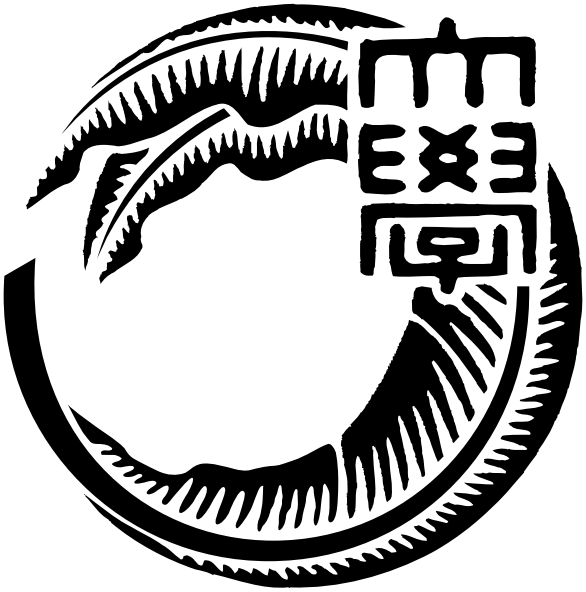
平成28年度 卒業論文

ゲームエンジンにおける木構造データベースJungleの提案



琉球大学工学部情報工学科

135768K 武田 和馬

指導教員 河野 真治

目 次

図 目 次

表 目 次

第1章　研究目的

プログラムからデータを分離して扱うデータベースには、プログラム中のデータ構造とRDBの表構造のズレによりインピーダンスミスマッチという問題がある。

データベースのレコードをプログラム中のオブジェクトとして扱えるORMapperやデータベース自体も、表に特化したKey Value Storeや、Json、XMLの不定形のデータ構造を格納するように機能拡張されている。

ORMapperではデータベースのレコードをプログラム中のオブジェクトとして扱うことができる。オブジェクトに対する操作を行うとORMapperがSQLを発行し、処理を行ってくれる。

しかしレコードをプログラム中のオブジェクトを対応させるORMapperの技術でインピーダンスミスマッチを解決することはできない。

JsonやXMLを扱えるデータベースでは通常スキームを必要としないため、特に設計を行わずデータを格納することができる。

しかし、不定形の構造の変更をトランザクションとして、Jsonの一括変更という形で処理されてしまっており、並列アプリケーションには向いていない。

当研究室ではデータの変更の際に過去の木構造を保存する非破壊木構造データベースであるJungleを提案している[?]。

本研究ではJungleをUnityを用いたゲームで使用する方法を提案する。データベースとしてJungle DBをC#で再実装を行い、Unity向けに組み込みを行う。

第2章　Unityでのデータベースの取扱い

2.1　UnityとJungleの関係

Unityは3Dゲームエンジンで、ゲームを構成する要素(Object)をC#で制御する。Objectは一つのゲームのシーン(一画面の状況)の中で木構造を持つ。これをシーングラフと言う。シーングラフをそのままJungleに格納するという手法が考えられる。

2.2　Unityにおけるデータベース

Unityでのデータベースとして考えられるものとしてはMySQL、SQlite3、PlayerPrefsが挙げられる。

PlayerPrefsとは、Unityに特化したバイナリ形式でKeyとValueのみで保存されるものである。セーブ機能に特化していてメモリ上にDBを展開するものではない。

第3章　Jungle-Sharpの実装

Java と C#はよく似た言語であり、移行はそれほど難しくない。Jungle の中心部分である木構造と Index を構成する赤黒木のコードはほぼ変更なく移行できた。C#ではインナークラスが使えないので明示的なクラスに変換する必要があった。

3.1　AtomicRefarence の実装

Jungle の木の変更 (commit) は CAS(Check and Set) を用いて atomic に行われる。競合している書き込み中に自分の書き込みが成功した場合に関数 `success()` が成功する。

Java では AtomicRefarence が標準であるが C#にはなかったため AtomicRefarence の Class を新たにつくった。

AtomicReplace

```
// C#
public bool CompareAndSet(T newValue, T prevValue) {
    T oldValue = value;
    return (oldValue
        != Interlocked.CompareExchange
                  (ref value, newValue, prevValue));
}

// Java
AtomicRefarence<T> atomic = new AtomicRefarence<T>();
atomic.compareAndSet(prevValue, newValue);
```

3.2　List の実装

木やリストをたどる時に Java では Iterator を用いる。Iterator は次の値があるかを返す boolean hasNext() と、T という型の次の値を取ってくる T next() を持つ Object である。C#では木やリストを辿りながら yeild で次の値を返す。Java では以下のように実装されている。

List.java

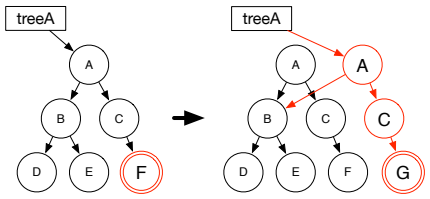
```
public Iterator<T> iterator() {
  return new Iterator<T>() {
    Node<T> currentNode = head.getNext();

    @Override
    public boolean hasNext() {
      return currentNode.getAttribute()
                                    != null;
    }

    @Override
    public T next() {
      T attribute
            = currentNode.getAttribute();
      currentNode
            = currentNode.getNext();
      return attribute;
    }
  };
}
```

C#ではIEnumeratorがあるのでそれを利用した。Listのforeachではを呼び出すために、一つずつ要素を返す必要がある。yield returnステートメントを利用することで位置が保持され、次に呼ばれた際に続きから値の取り出しが可能になる。以下にその実装例を示す。

List.cs

```
public IEnumerator<T> iterator() {
  Node<T> currentNode = head.getNext();
  while (currentNode.getAttribute() != null) {
    yield return (T)currentNode.getAttribute();
    currentNode = currentNode.getNext ();
  }
}
```

第4章　Jungle Database の概念



Jungle は名前付きの複数 の集合で出来ている。 ノードは自身の子のリス これはデータベースの レコードに相当する。通常 もとなる複数のノード がつくところである。

Jungle は、データの変 トから編集を行うノー ドまでのコピーを行い、新 ミックに入れ替える図 4.1。これを非破壊的木構 築している時にも、現 在の木を安全に読み出せ き込みの手間は大きく なる。

図 4.1: 非破壊的木構造の木の編集

4.1　Jungle Database の構造

非破壊木構造を採用している Jungle では、木の変更の手間は O(1) から O(n) となり得る。つまりアプリケーションに合わせて木を設計しない限り充分な性能を出すことは出来ない。逆に木の設計を行えば高速な処理が可能である。

Jungle はオンメモリで使用することを考えており、一度木のルートを取得すれば、その上で木構造として自由にアクセスしてもよい。

Jungle は commit log を持ち、それを他のノードやディスクに転送することにより、分散構成と持続性を実現する。

4.2 JungleDatabase の API

4.2.1 Jungle の木

Jungle は複数の木の名前を利用し、管理しており、名前により生成、編集を行う。以下に Jungle クラスが提供している木の生成、管理を行う API(表 4.1) に記述する。

表 4.1: Jungle に実装されている API

`JungleTree createNewTree(string treeName)`	Jungle に新しく木を生成する。木の名前が重複した場合、生成に失敗し null を返す。
`JungleTree getTreeByName(string treeName)`	Jungle から treeName と名前が一致する tree を取得する。名前が一致する Tree がない場合取得は失敗し null を返す

4.2.2 TreeNode

Jungle が保有する木は、複数のノードの集合で出来ている。ノードは、自身の子の List、属性名と属性値の組のデータを持つ。ノードに対するアクセスは表 4.2 に記述されている API を用いて行う。

表 4.2: TreeNode に実装されている API

`Children getChildren()`	ノードの子供を扱う Children オブジェクトを返す。
`Attribute getAttribute()`	ノードが保持しているデータを扱う Attribte オブジェクトを返す。

4.2.3 Either

jungle では例外処理を投げる時に Either クラスを用いて行う。返って来た Either のオブジェクトに対して、`isA()` で `Error` かどうかをチェックする。`Error` でない場合は `b()` で対象のオブジェクトを取り出す事ができる。

以下にルートノードの 2 番目の子どもを取ってくるの Either のサンプルコードを記述する。

```
Either<Error,TreeNode> either = children.at(2);
if (either.isA())
    return either.a();
TreeNode child = either.b();
```

4.2.4 Children と Attribute

Children クラスへのアクセスは表 4.3 に記述されている API を、Attribute クラスへアクセスは表 4.4 に記述されている API を用いて行う。

表 4.3: Children に実装されている API

`int size()`	子供の数を返す。
`<Either Error,TreeNode> at(int num)`	ノードが持つ子供の中から、 変数 `num` で指定された位置にある子ノードを返す。

表 4.4: Attribute に実装されている API

`byte[] get(string key)`	ノードが持つ値から、属性名 `key` とペアの属性値を `byte array` 型で返す。
`string getString(string key)`	ノードが持つ値から、属性名 `key` とペアの属性値を `string` 型で返す。

4.2.5 NodePath

Jungle では、木のノードの位置を `NodePath` クラスを使って表す。`NodePath` クラスはルートノードからスタートし、対象のノードまでの経路を、数字を用いて指し示すことで対象のノードの場所を表す。また、ルートノードは例外として-1 と表記される。`NodePath` クラスが`< -1,1,2,3>` を表している際の例を図 4.2 に記す。

4.2.6 木の編集

Jungle の木の編集は `JungleTreeEditor` クラスを用いて行われる。`JungleTreeEditor` クラスには編集を行うために、表 4.5 に記述されている API を用いて行う。

編集を行った後は、関数 `editor.commit()` で今までの編集をコミットすることができる。他の `JungleTreeEditor` クラスによって木が更新されていた場合はコミットは失敗し、`commit()` は `Error` を返す。その場合は、木の編集を最初からやり直す必要がある。

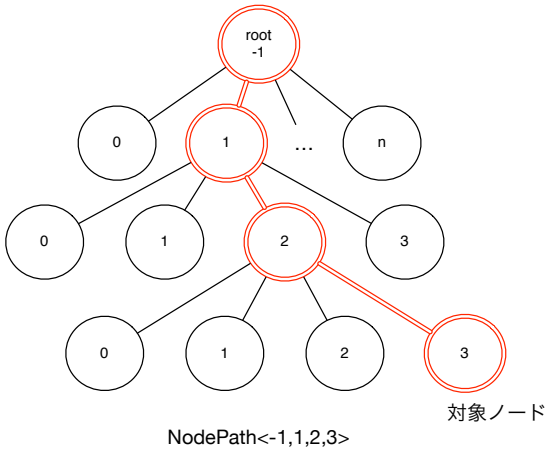


図 4.2: NodePath

表 4.5: Editor に実装されている API

Either<Error, JungleTreeEditor> addNewChildAt(NodePath path, int pos)	変数 path で指定した場所にある、ノードの子供の変数 pos で指定した位置子ノードを追加する
Either<Error, JungleTreeEditor> deleteChildAt(NodePath path,int pos)	変数 path で指定した場所にある、ノードの子供の変数 pos で指定した位置の子ノードを削除する。
Either<Error, JungleTreeEditor> putAttribute(NodePath path,String key,ByteBuffer value)	変数 path で指定した場所にあるノードに、属性名 変数 key 属性値 変数 value のペアで値を挿入する。
Either< Error, JungleTreeEditor> deleteAttribute(NodePath path,String key)	変数 path で指定した場所にあるノードが持つ、属性名 変数 key とペアで保存されているデータを削除する。
Either<Error, JungleTreeEditor> commit()	木へ行った変更をコミットする。自分が編集を行っていた間に、他の JungleTreeEditor クラスによって木が更新されていた場合、コミットは失敗する。

第5章　Unityで実装したアプリケーション

本論文ではC#で再実装を行ったJungleをUnityで作られたゲームの上に構築する。例題のゲームとしては図5.1のマインクラフトの簡易版を作成する。

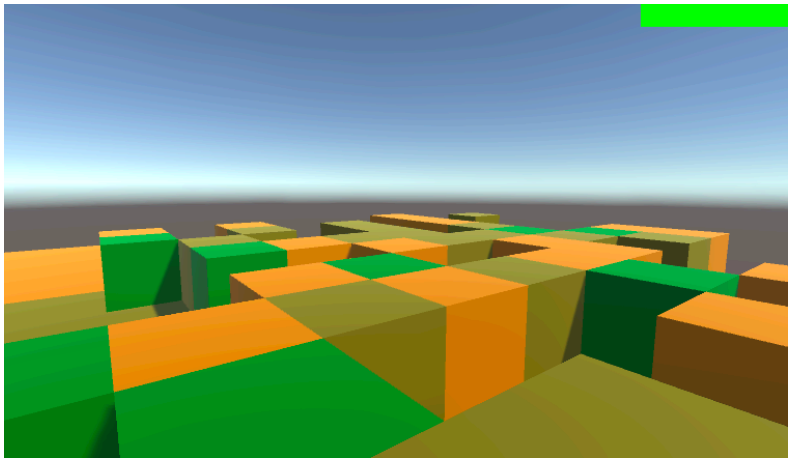


図 5.1: craft

5.1　データ設計

第6章　ベンチマーク

6.1　Javaとの比較

6.2　他のデータベースとの比較

第7章　結論

7.1　まとめ

7.2　今後の課題

　ネットワークゲームの課題はデータの偽装、つまりチートである。今後ネットワークとして対応させるにはそこを考える必要がある。

参考文献

[1] RICHARDSON, T., AND LEVINE, J. The remote framebuffer protocol. rfc 6143, mar 2011.

[2] TightVNC Software. `http://www.tightvnc.com`.

[3] RICHARDSON, T., STAFFORD-FRASER, Q., WOOD, K. R., AND HOPPER,. A. virtual network computing, jan 1998.

[4] LOUP GAILLY, J., AND ADLER, M. zlib: A massively spiffy yet delicately unobtrusive compression library. `http://zlib.net`.

[5] Surendar Chandra, Jacob T. Biehl, John Boreczky, Scott Carter, Lawrence A. Rowe. Understanding screen contents for building a high performance, real time screen sharing system. *ACM Multimedia*, Oct 2012.

[6] Yu TANINARI and Nobuyasu OSHIRO and Shinji KONO. Vnc を用いた授業用画面共有システムの実装と設計. 日本ソフトウェア科学会第 28 回大会論文集, sep 2011.

[7] Yu TANINARI and Nobuyasu OSHIRO and Shinji KONO. Vnc を用いた授業用画面共有システムの設計・開発. 情報処理学会システムソフトウェアとオペレーティング・システム研究会 (OS), may 2012.

[8] Tatsuki IHA and Shinji KONO. 有線 lan 上の pc 画面配信システム treevnc の改良. 第 57 回 プログラミング・シンポジウム, jan 2016.

[9] Nozomi TERUYA and Shinji KONO. 分散フレームワーク alice の pc 画面配信システムへの応用. 第 57 回 プログラミング・シンポジウム, jan 2016.

謝辞

本研究の遂行，また本論文の作成にあたり、御多忙にも関わらず終始懇切なる御指導と御教授を賜わりました河野真治准教授に深く感謝致します。

数々の貴重な御助言と細かな御配慮を戴いた金川 竜己さん、並びに並列信頼研究室の皆様に深く感謝致します。

最後に、有意義な時間を共に過ごした情報工学科の学友、並びに物心両面で支えてくれた両親に深く感謝致します。

2017 年 3 月

武田和馬