継続を使用する並列分散フレームワークのUnity実装

安田 亮[1,a)]　河野 真治[2,b)]

概要：FPS や MMORPG などのゲームにおける通信方式には、クライアントサーバ方式と p2p 方式の 2 つが考えられる。しかし、クライアントの負荷軽減やチート対策などを理由にクライアントサーバ方式が主流である。データの同期にはサーバを経由するため低速である。そこで本研究室で開発している分散フレームワーク Christie を用いることで、高速かつ、安全に、データの同期を行いたいと考えた。本研究では Chrisite をゲームエンジン Unity に対応するため、C#への書き換えを行う。

1. オンラインゲームにおけるデータ通信

2. Christie の基礎概念

Chrisite は当研究室で開発している分散通信フレームワークである。同じく当研究室で開発している GearsOS のファイルシステムに組み込まれる予定があるため、GearsOS を構成する言語 Continuation based C と似た概念を持っている。 Chrisite に存在する概念として以下のようなものがある。

- CodeGear
- DataGear
- CodeGearManager
- DataGearManager

以下は java 版の Chrisite について解説を行う。CodeGear はクラスやスレッドに相当する。DataGear は変数データに相当し、CodeGear 内で annotation を用いて変数データを取得する。CodeGear 内に記述した全ての DataGear の中にデータが格納された際に、初めてその CodeGear が実行されるという仕組みになっている。CodeGearManager はノードであり、CodeGear、DataGear、DataGearManager を管理する。DataGearManager は DataGear を管理するものであり、put という操作により変数データ、つまり DataGear を格納できる。DataGearManager の put 操作を行う際には Local と Remote のどちらかを選び、変数の key とデータを引数として渡す。Local であれば、Local の CodeGearManager が管理している DataGearManager に対し DataGear を格納していく。Remote であれば、接続した Remote 先の CodeGearManager が管理している DataGearManager に DataGear を格納できる。put 操作を行った後は、対象の DataGearManager の中に queue として保管される。DataGear を取り出す際には、CodeGearm 内で宣言した変数データに annotation をつける。DataGear の annotation には Take、Peek、TakeFrom、PeekFrom の 4 つがある。

Take　先頭の DataGear を読み込み、その DataGear を削除する。DataGear が複数ある場合、この動作を用いる

Peek　先頭の DataGear を読み込むが、DataGear が削除されない。そのため、特に操作をしない場合は同じデータを参照し続ける。

TakeFrom (Remote DGM name)　Take と似ているが、Remote DGM name を指定することで、その接続先 (Remote) の DataGearManager から Take 操作を行える。

PeekFrom (Remote DGM name)　Peek と似ているが、Remote DGM name を指定することで、その接続先 (Remote) の DataGearManager から Peek 操作を行える。

3. プログラムの例

Code 1 は Chrisite の機能を使用して hello world を出力する例題である。CodeGearManager を作り、setup(new CodeGear) を行うことで各 CodeGear に記述された DataGear の待ち合わせを行う。全ての DataGear が揃った場合に CodeGear が実行される。CodeGearaManager の作成方法は StartCodeGear を継承したものから、createCGM(port) を実行することにより、CodeGearManager が作成できる。

[1] 琉球大学大学院理工学研究科情報工学専攻
[2] 琉球大学工学部工学科知能情報コース
[a)] riono210@cr.ie.u-ryukyu.ac.jp
[b)] kono@ie.u-ryukyu.ac.jp

Code 1: StartHelloWorld

```
1  package christie.example.HelloWorld;
2
3  import christie.codegear.CodeGearManager;
4  import christie.codegear.StartCodeGear;
5
6  public class StartHelloWorld extends StartCodeGear {
7
8      public StartHelloWorld(CodeGearManager cgm) {
9          super(cgm);
10     }
11
12     public static void main(String[] args){
13         CodeGearManager cgm = createCGM(10000);
14         cgm.setup(new HelloWorldCodeGear());
15         cgm.setup(new FinishHelloWorld());
16         cgm.getLocalDGM().put("helloWorld","hello");
17         cgm.getLocalDGM().put("helloWorld","world");
18     }
19 }
```

4. Unity

5. annotation の書き換え

java 版では DataGear を取得する際に、annotation という java の機能を用いて行った。C#には annotation はなく、代わりに attribute を利用して DataGear の取得を行っている。以下の Code 2、Code 3 は java と C# における Take の実装である。

Code 2: java における Take annotation

```
1  package christie.annotation;
2
3  import java.lang.annotation.ElementType;
4  import java.lang.annotation.Retention;
5  import java.lang.annotation.RetentionPolicy;
6  import java.lang.annotation.Target;
7
8  @Target(ElementType.FIELD)
9  @Retention(RetentionPolicy.RUNTIME)
10 public @interface Take {
11 }
```

Code 3: C# における Take attribute

```
1  using System;
2
3  namespace Christie_net.annotation {
4  [AttributeUsage(AttributeTargets.Field)]
5  public class Take : Attribute { }
6  }
```

java で annotation を自作する際には、 @interfacs で宣言する。また、Code 2 の 8 行目では annotation 情報をどの段階まで保持するかを指定しており、Take の場合 JVM によって保存され、ランタイム環境で使用できる。9 行目では annotation の適用可能箇所を指定しており、フィールド変数に対して適応可能となっている。

C# で attribute を作成する際には、 System.Attribute を継承する必要がある。attribute の適用可能箇所については、Code 3 の 4 行目でフィールド変数を指定している。

6. MessagePack の相違点

Christie ではデータを送信する際に、MessagePack を使用してデータを圧縮し、送信している。java 版で使用している MessagePack はバージョンが古く現在はサポートされてない。そのため MessagePack の最新版とは記述方法が異なっている。Code **??** は MessagePack の使用方法を示したものである。

Code 4: java 版における MessagePack の使用方法

```
1  import org.msgpack.MessagePack;
2  import org.msgpack.annotation.Message;
3
4  public class Main1 {
5      @Message // Annotation
6      public static class MyMessage {
7          // public fields are serialized.
8          public String name;
9          public double version;
10     }
11
12     public static void main(String[] args) throws
           Exception {
13         MyMessage src = new MyMessage();
14         src.name = "msgpack";
15         src.version = 0.6;
16
17         MessagePack msgpack = new MessagePack();
18         // Serialize
19         byte[] bytes = msgpack.write(src);
20         // Deserialize
21         MyMessage dst = msgpack.read(bytes, MyMessage.
               class);
22     }
23 }
```

MessagePack を使用するには圧縮するクラスに対して @Message annotation をつける必要がある。これにより、クラス内で定義した public 変数が圧縮される。Code 4 の 17 - 21 行目は圧縮解凍の例であり、MessagePack のインスタンスを作成後、msgpack.write(data) を行うことで byte[] 型に data を圧縮できる。解凍には msgpack.read を使用し、圧縮された byte[] 型と圧縮対象のクラスを渡すことで解凍できる。

C# の MessagePack は複数存在しており、java と同様な書き方をする neuecc の MessagePack-CSharp を選択した。

Code 5: C# 版における MessagePack の使用方法

```
1  [MessagePackObject]
2  public class MyClass {
3      [Key(0)]
4      public int Age { get; set; }
```

```
5     [Key(1)]
6     public string FirstName { get; set; }
7     [Key(2)]
8     public string LastName { get; set; }
9
10    static void Main(string[] args) {
11        var mc = new MyClass {
12            Age = 99,
13            FirstName = "hoge",
14            LastName = "huga",
15        };
16
17        byte[] bytes = MessagePackSerializer.Serialize
              (mc);
18        MyClass mc2 = MessagePackSerializer.
              Deserialize<MyClass>(bytes);
19
20        // [99,"hoge","huga"]
21        var json = MessagePackSerializer.ConvertToJson
              (bytes);
22        Console.WriteLine(json);
23    }
24 }
```

7. C#への書き換え後のプログラムの例

Code 6: C# StartHelloWorld

```
1  using Christie_net.codegear;
2
3  namespace Christie_net.Test.Example.HelloWorld {
4  public class StartHelloWorld : StartCodeGear {
5
6      public StartHelloWorld(CodeGearManager cgm) : base
           (cgm) { }
7
8      public static void Main(string[] args) {
9          CodeGearManager cgm = CreateCgm(10000);
10         cgm.Setup(new HelloWorldCodeGear());
11         cgm.Setup(new FinishHelloWorld());
12         cgm.GetLocalDGM().Put("helloWorld", "hello");
13         cgm.GetLocalDGM().Put("helloWorld", "world");
14
15     }
16 }
17 }
```

8. Unity での動作

9. チート対策について

10. 実装の現状

参考文献

[1] RICHARDSON, T., AND LEVINE, J.: The remote framebuffer protocol. RFC 6143 (2011).
[2] TightVNC Software: http://www.tightvnc.com.
[3] RICHARDSON, T., STAFFORD-FRASER, Q., WOOD, K. R., AND HOPPER,: A. Virtual Network Computing (1998).
[4] LOUP GAILLY, J., AND ADLER, M.: zlib: A massively spiffy yet delicately unobtrusive compression library., http://zlib.net.
[5] 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 (2011).
[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) (2012).