sign=false)	将楼层添加到某个上行电梯
addLayerToDown(n,floorIndex,sign=false)	将楼层添加到某个下行电梯
checkExist(floorIndex,status)	检测某个指令是否已经存在
initTimer(n)	初始化定时器
clearTimer(n)	清除某个定时器
pushSequence(floorIndex,status)	将命令加入指令序列
processSequence()	模拟的处理指令序列线程函数，处理指令序列
arriveOperation(n,CFloor,sign=false)	电梯到达某个停滞楼层后执行的操作
arriveAnimate(n,CFloor,t_status,sign)	电梯到达某个楼层后自动执行动画，并更改一些数据
run(n)	模拟的电梯运行线程函数，模拟整体电梯运行
moveUP(n)	上行动画和数据更改
moveDOWN(n)	下行动画和数据更改
removeLight(n)	清除掉电梯数字显示板箭头灯光
removeOutsideLight(n,floorIndex,status)	清除外部控制板上某楼层的上行或者下行按钮的高亮
deleteOperationArray(index)	删除某一指令
turnLight(n,status)	更改上行/下行灯光方向

2.2 电梯运行逻辑

2.2.1 内部调度指令

当点击电梯内部控制板按钮时，会判断电梯状态。如果该按钮和电梯运行冲突，则无效。若不冲突，则添加到电梯停滞楼层序列，并高亮。如果电梯此时是空闲状态，则会对电梯参数进行一些设置。

```
$(".dial .button").click(function () {
    let elevatorIndex = $(this).parent()[0].id;
    elevatorIndex = parseInt(elevatorIndex.substr(7));
    let floorIndex = $(this)[0].textContent;
    floorIndex = parseInt(floorIndex);
    let judge = false;
    if(_elevatorArray[elevatorIndex]._Tstatus == STATUS_FREE){
        judge = true;
    }
    if(_elevatorArray[elevatorIndex]._Cstatus !=
_elevatorArray[elevatorIndex]._Tstatus){
        return;
    }
    if(floorIndex > _elevatorArray[elevatorIndex]._CFloor &&
_elevatorArray[elevatorIndex]._Tstatus != STATUS_DOWN){
        $(this).addClass("pressed");
        addLayerToUp(elevatorIndex,floorIndex,judge);
        if(_elevatorArray[elevatorIndex]._Tstatus == STATUS_FREE){
            _elevatorArray[elevatorIndex]._Tstatus =
_elevatorArray[elevatorIndex]._Cstatus = STATUS_UP;
        }
    }else if(floorIndex < _elevatorArray[elevatorIndex]._CFloor &&
_elevatorArray[elevatorIndex]._Tstatus != STATUS_UP){
        $(this).addClass("pressed");
        addLayerToDown(elevatorIndex,floorIndex,judge);
        if(_elevatorArray[elevatorIndex]._Tstatus == STATUS_FREE){
            _elevatorArray[elevatorIndex]._Tstatus =
_elevatorArray[elevatorIndex]._Cstatus = STATUS_DOWN;
        }
    }else{
        return;
    }
    if(_running[elevatorIndex] == RUNNING_OFF){
        _running[elevatorIndex] = RUNNING_ON;
    }
});
```

2.2.2 外部调度指令

- 外部请求

  点击会触发事件，此时会高亮按钮并调用 selectElevator函数

```
$(".up").off().on('click',function (e) {
  e.stopPropagation();
  let floorIndex = $(this).parent()[0].id.substr(5);
  if($(this).hasClass('on')){
      // if it has already dials the button;
      return;
  }
  floorIndex = parseInt(floorIndex);
  $(this).addClass('on');
  selectElevator(floorIndex,true);
```

```
    return false;
});
//click event on outside panel

$(".down").off().on("click",function (e) {
    e.stopPropagation();
    let floorIndex = $(this).parent()[0].id.substr(5);
    if($(this).hasClass('on')){
        return;
    }
    floorIndex = parseInt(floorIndex);
    $(this).addClass('on');
    selectElevator(floorIndex,false);
    return false;
});
```

selectElevator在判断指令状态后会调用pushSequence函数

```
function selectElevator(floorIndex,isUp) {
  let status = isUp?STATUS_UP:STATUS_DOWN;
  pushSequence(floorIndex,status);
}
```

pushSequence函数在检查该指令是否已经在指令序列中存储后(jquery迷之操作,click会多次触发,off和on混合使用无法解决)，若不存在则将其加入到指令序列中。

```
function pushSequence(floorIndex,status) {
  let result = checkExist(floorIndex,status);
  let process = false;
  if(result != -1){
      process = true;
  }

  if(!process){
      _operationArray.push(new Operation(floorIndex,status));
      console.log("push"+floorIndex+"   status"+status);
  }
}
```

- 指令调度

  该函数运行在一个独立的模拟线程中 每1s执行一次 执行时会进行嵌套循环 外循环为电梯 内循环为两个同级循环

  - 第一个内循环 第一个内循环在当前电梯处于**空闲状态**时执行，从指令序列中找出与当前电梯距离最近的指令，将该指令分配给当前电梯，并**继续进行第二个内循环**
  - 第二个内循环 第二个内循环用于寻找与当前电梯同向，并且楼层距离小于Interval(设为5)的楼层，若寻找到，则将该楼层添加到当前电梯的_SLayer里。

```
function processSequence() {
    let tempArray = _operationArray;
    let length = tempArray.length;
```

```
    if(length <= 0 ){
        return;
    }
    console.log(tempArray.toString());
    // if elevator is stopping
    for(let i = 0; i < _elevatorNum;i++){
        if(length <= 0){
            return;
        }
        if(_elevatorArray[i]._Cstatus != _elevatorArray[i]._Tstatus ||
_elevatorArray[i]._CanOpen){
            continue;
        }
        if(_elevatorArray[i]._Tstatus == STATUS_FREE){
            let minD = 100;
            let chooseIndex = -1;
            console.log(length);
            for(let j = 0;j <length;j++){
                let len =
computeDistance(tempArray[j]._Floor,_elevatorArray[i]._CFloor);
                console.log(tempArray[j]._status);
                if(len < minD){
                    minD = len;
                    chooseIndex = j;
                }
            }

 moveElevator(i,tempArray[chooseIndex]._Floor,tempArray[chooseIndex]._status ==
STATUS_UP);
            console.log("index"+i +"pick" + " "+tempArray[chooseIndex]._Floor + "
"+ tempArray[chooseIndex]._status);

 deleteOperationArray(checkExist(tempArray[chooseIndex]._Floor,tempArray[chooseInde
x]._status));
        //    tempArray.splice(chooseIndex,1);
            length--;
            if(length <= 0){
                return;
            }
        }else{
            for(let j = 0;j < length;){
                if((tempArray[j]._status == _elevatorArray[i]._Tstatus)){
                    let floor = tempArray[j]._Floor;
                    if(tempArray[j]._status == STATUS_UP && floor >
_elevatorArray[i]._CFloor
                        && computeDistance(floor,_elevatorArray[i]._CFloor) <
INTERVAL){
                        addLayerToUp(i,floor);
                        console.log("index"+i +"pick" + " "+floor+ " "+
tempArray[j]._status);

 deleteOperationArray(checkExist(floor,tempArray[j]._status));
            //            tempArray.splice(j,1);
```

```
                        length--;
                        continue;
                    }
                    if(tempArray[j]._status == STATUS_DOWN && floor <
_elevatorArray[i]._CFloor
                        && computeDistance(floor,_elevatorArray[i]._CFloor)<
INTERVAL){
                        addLayerToDown(i,floor);
                        console.log("index"+i +"pick" + " "+floor+ " "+
tempArray[j]._status);

 deleteOperationArray(checkExist(floor,tempArray[j]._status));
            //            tempArray.splice(j,1);
                        length--;
                        continue;
                    }
                }
                j++;
            }
        }
    }
}
```

2.2.3 电梯运行

在每个电梯线程里，都会周期执行run函数

如果电梯的_SLayer里含有当前楼层，则会进行滞留操作(会判断是否需要转向)

如果没有，则继续进行上行或下行操作。

```
function run(n) {
    if(_running[n] == RUNNING_ON){
        let c_status = _elevatorArray[n]._Cstatus;
        let t_status = _elevatorArray[n]._Tstatus;
        let TFloor = _elevatorArray[n]._TFloor;
        let CFloor = _elevatorArray[n]._CFloor;
        if((CFloor == TFloor) || ((c_status == t_status)
&&_elevatorArray[n]._SLayer.has(CFloor))){
            if(_timer[n]){
                console.log("clear Timer");
                clearTimer(n);
            }
            if(CFloor == TFloor){
                if(c_status == t_status){
                    console.log(  "index =" + n + "arrive "+ TFloor + "and t==c  ");
                    arriveOperation(n,CFloor)
                }else{
                    console.log(  "index ="+ n + "arrive "+ TFloor + "and t!=c  ");
                    arriveOperation(n,CFloor,true)
                }
            }else{
                console.log("index =" + n +"arrive"+CFloor);
```

```
                arriveOperation(n,CFloor);
            }
        }
        else{
            if(c_status == STATUS_UP){
                moveUP(n);
            }else if(c_status == STATUS_DOWN) {
                moveDOWN(n);
            }
        }
    }
}
```

arriveOperation里主要将当前的楼层删去。sign代表是否会进行转向。 由于会存在用户主动开关门的操作，所以一些数据逻辑也只能放到arriveAnimate函数中进行更改

```
function arriveOperation(n,CFloor,sign=false) {
    let t_status = _elevatorArray[n]._Tstatus;
    _elevatorArray[n]._SLayer.delete(CFloor);
    console.log("index =" + n + "delete" + CFloor);
    arriveAnimate(n,CFloor,t_status,sign);
}
```

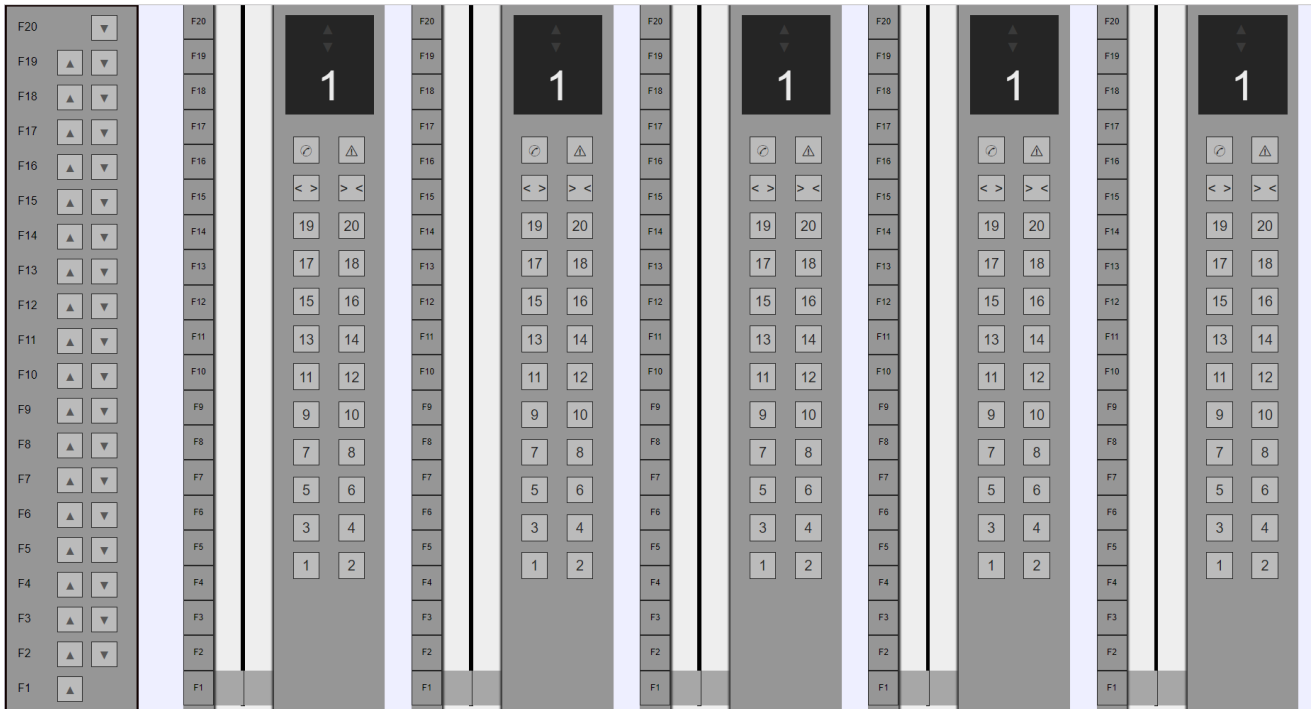
arriveAnimate主要使用延时操作setTimeout来完成各种动画的实现以及数据的更改。 sign代表是否会进行转向。如果sign为true，则会进行转向数据设置，更改_Tstatus 和 _Cstatus以及_Tstatus 这之后还会判断是否_SLayer为空，如果为空则会转换状态。 除此之外，也会判断是否受到了人为操作的干扰，比如有人按了开门键，如果有人进行这样操作，则会停止某些操作，而更改到那些按键事件中进行执行。

```
function arriveAnimate(n,CFloor,t_status,sign) {
    setTimeout(function () {
        _elevatorArray[n]._CanOpen = true;
        openDoor(n);
        if(sign){
            t_status = _elevatorArray[n]._Tstatus;
            _elevatorArray[n]._Cstatus = t_status;
            // turn direction
            turnLight(n,_elevatorArray[n]._Tstatus);
            let stopArray = Array.from(_elevatorArray[n]._SLayer);
            if(t_status == STATUS_UP){
                stopArray.sort(sortAesc);
            }else if(t_status == STATUS_DOWN){
                stopArray.sort(sortDesc);
            }
            if(stopArray.length != 0){
                _elevatorArray[n]._TFloor = stopArray[stopArray.length-1];
                console.log("turn index"+n+" TFloor to "+ stopArray[stopArray.length-
1])
            }
        }
        setTimeout(function () {
            closeDoor(n);
            removeOutsideLight(n,CFloor,t_status);
```

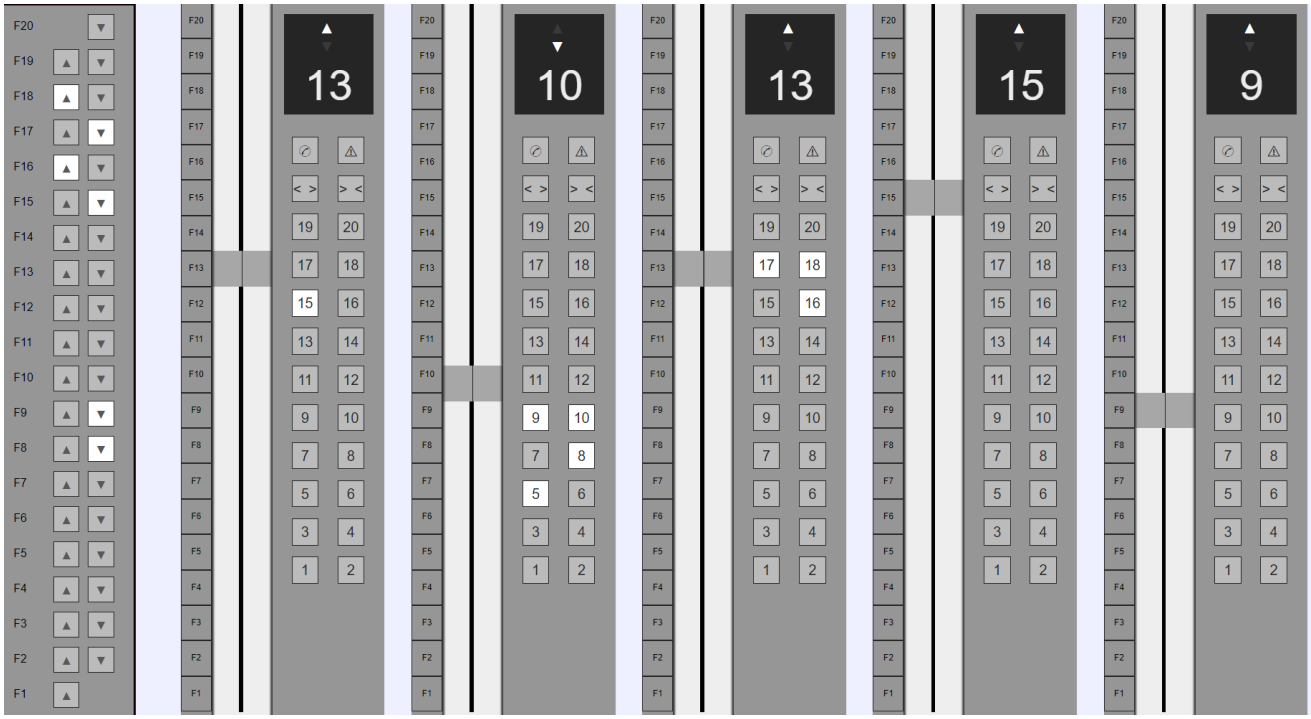
```
            setTimeout(function () {
                // if the sequence is empty, then the elevator will still in CFloor
                if(_elevatorArray[n]._SLayer.size == 0){
                    _elevatorArray[n]._Cstatus = _elevatorArray[n]._Tstatus =
STATUS_FREE;
                    _running[n] = RUNNING_OFF;
                    console.log("no stop floor so end..");
                    removeLight(n);
                }
                if(!_elevatorArray[n]._Interupt){
                    _elevatorArray[n]._CanOpen = false;
                    if(_timer[n] == -1){
                        _timer[n] = initTimer(n);
                        console.log("init Timer!!!");
                    }
                }
            },2000);
        },2000);
    },2000);
}
```

三、运行效果

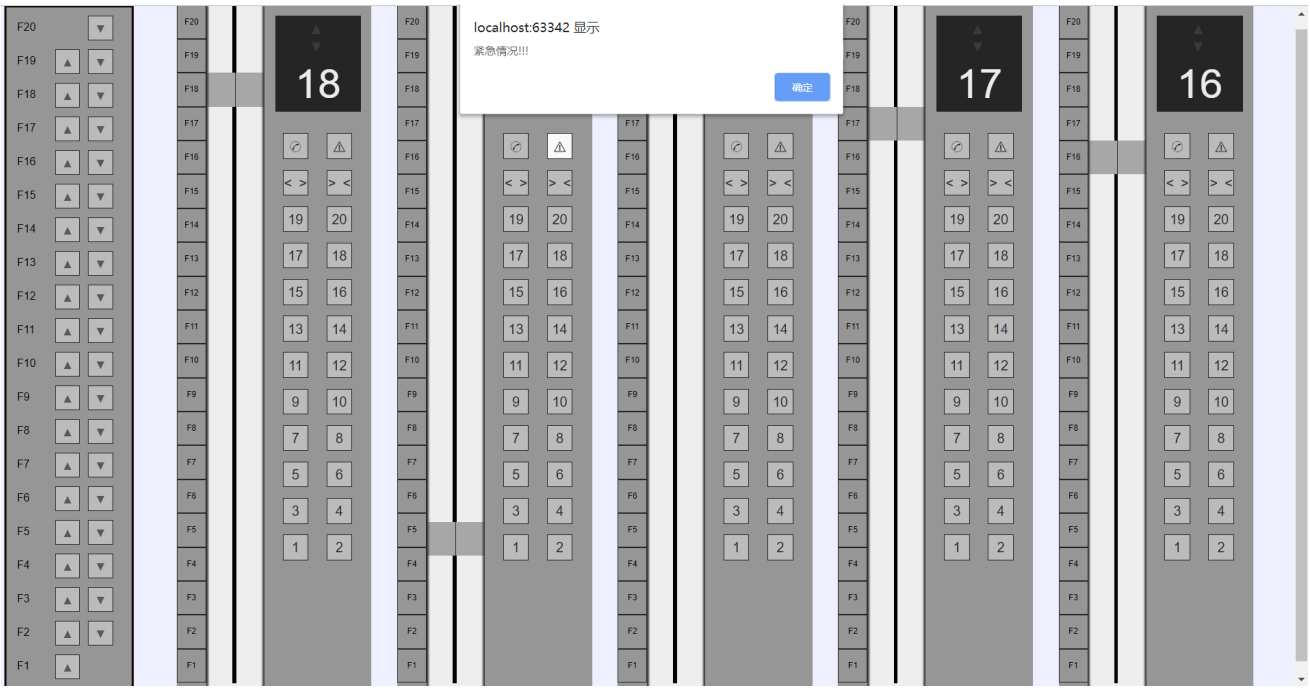
初始界面



运行截图



警报



四、分析

通过此次项目，我对进程调度有了更深的理解。在使用javascript模拟线程时，我遇到了很多的问题。一开始调度的处理并非分配了一个额外的线程，导致各个电梯会经常产生资源冲突产生很多bug。其次，手动开关门也给了我很大的困扰。因为代码如果设置的不合理就会多产生很多定时器，导致一个电梯同时被几个线程所操作，产生问题。后来经过细心思考，解决了这些问题。

但目前的代码并不是完美的。虽然前期进行了思考，但由于理解不到位，当时实现的效果并不是很好。后来虽然进行了修改，可以流畅地运行，但由于修改是在前面的代码上进行修改的，导致代码很乱。很多函数并不需要，而很多函数由于各种逻辑上的原因吸收了别的函数的代码显的比较臃肿。因此代码整体比较冗余。