柔軟性を追求したメモリ管理システム

– ゲーム開発に効果的なメモリ管理 –

2014 年 2 月 24 日 初稿

板垣 衛

■ 改訂履歴

稿	改訂日	改訂者	改訂内容
初稿	2014 年 2 月 24 日	板垣 衛	（初稿）

■ 目次

■ 概略

別紙の「ゲーム制御のためのメモリ管理方針」に基づいて、メモリ管理システムを設計する。

ゲーム要件に合わせた柔軟なメモリ配分、メモリ調整、メモリ使用状況確認を可能とし、状況に合わせた調整を行い易いシステムとする。

なお、本システムに導入する技術および技術用語について、本書には詳しい説明を記載しない。別紙の「ゲーム制御のためのメモリ管理方針」にて、元にした考え方やデータ構造、アルゴリズム、用語などの解説を示す。その前提で、本書では要件を簡潔に記述する。

■ 目的

本書は、ゲームシステムが求めるメモリ要件に柔軟に対応し、ゲーム開発を円滑にすることを目的とする。

■ 要件定義

▼ 基本要件

本書が扱うシステムの基本要件は下記の通り。

- ゲームで使用するメモリ全般を管理するためのメモリマネージャを用意する。
- マルチスレッドで利用可能な、スレッドセーフなメモリマネージャとする。
- メモリマネージャは、「ヒープメモリ」に対応し、自由なメモリ確保を可能とする。
- ヒープメモリは、メモリ確保、および、メモリ削除において、十分に高速に動作するものとする。
- ヒープメモリは、メモリ再配置に対応し、必要に応じて大きな連続空き領域を空けることが可能とする。

・ メモリ管理システムは、メモリ再配置の発生を気にせずメモリにアクセス可能な手段を提供するものとする。
  - スマートポインタのような、あるいは、スマートポインタと統合した仕組みとして、ポインタを隠蔽したアクセス手段を用意する。

・ メモリマネージャは、「プールアロケータ」に対応し、小さなメモリを効率的に、かつ、高速に扱えるものとする。

・ プールアロケータは、必要に応じてプールの増減を自動的に行うものとする。

・ メモリマネージャは、「スラブアロケータ」に対応し、任意のクラス／構造体の配列を扱えるものとする。

・ スラブアロケータは、必要に応じてキャッシュの増減を自動的に行うものとする。

・ メモリマネージャは、ヒープメモリを任意の「ゾーン」に区切って、複数の物理的なメモリ区画を扱えるものとする。
  - プール、スラブはゾーンと無関係。

・ ゾーンのメモリ境界は状況に応じて自動的に変動し、メモリ限界の警告を出しつつも、ゲームを止めないようにする。

・ メモリマネージャは、メモリの「カテゴリ」を扱い、カテゴリごとの集計とメモリ制限を行うものとする。
  - メモリ確保時にカテゴリを指定する。
  - ヒープとプールはカテゴリの制限を受ける。
  - メモリ制限はあらかじめカテゴリごとに設定しておき、ゲーム中に変更することが可能。
  - スラブはカテゴリを扱わない。

・ メモリマネージャは、スマートポインタをメモリマネージャレベルでサポートするものとする。
  - 参照カウンタを標準的に管理する。
  - スラブはスマートポインタに対応しない。

・ メモリマネージャは、メモリリークの検出をサポートする機能を備えるものとする。
  - メモリリークの原因追及のために、デバッグ情報として、メモリ確保時の情報を少しでも多く記録する。

・ 【できれば】メモリマネージャは、仮想アドレスによる動的なメモリ増強に対応するものとする。

・ 【できれば】メモリマネージャは、グラフィックメモリのような、直接アクセスできないメモリの配分の管理も可能なものとする。

▼ 要求仕様／要件定義

以下、本書が扱うシステムの要件を定義する。なお、要件として不確定の要求仕様も併記する。

・ メモリマネージャは、64bit システムであっても、内部は 32bit で扱う。
  - 先頭領域からのオブセットでポインタを管理する。

・ メモリマネージャは、一つのゲームシステムに複数のインスタンスを持つことができる。
  - 一つのメモリマネージャのインスタンスが管理できるメモリ領域のサイズは、一つにつき最大 16GB とする。
  - メモリ情報は 32bit で管理するが、ポインタは最低 8 バイトアラインメントされるため、3bit 分はシフトすることが可能。1bit 他の用途に利用する可能性を考慮し、2bit シフトでポインタを扱うものとすると、16GB のメモリ空間を表現できる。
  - メモリ領域の全体サイズも同様に 2bit シフトで扱うが、それ以外のメモリサイズは全てシフトなしで扱う。
    - そのため、個々のメモリサイズの上限は 4GB となる。
    - 例えば、メモリ確保要求の上限は 4GB であり、ヒープメモリの領域は（一ゾーンあたり）4GB が上限となる。

・ メモリマネージャは、「ヒープアロケータ」、「プールアロケータ」、「スラブアロケータ」の混成システムとする。

・ メモリ領域は、「ヒープメモリ領域」と「ページ領域」で構成する。

・ ヒープメモリ領域は、任意の「ゾーン」を設定し、物理的に区分けされた複数の領域に分けて扱う。

・ ページ領域は、「バディシステム」によって管理し、多目的な用途に使用する。
  - ページ領域は１ページ 4KB で構成し、各サブシステムからの要求に応じた割り当てと返却を行う。

・ ページ領域では、「ヒープメモリの管理情報」、「プールメモリ」とその管理情報、「スラブキャッシュ」とその管理情報を扱う。

・ プールメモリは 256 バイト以下の小さいメモリを管理し、メモリマネージャに対する

メモリ確保要求がこの範囲内であれば、プールメモリから確保する。それより大きい場合はヒープメモリから確保する。

・ プールメモリは、必要に応じてページ領域のメモリを割り当て、自動拡張する。

・ ページ領域およびプールメモリにはサイズの限界が設定され、プールメモリの追加ができない場合は、小さいメモリでもヒープからメモリを確保する。
  - さらにヒープからも確保できなかった場合、要求サイズより大きいプールからの確保を試行する。例えば、16 バイトの要求に対して、16 バイトプールからの確保ができなかった場合、ヒープからの確保を試し、それがダメなら 24 バイトのプールから、32 バイトのプールから…と、順次試していき、最後に 256 バイトのプールからも確保できなければ、メモリ確保失敗となる。

・ ヒープメモリとプールメモリは表向き区別されず、どちらからメモリが確保されたとしても、「カテゴリによるメモリ制限」、「スマートポインタ対応」が可能。

・ メモリ確保時は、「カテゴリ」を指定する。
  - メモリマネージャに対して、任意のカテゴリを複数設定しておくができる。
  - カテゴリを設定する際、カテゴリごとのメモリ制限も設定する。
  - メモリ制限を超えるメモリ確保要求は、たとえメモリに空きがあっても、失敗する。
  - メモリ制限は、「超えたら警告」のサイズと、「超えたら確保失敗」のサイズの２段階で設定する。

・ メモリ確保したヒープおよびプールは、ページ領域に管理情報を持ち、参照カウンタを扱うことができる。
  - メモリマネージャ専用のスマートポインタテンプレートクラスがあり、それを使うと、管理情報の参照カウンタを扱う。

・ ヒープメモリはメモリ再配置に対応する。
  - ただし、確保したメモリはデフォルトでは再配置可能にならない。
  - まず、ゾーンに対して「再配置可能」属性が設定されている必要がある。
  - 続いて、確保したメモリをスマートポインタで扱うと、そのメモリが「再配置可能」なメモリになる。
  - スマートポインタを通してメモリにアクセスすると、メモリが再配置されても気にせずアクセスし続けることができる。（スマートポインタの内部では、直接データのポインタを持たず、管理情報のポインタを扱っているため）

・ スラブアロケータは、ヒープ／プールと異なり、専用のメソッドでメモリの確保／解放を行う。
  - スラブアロケータでは、任意のクラス／構造体の配列を扱う。

- 専用のメソッドを通してスラブキャッシュの作成（配列の予約）、スラブオブジェクトの確保／解放、スラブキャッシュの破棄を行う。
- スラブキャッシュは自動拡張に対応する。
  - 拡張の際に、幾つずつオブジェクトを拡張するかは、スラブキャッシュ作成時に指定しておく。
  - 必要に応じて随時追加でページを割り当てる。

・ メモリ確保したスラブは、ヒープ／プールと同様にページ領域に管理情報を持つ。

・ メモリ確保したヒープ／プール／スラブの管理情報はデバッグ情報を保持する。
- メモリ確保を要求した処理のファイル名や関数名などを扱う。

・ ヒープメモリのゾーン、および、ページ領域は、メモリ不足になると、境界の移動によってメモリを増やすことを試みる。
- 隣接するゾーンの最後方（もしくは先頭）に空きがある場合だけ、境界を動かすことができる。
- これは標準的な機能というよりも、メモリが予定外に不足した場合の救済措置である。
- メモリ不足を画面表示などで通知しながらも、ゲームを止めないようにするための措置である。
  - ゲームを止めない方が問題発生時の状況を確認しやすい。その時のプレイヤーはゲームをポーズ状態にしてプログラマーを呼ぶなどする。

・ 【可能なら】この救済措置を仮想アドレスによって実現する。
- あらかじめ物理メモリに余裕をもたせておいた状態で、各ゾーンおよびページ領域は、隙間を空けたアドレスを設定してメモリを割り当てる。追加が必要になったゾーンには新たに物理メモリを割り当てて拡張する。
- この処理を行う場合、「メモリリージョン」（仮想アドレスと物理メモリのマッピング）を管理する必要がある。

・ 【可能なら】メモリマネージャが直接操作できないメモリの管理にも対応する。
- グラフィックメモリのように、直接メモリにアクセスできないようなメモリの配分・管理に対応する。
- ヒープメモリ領域を実際に用意せず、管理情報のみを扱う動作モードを用意する。
- このモードでは、プールアロケータやスラブアロケータも機能しない。

・ 【可能なら】デバッグ用に、メモリリークを検出する機能を備える。
- マークスイープ GC（ガベージコレクション）の仕組みを用意し、任意のタイミングでその処理を有効化／無効化する。

- 有効化すると、「マークフェーズ」の処理を実行し、静的変数領域とメモリマネージャが管理するメモリ領域全体を走査し、「メモリマネージャ内のポインタ」と推測できる値を拾い、使用中メモリとして管理情報にマークを付ける。
  - 非常に重い処理である。
- マークフェーズが終わったら、「スイープフェーズ」の処理を実行し、「マークが付いていないが確保状態」のメモリをピックアップし、メモリリークとみなし、ログ出力する。
  - 通常のガベージコレクションのように、自動解放はしない。
- メモリリーク情報の出力は、ソフトリセット時にも行う。
  - ソフトリセット時、メモリマネージャの破棄（デストラクタ）のタイミングまで明示的に破棄されずに残ったメモリをメモリリークとみなし、ログ出力する。

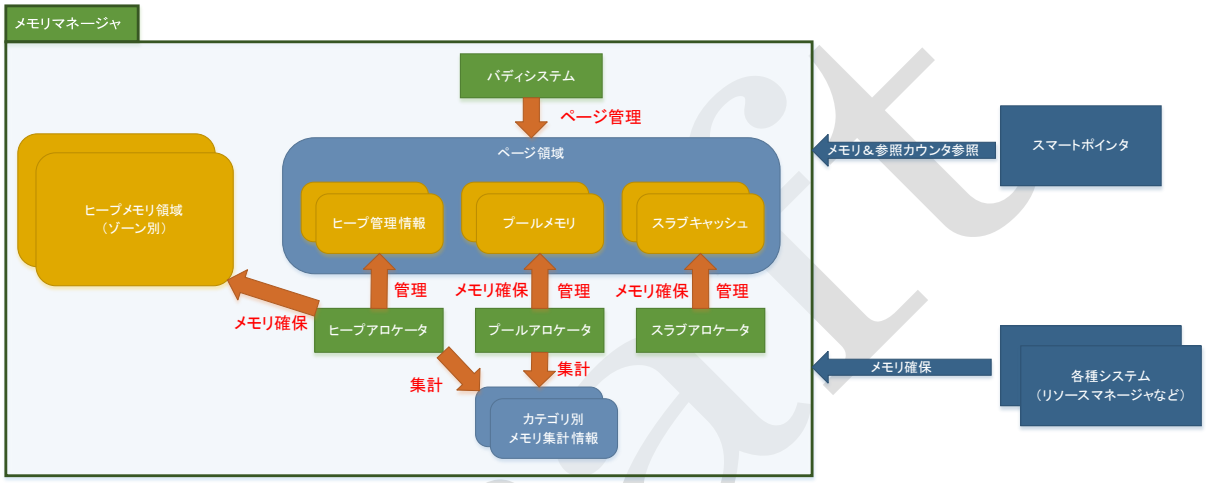
・ 別紙の「開発の効率化と安全性のためのリソース管理」に示すとおり、メモリの許す限りリソースを破棄せず、再利用に備えて保持し続け、メモリ不足になった時に、未使用リソースを破棄する。
この処理が本当にメモリ不足になってからの実行だと、メモリマネージャ側もメモリ追加処理を行おうとするため、すぐにメモリが異常な状態になってしまう。
この問題を考慮し、80%ぐらいのメモリ消費状態でリソースマネージャにコールバック通知し、リソース削除を促すようにする。

■ 仕様概要

▼ システム構成図

要件に基づくシステム構成図を示す。

メモリマネージャのシステム構成図：



■ 処理／データ仕様

別紙の「ゲーム制御のためのメモリ管理方針」にアルゴリズムやデータ構造を詳述しているため、省略。

■■以上■■

■ 索引

索引項目が見つかりません。

柔軟性を追求したメモリ管理システム

以　上