数据结构与算法设计综合训练

组名： Thinker & Performer

成员： 乔波(软件 44 2141601066)

武晗(软件 44 2141601058)

寻医农(软件 44 2141601070)

实验报告提交日期： 2016/7/17

联系电话： 乔波 18292876310

武晗 13992896358

寻医农 18629045069

目录

一、 实验名称 1
二、 需求及规格说明 1
1. 需求分析 1
2. 规格说明 1
1) 模块间关系 1
2) 内存管理接口与具体算法实现 2
3) 用户输入指令解析与内存管理器交互 3
4) 异常归类与处理 4
5) 可视化内存池状态管理 5
三、 设计 6
1. 设计思想 6
1) 伙伴方法的设计 6
2) 顺序适配方法的设计 6
2. 设计表示 8
1) 伙伴方法中的结构 8
2) 顺序适配的结构 9
3. 实现注释及表示 10
四、 调试报告 11
1. 问题及解决方案 11
2. 设计与编码回顾 12
1) 问题定义与需求分析 12
2) 设计与编码 12
3) 测试 12
3. 时空分析 13
1) 时间性能 13

2) 空间性能 13
4. 改进设想 13
1) 顺序适配方法中提高检索空闲块效率的方法 13
2) 对于顺序适配方法链表的优化 13
五、 运行结果展示 13
1. 字符界面运行截图 14
1) 伙伴方法 14
2) 顺序适配 15
2. 可视化界面运行截图 17
1) 初始界面 17
2) 功能概览 18
3) 内存池划分区块 18
4) 变量列表 19
5) 操作历史 19
6) 控制台 20
7) 放大功能 20
六、 实验总结 21
1. 实验开设意义、重要性、必要性 21
2. 实验收获 21
3. 实验题目意见与建议 22
4. 致谢 22
七、 参考文献 22

数据结构与算法综合训练实验报告

一、 实验名称

本次实验的名称为：动态存储管理器的简易实现及内存分配状态状态可视化展示。

二、 需求及规格说明

1. 需求分析

实验题目要求实现一个动态存储管理器的模拟程序，该程序可以模拟如下任务：

- 任意大小存储空间的分配；
- 已分配的存储空间的回收；
- 用图形或者其他的方式展现每次分配空间操作、或者回收空间操作之后存储池的状态。

具体要求实现的操作命令包括初始化存储池（*init*）、申请变量空间（*new*）、删除变量（*delete*）、为变量赋值（*write*）、读取变量值（*read*）等，并且用可视化的方式展示存储池和操作的状况，同时对可能出现的异常情况做出处理。

在以上实验要求的基础上，经过小组讨论，考虑到对于语言的掌握程度、实现模拟分配算法以及实现可视化界面显示等因素，在实验中以 Java 作为实现语言，对用于内存管理的伙伴方法（*Buddy Method*）与顺序适配方法（*Sequential Fitting Method*）进行演示性的实现，其中顺序适配方法分别使用三种空闲内存区块查找策略进行实现，即首先适配（*First Fitting*）、最优适配（*Best Fitting*）与最差适配（*Worst Fitting*）。同时在实验过程中使用 JavaFX 框架作为可视化界面基础，实现一个对于程序管理的内存池的状态进行展示的可视化程序。

2. 规格说明

对于本次实验中要求的存储空间动态管理功能，实验过程中使用内存中连续空间分配（具体使用 Java 语言环境中的数组作为内存池进行分配管理演示）并在用户手动输入的指令进行内存空间分配与回收，实现具体划分为下列模块，即内存池管理接口与具体算法实现、用户输入指令解析与内存管理器交互、异常归类与处理和可视化内存池状态管理。下面对各部分的功能进行说明。

1) 模块间关系

动态存储管理器中实现的功能及各个模块之间的联系如下图所示：

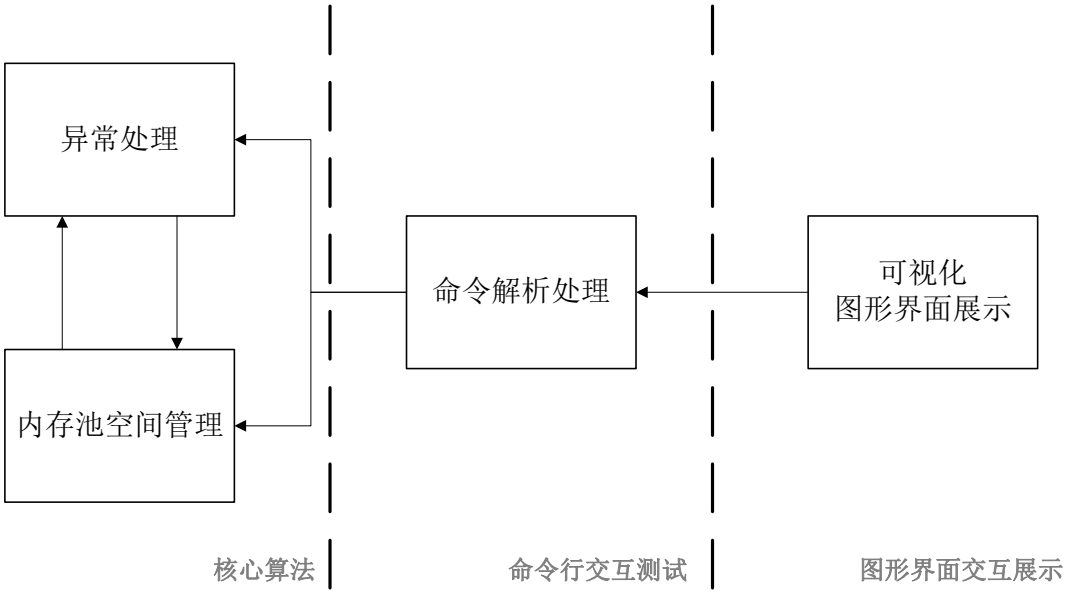


图 2-1 动态存储管理器模块联系图

2) 内存管理接口与具体算法实现

内存管理部分包括内存管理与空间分配两部分，对于一个最简单的实用的内存空间分配模块，需要提供获取空间与释放空间的两个方法即可，此外还需要提供初始化内存池的接口，该模块在其内部管理维护内存池状态。对于内存空间分配管理需要管理用户定义的变量，使用提供数据便捷的写入读出的接口。

根据实验的要求，作为实验过程的核心的内存管理接口及实现需要提供一下操作：

- 初始化：提供接口控制初始化内存池，初始化时需要提供一个占用空间下限，具体的实现在收到初始化请求之后可以通过向操作系统或其他中间运行环境申请需要的空间，同时为了满足特定内存管理的实现方式的需要，可以向环境申请多余参数说明的空间的内存。当具体的实现认为请求的初始化空间不合理时可以抛出异常；
- 空间分配：提供接口返回一块连续的内存空间，其空间大小必须大于或等于请求的数量；
- 空间释放（回收）：当请求的变量使用完成之后，将变量的空间重新进行管理，并在之后空间分配请求中可以在此使用；
- 变量的定义：请求内存管理器调用内存分配器分配一定的空间并将这一段空间与一个变量进行关联；
- 数据存入：将数据存储在在分配的内存空间之中，存储的数据类型限定为字符串；

- 数据读取：将存储在与变量相关联的内存空间中的字符串读出并返回。

3) 用户输入指令解析与内存管理器交互

用户输入指令可以作为命令行交互界面实现程序的测试，需要实现的命令解析及相应格式规范与内存管理接口相对应，具体说明如下：

- 初始化存储池（*init*）：初始化指定大小的存储池。
  - 格式：`init` *memoryPoolSize*
  - 参数：*memoryPoolSize* – 整型，存储池最小占用空间数量，具体合法取值范围由具体实现决定，至少为正数。
- 申请变量空间（*new*）：申请至少为指定大小的空间并赋于指定名称的变量。
  - 格式：`new` *variableName* `=` *variableSize*
  - 参数 1：*variableName* – 字符串，变量名称，由数字、字母、下划线以及货币符号组成的以字母开始区分大小写的字符序列[1] 。
  - 参数 2：*variableSize* – 整型，为新变量分配的空间数量下限。
- 删除变量（*delete*）：在存储池中删除指定名称的变量并回收空间。
  - 格式：`delete` *variableName*
  - 参数：*variableName* – 字符串，已定义的变量名称，要求同上。
- 为变量赋值（*write*）：将指定内容存储于指定变量中。
  - 格式：`write` *variableName* `= "`*variableContent*`"`
  - 参数 1：*variableName* – 字符串，已定义的变量名称，要求同上。
  - 参数 2：*variableContent* – 字符串，将存储于变量中的内容，对于字符“""”与“\”需要转义输入。
- 读取变量值（*read*）：读取指定名称变量的存储内容。
  - 格式：`read` *variableName*
  - 参数：*variableName* – 字符串，已定义的变量名称，要求同上。

[1] 在实现的命令解析模块中使用正则表达式“`[A-Za-z]\w*`”进行变量名合法性判断判断。

4) 异常归类与处理

关于处理存储管理器在进行每个操作的过程中可能存在的各种异常，首先考虑所有的操作，尤其是需要用户输入的操作，因为“一切输入都是有害的”，其次判断各种操作可能出现的异常，然后对这些异常进行分类，建立合适的异常类型，最后应用于程序中的各个操作模块。下面展开讨论。

首先，考虑所有的操作，基本可按时序分为两个阶段：第一阶段——用户输入命令，由命令解析器 Parser 进行分析；第二阶段——Parser 将解析后的指令传递给存储管理器 MemManager 执行。下面考虑两个阶段的操作中可能出现的异常情况：

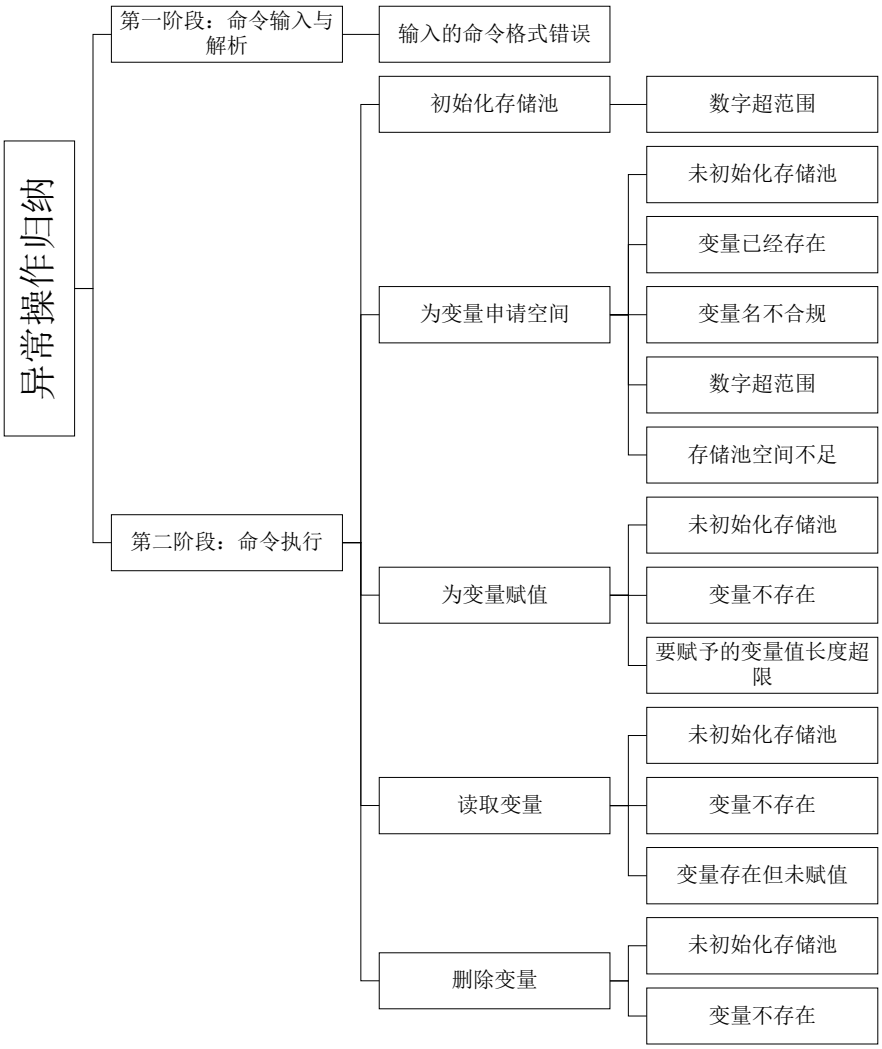


图 2-2 异常操作分类与归纳

其次，根据以上的列举结果进行归纳分析，将异常情况分成三个大类：空操作对象、参数错误、空间不足，再分为若干个子类型，具体分类情况如下：

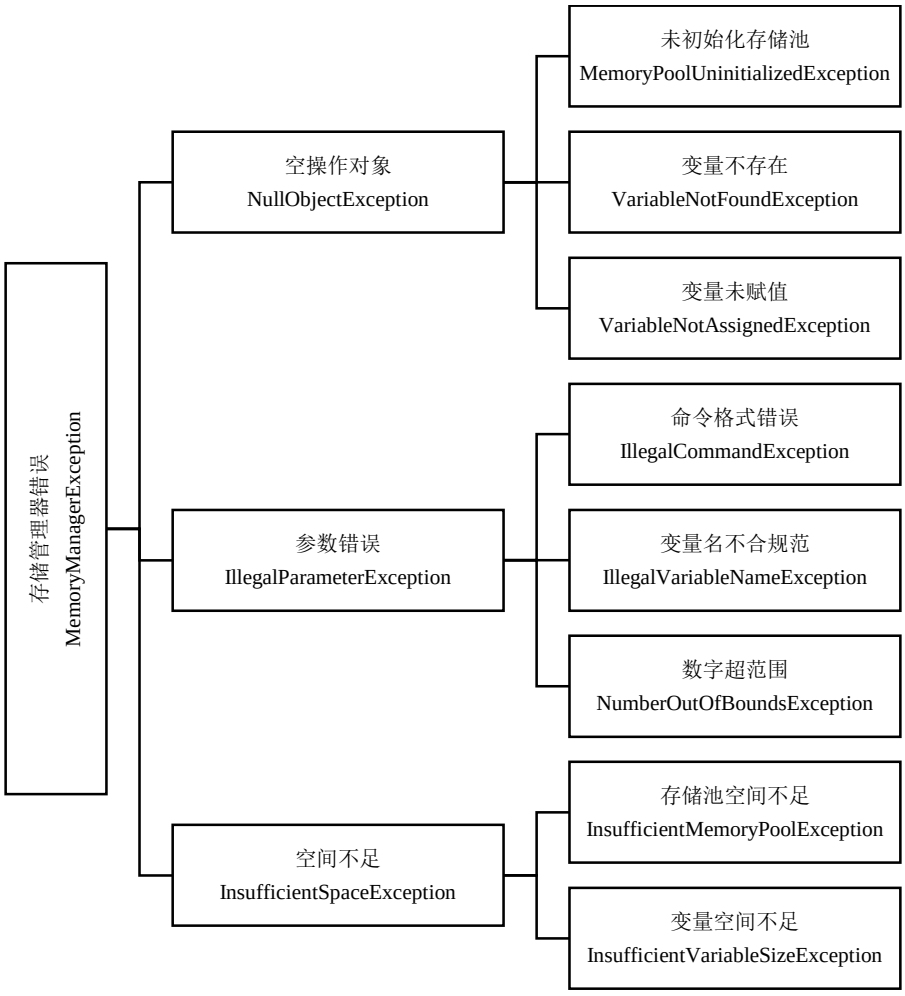


图 2-3 自定义异常类型及继承关系

最后，在将上述异常类型应用于程序中的各个操作模块时，根据的原则是：模块负责什么，就检测哪部分的异常。例如，命令解析器（Parser）负责解析用户输入的命令，故 Parser 负责判断用户输入的命令是否符合规范，如果用户在没有初始化存储池的情况下输入“new a=20”，那么 Parser 任然认为这则命令是正确的，至于存储池是否已经初始化，则交给存储管理器（MemManager）进行判断，如果未初始化，则由 MemManager 抛出未初始化存储池异常（MemoryPoolUninitializedException）。再例如，如果用户已经初始化存储池后输入“new a=20”，那么存储管理器（MemManager）就照此执行，至于存储池的剩余空间是否足够分配给变量 a，MemManager 不予理睬，交给存储池分配器（Allocator）进行判断，如果存储池的剩余空间不足，则由 Allocator 抛出存储池空间不足异常（InsufficientMemoryPoolException）。

5) 可视化内存池状态管理

内存池中需要存储大量的数据，在经过一段时间的使用之后，内存池中将会存在不同大量的大小的内存片段，为了维护大量的内存片段，内存分配算法需要在内存池中占用一

定量的空间对这些片段加以管理。当内存池容量较小时，可以通过控制台打印的方式输出内存池状态，而当内存池容量较大时，使用控制台的文字输出将会变得很不直观，需要使用图形化的方式展示内存池中每个位置的用途及状态。

三、 设计

该部分主要描述核心内存分配管理模块的设计与实现细节。

1. 设计思想

对于内存分配管理核心模块，采用不同分配方式时有不同的设计。

1) 伙伴方法的设计

伙伴方法中通过一种简化的空间分配方案简化了分配过程中需要维护的复杂数据结构，以及处理过程中的复杂逻辑。基本思想是将内存块的大小限定为2^k ，并且每一个大小为2^k 的内存块都是由一个2^{k+1} 的内存块划分得到，在划分的过程中得到的另一半内存块也是一个大小为2^k 内存块，即为伙伴。

这样的划分可以得到简化的基本操作：

- 分配：确定需要内存区块的大小（确定2^k 中的k ），查找是否存在这样的内存块（记录合适第一个大小的内存块的位置），若存在则将其分配，并移除空闲列表，否则将k 更大的内存块进行分割，之后分配一块，将另一块添加到相应的空闲列表，如果没有合适的更大的内存块，则分配失败；
- 回收：确定将回收的内存块的伙伴内存块是否空闲，若空闲，则合并内存块，得到更大的内存块，重复该过程，直至伙伴内存块被占用（或不存在、被分割为更小的内存块），这是将最终的空闲块添加的到空闲表中。

使用伙伴方法的缺点在于内存的使用效率较低，当空间不是2^k 时会分配到最接近的更大的2^k 内存块，而且内存块之间的分配在物理上限定为“伙伴”位置，更可能出现空闲空间无法被利用的情况。

2) 顺序适配方法的设计

顺序适配方法取消了对每个内存块大小的限制，使得内存块大小可以为任意值，而这样在处理和维护内存池的过程变得较为复杂。

关于删除变量的方法。考虑删除一个变量并释放它占用的存储块时，共有 9 种可能的情况如下：

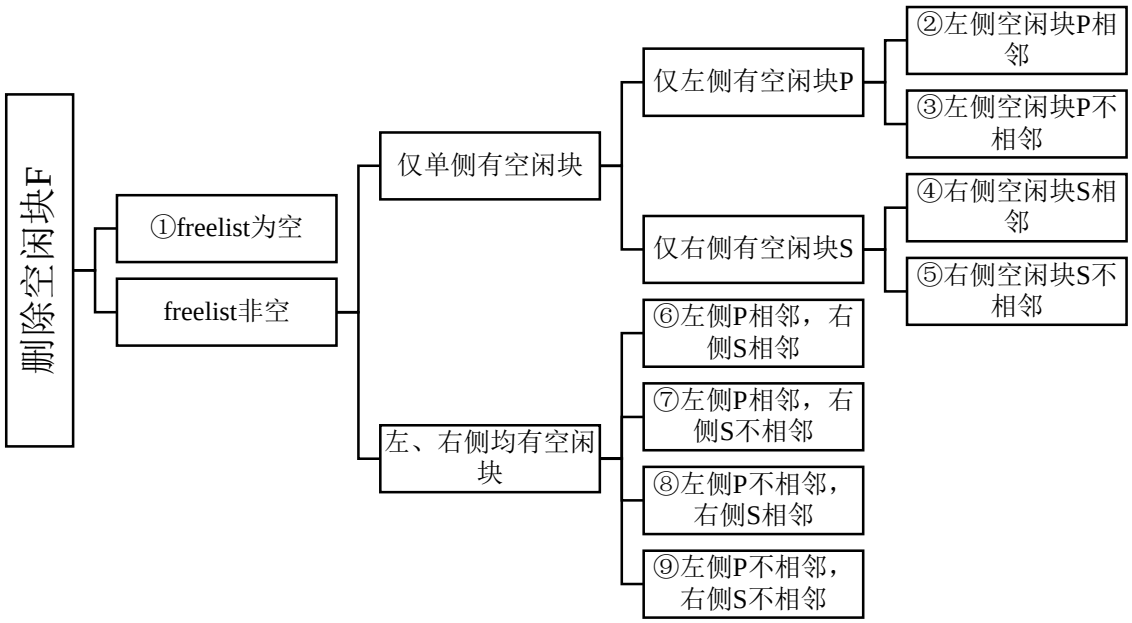


图 3-1 顺序适配方法中空闲块的删除操作流程

分别采用如下的处理方法：

①freelist 为空：将 F 标记为唯一的空闲块，并令 freelist 指向 F；

②左侧空闲块 P 相邻：将 P 与 F 合并；

③左侧空闲块 P 不相邻：将 F 标记为新的空闲块，调整 P 的指针，调整原本左指针指向 P 块的指针；

④右侧空闲块 S 相邻：将 S 与 F 合并，并调整原本指向 S 块的指针，令 freelist 指向 S 与 F 合并后的块；

⑤右侧空闲块 S 不相邻：将 F 标记为新的空闲块，调整原本右指针指向 S 块的指针；

⑥左侧 P 相邻，右侧 S 相邻：将 P、F、S 合并，调整原本指向 S 块的指针，，令 freelist 指向 P、F、S 合并后的块。

⑦左侧 P 相邻，右侧 S 不相邻：将 P 与 F 合并；

⑧左侧 P 不相邻，右侧 S 相邻：将 S 与 F 合并，调整原本指向 S 块的指针，令 freelist 指向 S 与 F 合并后的块；

⑨左侧 P 不相邻，右侧 S 不相邻：将 F 标记为新的空闲块，并调整 P、F、S 的指针。

关于顺序适配的其它方法，使用了教材中的实现，且十分简单朴素，此处不再复述。

这种方法的空间利用率要高于伙伴方法但是在查找空闲块与回收占用块需要消耗较高的成本。

2. 设计表示

1) 伙伴方法中的结构

在设计存储的数据结构的过程中，考虑到应用环境中维护分散的数据会使得问题的焦点变得分散，也在使用辅助数据结构时产生对于特定语言环境的较强的依赖。在设计此部分的表现结构时，将所有相关数据均存储在内存池中，不占用其他数据存储状态信息。

对于伙伴方法的数据结构设计，在初始化内存时限定申请的内存块大小为$2^k + C$ 其中C为一个常数，表示存贮在内存池最起始位置的一些元信息，包括整个内存块的大小，内存块中不同k值的空闲块的起始位置。

内存池的分配结构如下示意图说明：

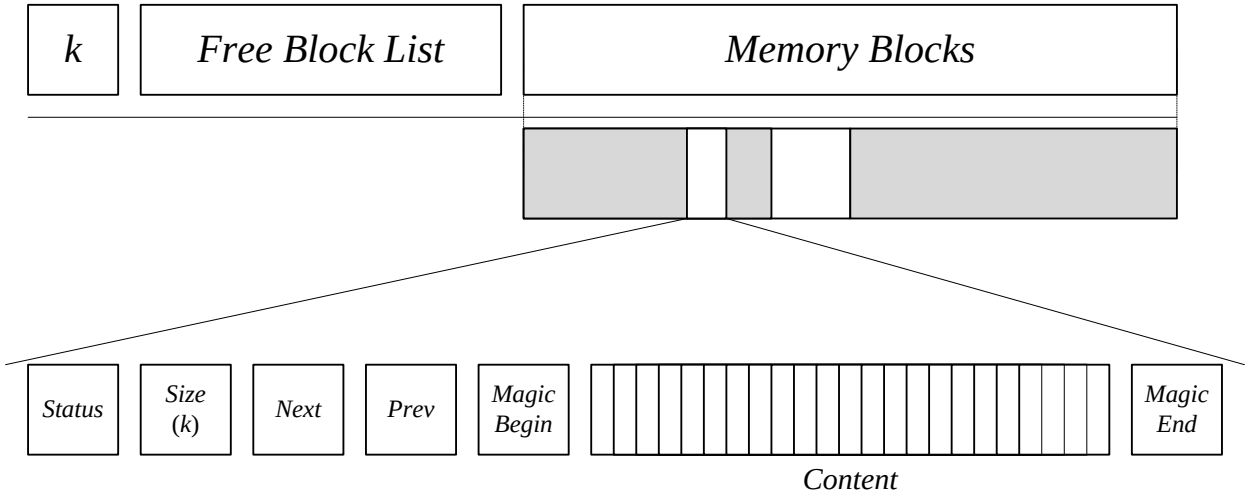


图 3-2 伙伴方法内存结构

在图中内存划分为总内存大小标识（*k*）、不同大小的内存块的第一个空闲块位置列表（*Free Block List*）和分配内存块（*Memory Block*）三部分。其中对于每一个用于分配的内存块，结构包括信息部分与数据部分（图中每一个正方格表示一个字节或一个字[2]），状态（*Status*）位记录是否为空闲块，大小（*Size*）位记录当前块的大小，之后的两个位（*Next*，*Prev*）对于空闲块用于构建一个连接相同 *k* 值的空闲块的双向链表，对于占

[2] 由于空闲列表需要存储指针（下标），从而每一个单元的大小取决于内存池的大小（存储数组下标或相对位置）或机器的体系架构（直接存指针）。

用的内存块没有意义[3]，再之后的部分是数据块，在演示中为了可视化显示的方便直观，添加了标志位标记内容的起始与终止[4]。

2) 顺序适配的结构

在顺序适配方法中，由于每个内存块的大小都不相同，无法通过对不同大小的空闲块均建立空闲空间列表实现空闲空间的分配，在数据结构的设计中只记录了空闲空间的起始，其余部分均为用于分配的数据块，具体设计如下如所示。

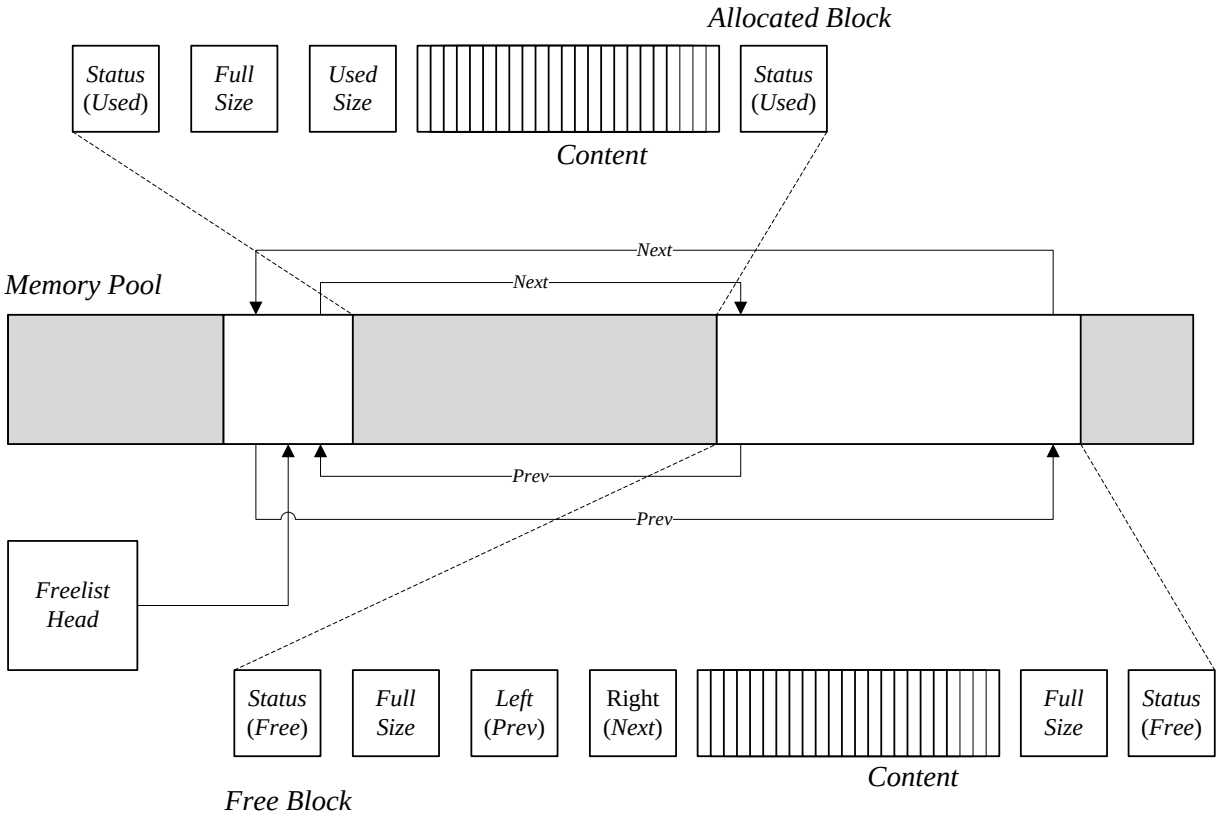


图 3-3 顺序适配方法内存块结构

在该数据结构中，空闲块之间通过记录前后空闲块的位置串接成双向循环链表。对于空闲块，在可用空间之外记录了前后空闲块的指针（数组下标）、状态和可用空间大小，对于占用块，记录了状态、可用空间大小和已用空间大小。

此外，在此基础上设置一个指针（或数组下标）记录上次分配的位置可以在一些场合获得查找效率的提升。

[3] 更符合实际的处理中可以将这两块进行数据的存储。

[4] 标志位在一些场合也用于进行一定的空间访问越界检测，例如在 C\C++中可以通过指针直接修改任意内存位置，从而有可能结构

3. 实现注释及表示

实现内存分配的抽象数据类型（*ADT*）如下代码所示（使用 Java 中接口的形式表示）：

```
public interface AllocatorADT {
    /*初始化存储池*/
    void init(int size) throws NumberOutOfBoundsException;
    /*为变量申请空间*/
    Variable newVariable(String variableName, int size) throws InsufficientMemoryPoolException;
    /*向变量存储空间写入数据*/
    void write(Variable variable, String value) throws InsufficientVariableSizeException;
    /*从变量存储空间读数据*/
    String read(Variable variable) throws VariableNotAssignedException;
    /*删除变量*/
    void deleteVariable(Variable variable);
    /*展示存储池*/
    void show(AllocatorADT allocator, List<String> sortVariableList);
}
```

对于以上 ADT，顺序适配方法的实现中使用的方法及相应说明如下表所示：

方法	说明	时间复杂度
AllocatorSequential(int size)	构造方法，调用 init 方法初始化存储池	$O(1)$
void init(int size)	初始化存储池	$O(1)$
Variable newVariable(String variableName, int size)	为变量申请空间	首先适配：$O(n)$ 最佳适配：$O(n)$ 最差适配：$O(n)$
int pickFreeBlock (int size)	查找空闲块（首先适配）	$O(n)$
int pickFreeBlock (int size)	查找空闲块（最佳适配）	$O(n)$
int pickFreeBlock (int size)	查找空闲块（最差适配）	$O(n)$
boolean write (Variable var, String val)	向变量存储空间写入数据	$O(m)$，m 为变量值长度，忽略 m 的长度则为 $O(1)$
String read(Variable variable)	从变量存储空间读数据	$O(m)$，m 为变量值长度，忽略 m 的长度则为 $O(1)$
void deleteVariable (Variable variable)	删除变量	$O(n)$
void show(AllocatorADT a, List<String> sortVariableList)	展示存储池	

对于伙伴方法的实现，相应的方法说明如下：

方法	说明	时间复杂度
AllocatorSequential(int size)	构造方法，调用 init 方法初始化存储池	$O(1)$
void init(int size)	初始化存储池	$O(1)$
Variable newVariable(String variableName, int size)	为变量申请空间	$O(1)$ [5]
boolean write (Variable var, String val)	向变量存储空间写入数据	$O(m)$，m 为变量值长度，忽略 m 的长度则为 $O(1)$
String read(Variable variable)	从变量存储空间读数据	$O(m)$，m 为变量值长度，忽略 m 的长度则为 $O(1)$
void deleteVariable (Variable variable)	删除变量，并回收空间	$O(1)$ [6]
void show(AllocatorADT a, List<String> sortVariableList)	展示存储池中的内存块的详细信息以及相应空闲块列表	

显然，伙伴方法在处理各种情况的操作过程中均可以很高效的完成。

四、 调试报告

1. 问题及解决方案

在编码实现算法结构过程中，遇到了一些问题，通过小组讨论、查阅图书、网络搜索等方式的得以解决。下面列举部分问题及其解决方案。

- 存储池分配器 `allocator` 的 `show()`方法中获取变量的名字

问题说明：程序中，用户定义的变量的名字直接决定了其对应的 `Variable` 类的对象的名字，但变量名并没有存储在存储池分配器 `allocator` 中；而 `allocator` 的 `show()` 方法在打印区块信息时，如果是被占用块，则希望打印变量的名字。

解决方案：将存储变量的名字的 `HashMap` 的引用作为一个参数，由 `memoryManager` 传递给 `allocator`，从而使得 `allocator` 可以获取变量的名字。

- 在使用 JavaFX 中 Canvas 元素绘图时，随着内存块的增加，出现严重卡顿现象。

[5] 如果考虑内存池的空间大小，在最差的情况下需要多次分解较大的内存块直至得到合适大小的内存块，这将为 $O(\log s)$ 其中 s 为内存池的大小，即为 $O(k)$，k 为常数。

[6] 同上说明

问题说明：在使用 Canvas 进行绘图过程中，由于未进行优化，绘图效率过低，造成卡顿现象。

解决方案：类似于普通程序的优化，首先确定程序中最消耗资源（运行速度最慢）的位置，然后在相应位置分析原因，进行优化。在实现放大镜效果中，放大镜使用 Clip 放大裁切得到，该方法在实现上需要对每一次元绘图操作中每一个像素点的位置进行判断，需要很大的计算量，从而大大增加了渲染一帧需要的时间。为此采用限制 Clip 函数作用的区域进行优化。如下如所示，最初绘制的区域为整个圆形的外切正方形内全部图示单元，首先优化处理了距离过大的位置，减少了绘制调用次数，之后又进一步优化，对于内部的不需要裁切的单元直接进行绘制（此处略去相应绘制以示差异）。

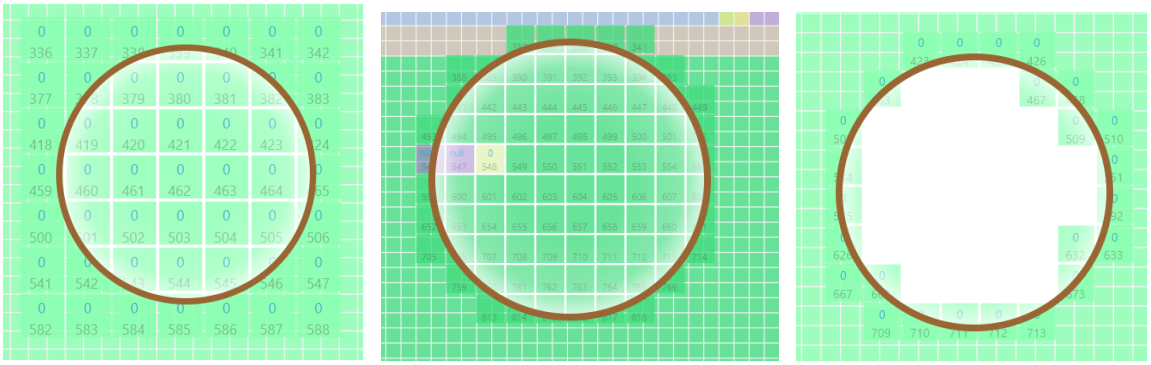


图 4-1 Clip 函数裁切区域优化

2. 设计与编码回顾

1) 问题定义与需求分析

本次实验中采用了软件工程的方法，通过小组讨论，首先根据题目要求明确问题的定义，即一个模拟的动态存储管理器。然后进行需求分析，并建立规格说明书，从而为后续的设计与编码提供一个完整和一致的概念。

2) 设计与编码

首先进行系统设计，将实验过程分为算法学习、核心算法实现、异常处理与命令行交互实现以及可视化界面的实现。设计过程中以小组讨论的形式为主，小组成员共同确定实现方式。

在编码实现过程中使用 Git 进行代码库的同步与版本的管理，Git 工具的使用可以方便的整合不同组员的代码，及时检查指出其他组员的疏漏之处。

3) 测试

首先进行单元测试，检查并修改模块中的错误，从而保证每个模块都正常工作。然后进行集成测试，检查并修改系统中的错误，从而保证系统正常工作。当以上小组人员的测试结束后，邀请其他人对程序进行试用，进一步检查系统中的错误。

在测试的过程中，对发现的错误根据其类型进行修改，如果是设计上的错误，就对代码进行必要的重构，如果是操作不合规范导致的错误，就增设相应的异常处理操作。

3. 时空分析

1) 时间性能

时间性能分析已在 3.3 实现注释一节中作了说明，此处不再重复说明。

2) 空间性能

- 顺序适配方法

顺序适配中额外消耗的空间均直接处于组织维护内存块，由于每个内存块中的额外消耗的空间数量固定，使用顺序适配方法处理空间利用率很高，而且如果分配的内存较大，则效率会更高一些。

- 伙伴方法

伙伴方法存在着为了使分配空间对齐到 2^k 而需要添加的不可利用的空间，由于空间的管理均按照 2^k 进行处理，无法利用剩余空间。如果请求的数据块近似均匀分布，则将有 1/4 以上的空间将被闲置而无法利用。

4. 改进设想

1) 顺序适配方法中提高检索空闲块效率的方法

实验中，检索空闲块采用了直接遍历的方式，该情况下，时间复杂度为 $O(n)$ 。考虑将空闲块的组织方式改为 BST 树，通过在每个存储池中维护一个 BST 树，将空闲块的大小作为排序的关键字。从而，可以将首先适配、最佳适配、最差适配的时间时间性能提高至 $O(\log n)$ 。

2) 对于顺序适配方法链表的优化

在顺序适配方法中，空闲块使用双向非循环链表进行连接，可以实现在回收空间过程中空间的快速收集，即判断是否可以与左右内存块合并，如果可以合并，则将合并内存块从空闲块链表中删除，然后合并，将最终合并的结果插入在空闲块链表起始。这样并不会影响空闲块的查找效率，但可以简化删除的实现逻辑。

五、 运行结果展示

整个程序的运行界面包含字符界面的控制界面与图形界面的可视化交互界面，运行结果截图在本章节中呈现。

1. 字符界面运行截图

1) 伙伴方法

- 初始化内存池

```
init 1000
FreeList
  1: NULL;  2: NULL;  3: NULL;  4: NULL;  5: NULL;  6: NULL;
  7: NULL;  8: NULL;  9: NULL; 10:   31;
Memory Allocation:
   31(    0) : F : 10
```

- 添加变量并分配空间

```
new a = 10
FreeList
  1: NULL;  2: NULL;  3: NULL;  4:   47;  5:   63;  6:   95;
  7:  159;  8:  287;  9:  543; 10: NULL;
```

- 写入数据

```
write a = "good"
FreeList
  1: NULL;  2: NULL;  3: NULL;  4:   47;  5:   63;  6:   95;
  7:  159;  8:  287;  9:  543; 10: NULL;
Memory Allocation:
   31(    0) : B : 4 : a
   47(   16) : F : 4
   63(   32) : F : 5
   95(   64) : F : 6
  159(  128) : F : 7
  287(  256) : F : 8
  543(  512) : F : 9
```

- 读出数据

```
read a
good
FreeList
  1: NULL;  2: NULL;  3: NULL;  4:   47;  5:   63;  6:   95;
  7:  159;  8:  287;  9:  543; 10: NULL;
Memory Allocation:
   31(    0) : B : 4 : a
   47(   16) : F : 4
   63(   32) : F : 5
   95(   64) : F : 6
  159(  128) : F : 7
  287(  256) : F : 8
  543(  512) : F : 9
```

- 删除变量

```
delete a
FreeList
  1: NULL;  2: NULL;  3: NULL;  4:   31;  5:   63;  6:   95;
  7:  159;  8:  287;  9:  543; 10: NULL;
Memory Allocation:
   31(    0) : F : 4
   47(   16) : F : 4
   63(   32) : F : 5
   95(   64) : F : 6
  159(  128) : F : 7
  287(  256) : F : 8
  543(  512) : F : 9
```

2) 顺序适配

- 初始化内存池

```
init -10
数字超范围
init 50+26
数字格式错误
init 1000
-------------------------------------------------------------------------------------------------
|    block    |  status  |   size   | used size | variable name | left pointer | right pointer |
-------------------------------------------------------------------------------------------------
|    0 -  999|   free   |    994   |           |               |        0     |         0     |
-------------------------------------------------------------------------------------------------
```

- 添加变量并分配空间

new a = 10
未初始化存储池
init 1000

block	status	size	used size	variable name	left pointer	right pointer
0 - 999	free	994			0	0

new a = -5
数字超范围
new a = 10000
(顺序适配方法——最佳适配)
存储池空间不足
new a = 3.14
无法识别的数字
new 6a = 10
变量名不合规范
new a = 10
(顺序适配方法——最佳适配)

block	status	size	used size	variable name	left pointer	right pointer
0 - 985	free	980			0	0
986 - 999	reserved	10	0	a		

new a = 25
变量已存在

- 写入数据

read c
变量不存在
read b
变量未赋值
read a
hello

block	status	size	used size	variable name	left pointer	right pointer
0 - 956	free	951			0	0
957 - 985	reserved	25	0	b		
986 - 999	reserved	10	5	a		

- 读出数据

write a = "helloworld!"
变量空间不足
write a = "hello"

block	status	size	used size	variable name	left pointer	right pointer
0 - 985	free	980			0	0
986 - 999	reserved	10	5	a		

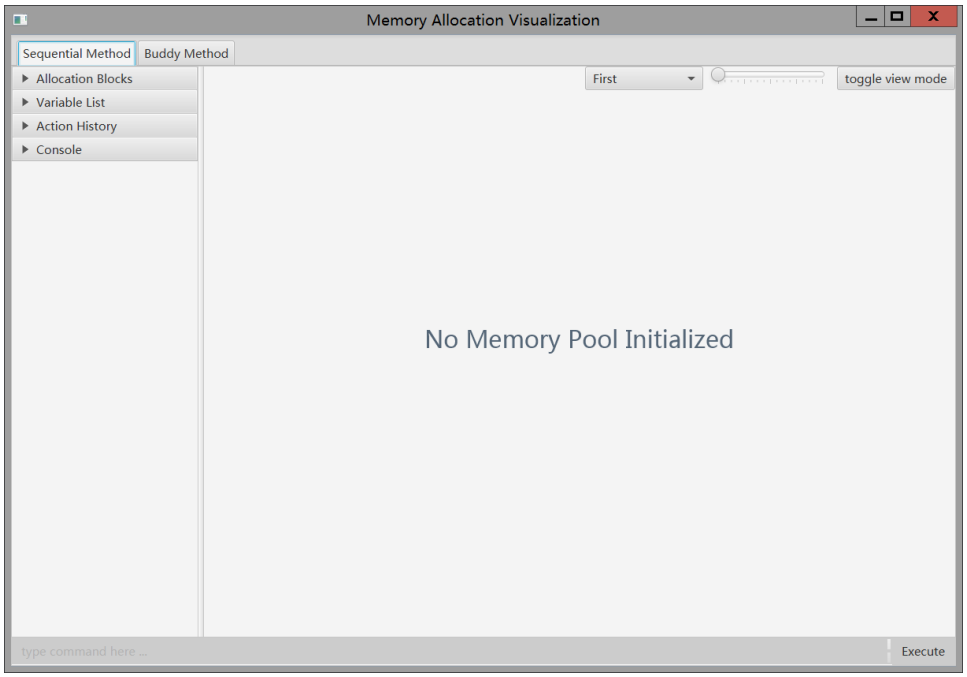
- 删除变量

delete c
变量不存在
delete b

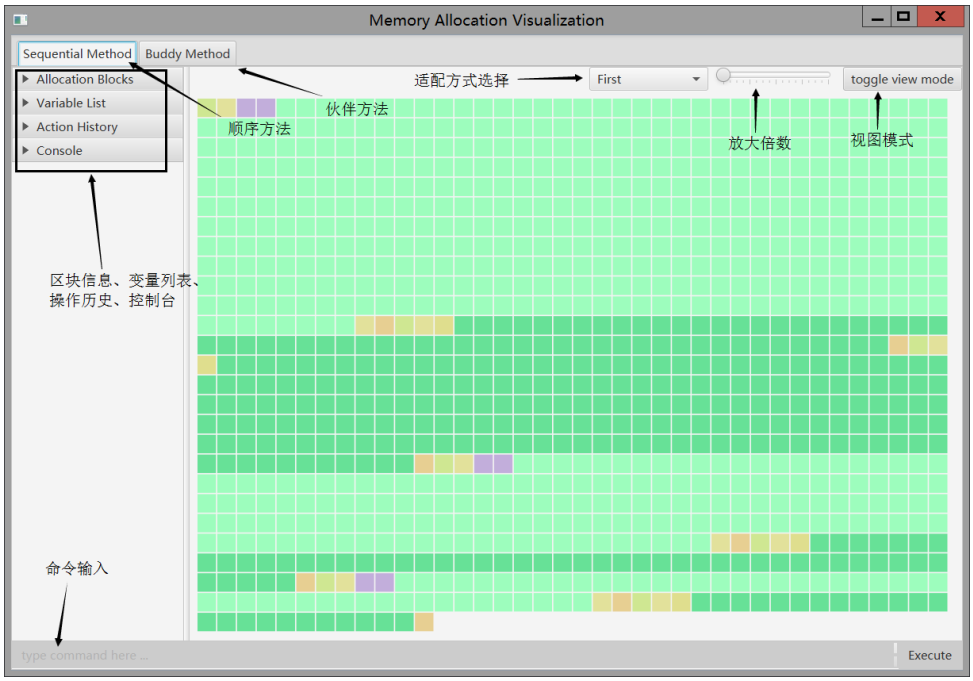
block	status	size	used size	variable name	left pointer	right pointer
0 - 985	free	980			0	0
986 - 999	reserved	10	5	a		

2. 可视化界面运行截图

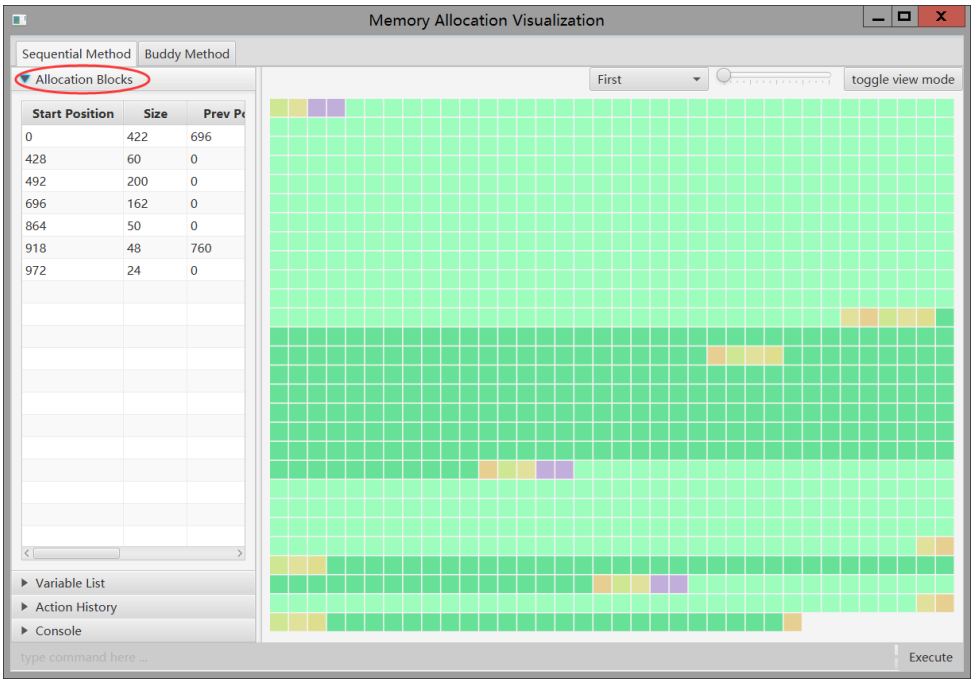
1) 初始界面



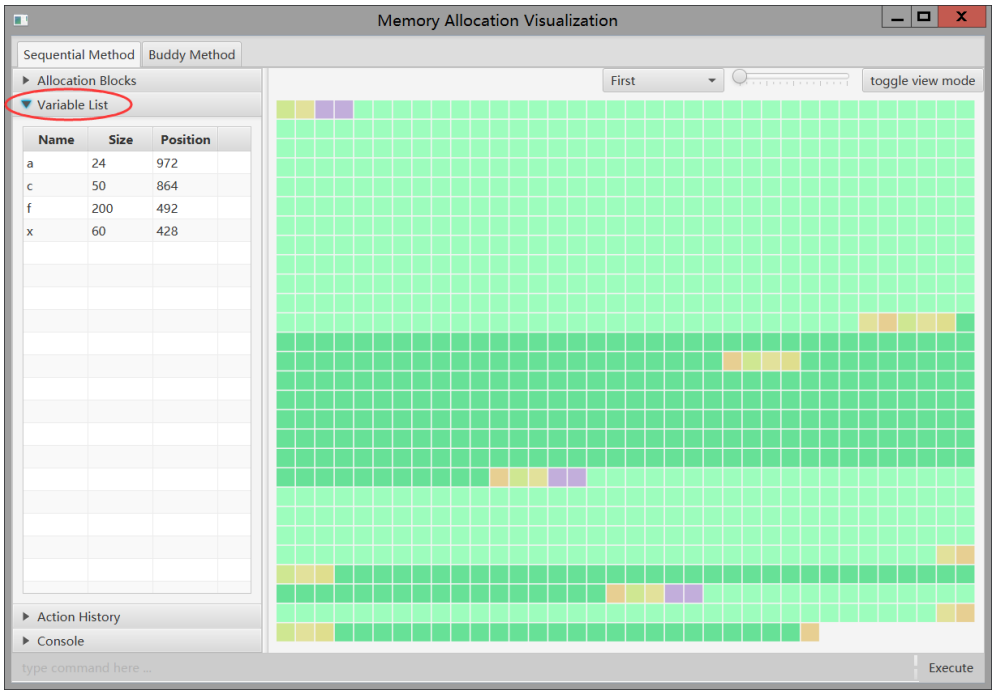
2) 功能概览



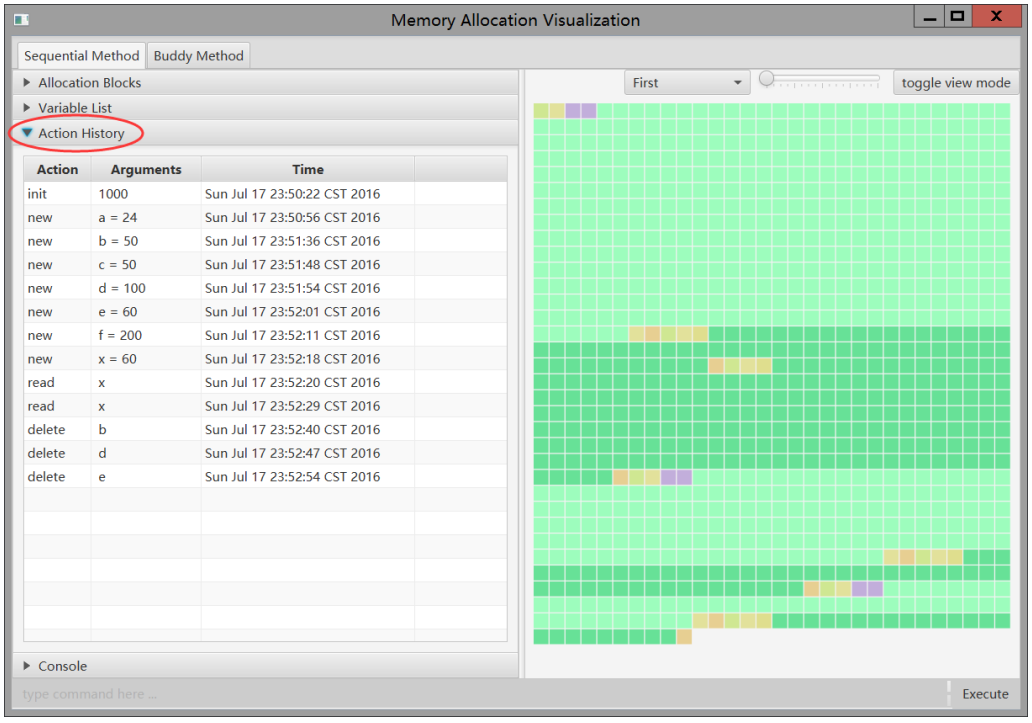
3) 内存池划分区块



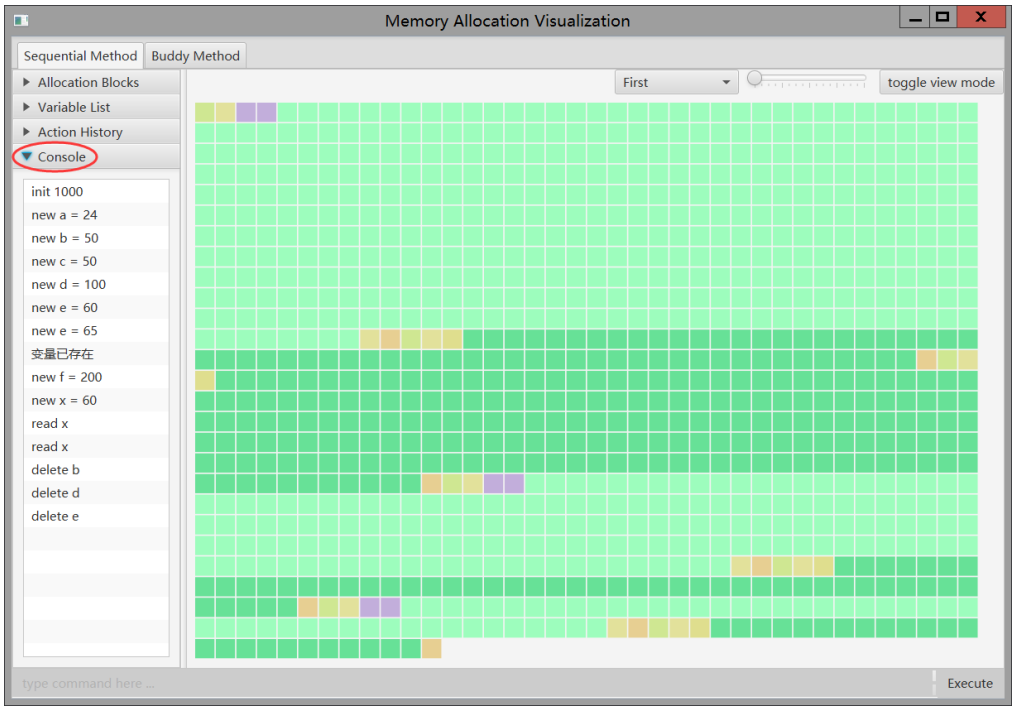
4) 变量列表


Memory Allocation Visualization
Sequential Method
Buddy Method
Allocation Blocks
Variable List
Name
Size
Position
a
24
972
c
50
864
f
200
492
x
60
428
Action History
Console
type command here ...
First
toggle view mode
Execute

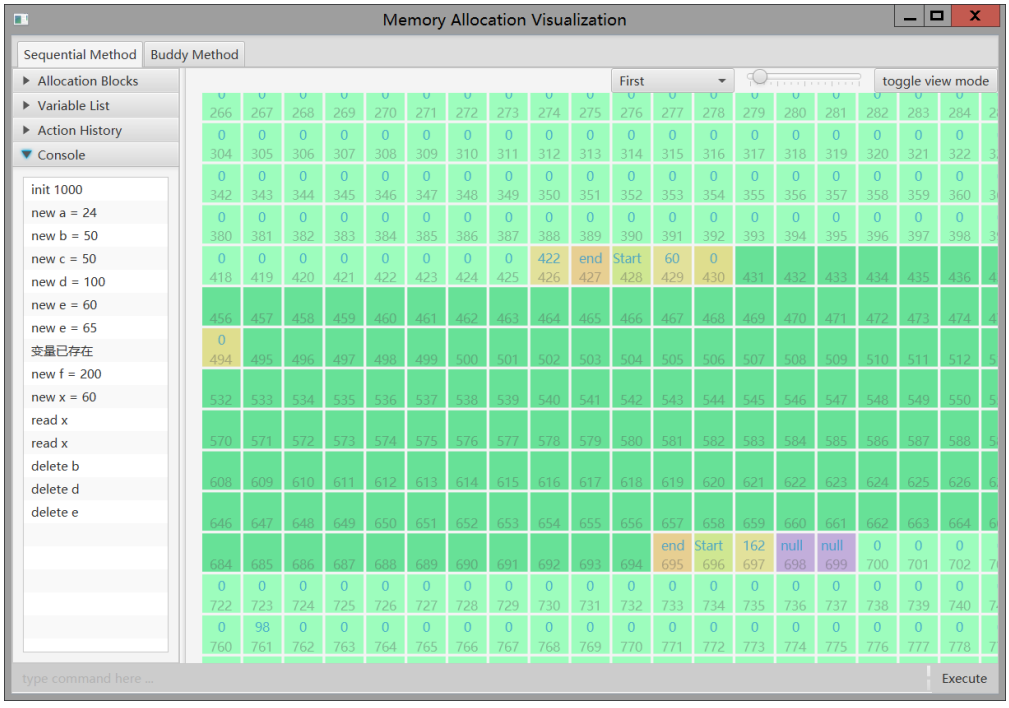
5) 操作历史

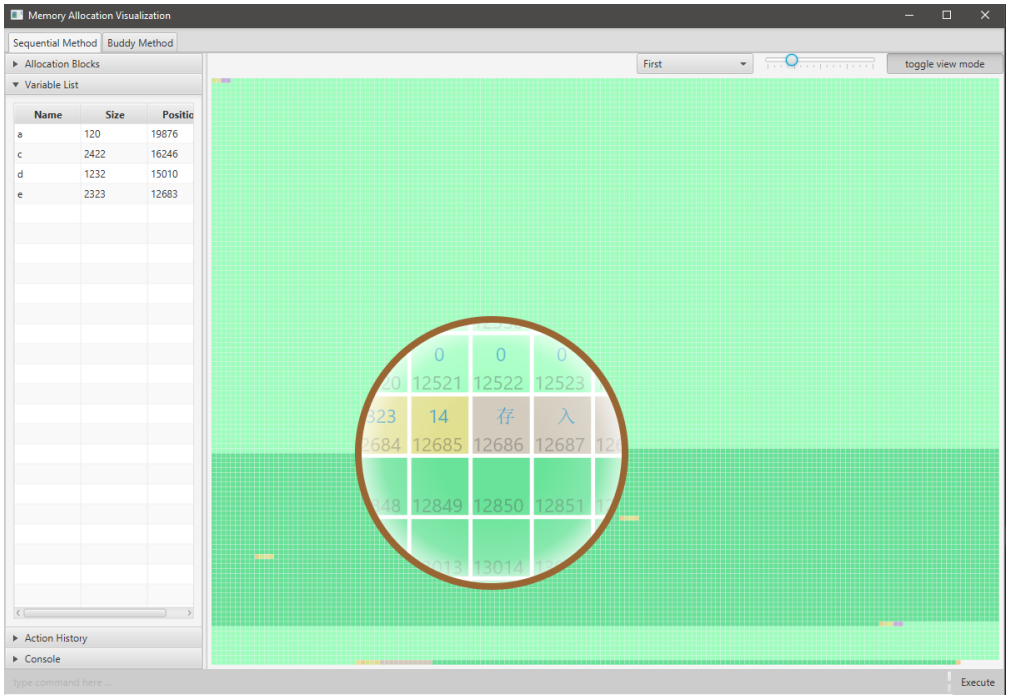

Memory Allocation Visualization
Sequential Method
Buddy Method
Allocation Blocks
Variable List
Action History
Action
Arguments
Time
init
1000
Sun Jul 17 23:50:22 CST 2016
new
a = 24
Sun Jul 17 23:50:56 CST 2016
new
b = 50
Sun Jul 17 23:51:36 CST 2016
new
c = 50
Sun Jul 17 23:51:48 CST 2016
new
d = 100
Sun Jul 17 23:51:54 CST 2016
new
e = 60
Sun Jul 17 23:52:01 CST 2016
new
f = 200
Sun Jul 17 23:52:11 CST 2016
new
x = 60
Sun Jul 17 23:52:18 CST 2016
read
x
Sun Jul 17 23:52:20 CST 2016
read
x
Sun Jul 17 23:52:29 CST 2016
delete
b
Sun Jul 17 23:52:40 CST 2016
delete
d
Sun Jul 17 23:52:47 CST 2016
delete
e
Sun Jul 17 23:52:54 CST 2016
Console
type command here ...
First
toggle view mode
Execute

6) 控制台


Memory Allocation Visualization
Sequential Method
Buddy Method
Allocation Blocks
Variable List
Action History
Console
First
toggle view mode
init 1000
new a = 24
new b = 50
new c = 50
new d = 100
new e = 60
new e = 65
变量已存在
new f = 200
new x = 60
read x
read x
delete b
delete d
delete e
type command here ...
Execute

7) 放大功能


Memory Allocation Visualization
Sequential Method
Buddy Method
Allocation Blocks
Variable List
Action History
Console
First
toggle view mode
init 1000
new a = 24
new b = 50
new c = 50
new d = 100
new e = 60
new e = 65
变量已存在
new f = 200
new x = 60
read x
read x
delete b
delete d
delete e
type command here ...
Execute



六、 实验总结

1. 实验开设意义、重要性、必要性

数据结构与算法综合训练是对于数据结构课程中学习的基本数据结构的实际运用，也是一次对于团队协作的编程实践，除了设计需事先算法本身需要细致缜密的思考，在设计命令行交互接口以及可视化图形界面的过程中也需要较多的投入，这些对于小组成员的算法思维能力、程序编码能力、团队协作能力都有重要的意义。

本次实验题目要求的方向与操作系统的内存管理有着一定能够程度的联系，在更实用化的算法实现中将需要对系统的底层以及内存等存储器件物理结构有进一步的了解，这些方向的深入将成为之后学期中后继课程的衔接。

2. 实验收获

这次实验过程中一方面收获在于通过学习教材提供的内存分配算法并在其算法基础上进行实现，体会到了简单的链表等数据结构在较复杂的实际场景中的应用，也增进了对于相关的计算机软件系统较为底层的组件的了解。另一方面在使用 Java 语言进行编码实现的过程中对原有 Java 语言的了解更加深入，对于语言的细节的掌握以及 JDK 中提供的工具是使用的熟练程度有了进一步提高。例如，在实验中我碰到这样的一个问题。在父类中，有一个方法 pickFreeBlock 只供其内部的成员方法调用，故一开始设置为 private，其子类在继承该父类后，重写了一个同名方法 pickFreeBlock，但测试结果显示子类调用的是其父类的 pickFreeBlock 方法而非自身的。究其原因，是父类调用的 pickFreeBlock 方法

是父类自己的 pickFreeBlock 方法而非子类的 pickFreeBlock 方法。于是这启示我，要调用子类的同名方法，必须让子类继承该方法并覆盖之。

3. 实验题目意见与建议

实验内容能够涉及到计算机软件体系中与硬件交互较为紧密的部分，对于这个部分的学习了解可以增加对于计算机系统的原理的理解，而且内存管理相关内容与操作系统等后续课程具有较为紧密的联系，这次实验对于后续课程的相关内容提供了引导。在题目的基础上希望老师可以提供一些更具体的实现拓展方向，或相关参考资料，便于实验基础上的进一步探索。

4. 致谢

通过这次项目实践，我们都有了很大的收获，而这些与很多来自他人的帮助密不可分，在此向所有在实验过程中给予帮助和支持的人表示感谢。

首先感谢原盛老师。一个学年的细致讲解与用心导引，使我们对于 Java 语言、Java Web 相关基础技术、基本数据结构、初级的经典算法等知识有了很大的提高，这些内容将与今后课程学习、程序编写实践过程中密切相关。我们在这次实验任务的完成过程中需要的理论基础与老师的授课密不可分。感谢原盛老师这一年的指导和帮助，在大学期间真的是我们难得一遇的好老师。

其次要感谢小组成员之间相互配合共同完成这次实验，项目实现初期，小组成员对项目实现的语言，用于内存管理的伙伴方法与顺序适配方法进行演示性实验的方式，以及决定使用什么框架作为可视化界面基础进行了积极地讨论。以课本为依据，小组成员相互讨论之间学习了动态存储管理器所需的相关知识。在算法及可视化界面的编码实现过程中，成员之间相互讨论确定实现方案，查找程序错误，也互相之间在合作中学到了很多，包括解决问题的思路、具体算法的实现过程以及出现问题后如何调整方向等等，都有收获很大。

再有是要感谢身边的一些同学在课程实验完成过程中的帮助，通过与同学的交流讨论，解决了很多实验过程中所遇到的问题。

七、 参考文献

[1] Y.Daniel Liang 著;戴开宇译.Java 语言程序设计[M].北京：机械工业出版社, 2015.