# Unity における JungleDB の有用性

135768K 武田和馬　指導教員：河野真治

## 1　非破壊木構造データベース

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 を開発している [1]。

本研究では Jungle を Unity を用いたネットワーク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。

Jugnle の木は、子供を複数持つノードからなる。子供は順序付けられており、任意の位置で作成削除することができる。ノードは属性名と属性値の組からなる表を持つ。

## 2　Unity での DB の取り扱い

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。

Jungle は Java で書かれていたので、Unity で使うには Jungle を C#で実装する必要がある。クライアント側は C#、サーバー側は Java で動作する Jungle を用いる。Jungle の分散機構を用いてネットワークゲームに必要な通信を行う。通信は MessagePack を基本に実装されている。従って、C#側でも MessagePack を用いてデータのやり取りを行いたい。

図 2 のようにクライアント側では 2 つの Tree を持っている。GameTree から CommunityTree にコピーする。サーバー側とは CommunityTree で同期する。

図 2 のようにサーバー側では Player ごとに Tree を持っている。

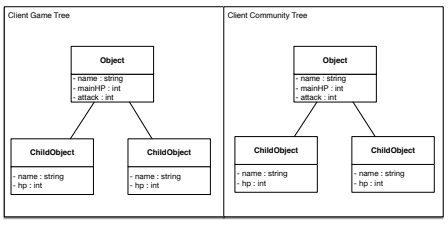


図 1: クライアント側の木構造

Unity ではデータの保存の際に SQlite3、PlayerPrefs といった DB がよく使われている。PlayerPrefs とは、Unity に特化したバイナリ形式で Key,Value のみで保存されるものである。これらの DB には Object を直接格納することはできない。また、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。

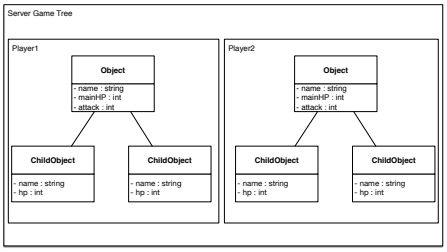


図 2: サーバー側の木構造

## 3　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はない。実際 Jungle の中心部分である木構造と Index を構成する赤黒木 [2] 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。

異なる部分は一つは木を変更した後、木のルートを Atomic に置き部分である。もう一つは木をたどる時に使う Itertor である。

Jungle の木の更新 (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にの中で自分の書き込みが成功した場合に関数 `success()` が成功する。Java には AtomicRefarence が標準であるが C#はなかったため、AtomicReference の Class を新たに作った。

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);
```

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#では木やリストをたどりながら yeild で次の値を返す。

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#には IEnumerator があるのでそれを利用した。List の foreach では Iterator を呼び出すため、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。

List.cs

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

## 4 ベンチマーク

Unity では Sqlite3,PlayerPrefs がデータの保存として利用される。今回の検証は Insert を 1000 回行い、push(または Save) を行うまでの時間を測定する。

使用した機材は以下の通りである。

- OS : Windows 10
- CPU : Intel Core i7-4700MQ 2.4GHz
- Unity : 5.4.2f1

表 1: 速度測定

| データベース | 速度 (ms) |
|---|---|
| Jungle | 13 |
| Sqlite3 | 15062 |
| PlayerPrefs | 36 |

Jungle が早い理由としてデータを書き出さずメモリ上にデータを持っているからである。これはゲームを終了するとデータが消される。Sqlite3 が遅い理由としてはデータを Insert する毎に DB を書き込みを行っているため遅い。PlayerPrefs はデータを書き出すが、データをセットしたのち一度だけまとめて書き出すため早い。

## 5 これからの作業

Jungle は分散型のデータベースを目指しているため、MessagePack の実装を行う。今後はサーバーとの連携も行う。Jungle は RDB と異なりデータを自由に格納することができる。そこでデータベース設計を確立させる必要がある。

## 参考文献


[1] 金川竜己 非破壊的木構造データベース Jungle とその評価

[2] これでわかった赤黒木

http://www.moon.sannet.ne.jp/okahisa/rb-tree/