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しつつ新しく木構造を作成する非破壊的木構造を用いたデータベースである Jungle を開発している [1]。

本研究では Jungle の実用例として、ゲームのバックエンドとして利用出来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。

Unity は 3 D ゲームエンジンである。ゲーム構造はシーングラフに類似している。ゲームシーンに子ノードのゲーム要素 (GameObject) があり、その要素自身も親子関係になっている。Jungle はこれと同じ構造を持っているため、相性が良い。

2　Unity での問題点

Unity ではデータの保存の際に MySQL、SQlite3、PlayerPrefs といった DB がよく使われている。しかし、木構造でゲームは構成されているため、ゲーム構造を保存するには RDB 向けにノードの関係を変換する必要がある。つまり、そのまま格納することができればスケールアウトするデータにも対応でき、データベース設計も簡略化できると考え、C#に書き直すことにした。

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。

3　Jungle-Sharp の実装

Jungle は Java で書かれているものであったので Unity で使うには C#で実装する必要があった。なお、将来的にはクライアント側は C#、サーバー側は Java で動作させ MessagePack を用いてデータのやり取りを行う。

4　AtomicRefefarence の実装

Jungle の木の更新 (commit) は、CAS(check and set*図 1) 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。

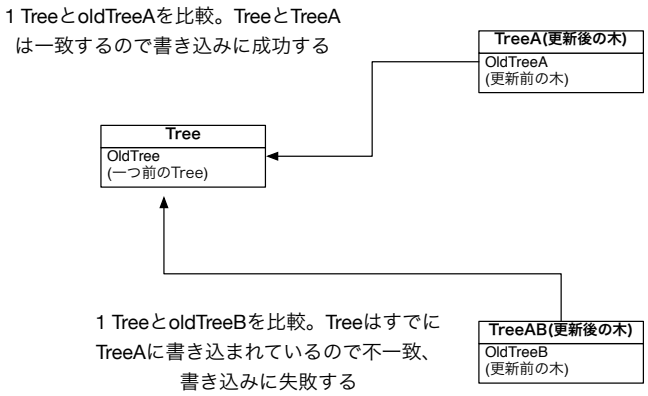

図 1: Check and Set

ソースコード 1 AtomicReference.cs

```
public class AtomicReference <T> where T : class {
  private T value;

  public AtomicReference(T value) {
    this.value = value;
  }

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

  public T Get() {
    return value;
  }
}
```

CompereAndSet メソッドでは Interlocked Operation を利用した。これによりスレッドセーフな値の変更を行うことが可能になる。

5 List の実装

Jungle では、木の編集や、特定の Node 下の Tree の探索、Node の親をたどるためには全てその Node への Path が必要になる。その管理を List で行っている。List を実装する際に Iterator が必要となる。Java ではインナークラスで Iterator を返すが

ソースコード 2 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。

ソースコード 3 List.cs

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

6 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 : Unity 5.4.2f1

表 1: 速度測定

データベース	速度 (ms)
Jungle	100
Sqlite3	100
PlayerPrefs	100

図 3 の結果より JungleDB の速度を確認することができた。

7 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