CrestMuse Toolkit レシピ

1 CrestMuse Toolkit を Processing から利用する

- CrestMuse Toolkit （CMX) は Java のライブラリなので，Processing から利用することができます.
- ただ，CMX には大量のクラスがあり，どのクラスをどう使えばいいかを調べるのは大変です．そこで，CMX の主要な機能は，CMXController というクラスから呼び出せるようになっています.

```
import jp.crestmuse.cmx.processing.*;

void setup() {
  size(320, 240);
  noLoop();
  CMXController cmx = CMXController.getInstance();
  cmx.mp3read(“ sample1.mp3 ”);
  cmx.playMusic();
}

void draw() {
   ellipse(width/2, height/2, width-40, height-40)
}
```

2 CrestMuse Toolkit を Groovy から利用する

- CrestMuse Toolkit には，CMXApplet というクラスが用意されています．このクラスを継承したクラスを作成することで，アプリケーションを簡単に作ることができます．

- CMXApplet クラスには，CMXController クラスから呼び出せるほぼすべての機能を呼び出すことができるようになっています．CMXApplet クラスのサブクラスからは，まるで組み込み関数を使用しているような感覚で使うことができます．CMXApplet クラスは，Processing の主要クラスである PApplet クラスのサブクラスとなっていますので，Processing の主要機能も利用することができます．

```
import jp.crestmuse.cmx.processing.*

class MyApplet extends CMXApplet {
  void setup() {
    size(320, 240)
    noLoop()
    mp3read('sample1s.mp3')
    playMusic()
  }

  void draw() {
    ellipse(width/2, height/2, width-40, height-40)
  }

}

MyApplet.start("MyApplet")
```

3 音楽を再生する

- CMXController クラスに wavread，mp3read，smfread というメソッドがあるので，音楽ファイルの形式に合わせてこれらのメソッドを利用することで，音楽ファイルを読み込むことができます．
- 音楽ファイルの読み込み後は，同クラスの playMusic メソッドを呼び出すと再生が開始します．
- 再生中の音楽を停止するには，同クラスの stopMusic メソッドを使います．現在，音楽を再生中かどうかを確かめるには，同クラスの isNowPlaying メソッドを使います．

Processing:

```
import jp.crestmuse.cmx.processing.*;

CMXController cmx = CMXController.getInstance();

void setup() {
  cmx.mp3read(createInput("sample1.mp3"));
  cmx.playMusic();
}

void draw() {

}

void mouseClicked() {
  if (cmx.isNowPlaying()) {
cmx.stopMusic();
  } else {
cmx.playMusic();
  }
}
```

4　音楽再生中に現在の再生位置を取得する

- 音楽を再生中にCMXControllerクラスのgetMicrosecondPositionメソッドを呼び出すと，現在の再生位置をマイクロ秒単位で取得できます.

Processing:

```
import jp.crestmuse.cmx.processing.*;

CMXController cmx = CMXController.getInstance();

void setup() {
  cmx.mp3read(createInput("sample1.mp3"));
  cmx.playMusic();
}

void draw() {

}

void mouseClicked() {
  println(cmx.getMicrosecondPosition());
}
```

5　音楽の再生・停止に連動して何かをする

- 音楽の再生・停止に連動して何か処理を行いたい場合は，MusicListener インタフェースを実装したクラスを作り，そのインスタンスを引数として，CMXController クラスの addMusicListener メソッドを実行します．CMXApplet クラスを継承してアプリケーションを作成している場合は，CMXApplet クラスはすでに MusicListener インタフェースを実装していますので，自分が作っているクラスの中に musicStarted, musicStopped, synchronize の各メソッドを作るだけで OK です.

Groovy:

```
import jp.crestmuse.cmx.processing.*

class MyApplet extends CMXApplet {
  void setup() {
    mp3read('sample1s.mp3')
    playMusic()
  }

  void musicStarted() {
    println('The playback of music has started.')
  }

  void musicStopped() {
    println('The playback of music has stopped.')
  }

  void synchronize() {
    print('.')
  }
}

MyApplet.start('MyApplet')
```

Processing:

```
import jp.crestmuse.cmx.processing.*;
import jp.crestmuse.cmx.sound.*;

CMXController cmx = CMXController.getInstance();

class MyMusicListener implements MusicListener {
  void musicStarted(MusicPlaySynchronizer musicSync) {
    println("The playback of music has started.");
  }

  void musicStopped(MusicPlaySynchronizer musicSync) {
    println("The playback of music has stopped.");
  }

  void synchronize(double currentTime, long currentTick,
                   MusicPlaySynchronizer musicSync) {
    print(".");
  }
}

void setup() {
  cmx.mp3read(createInput("sample1s.mp3"));
  cmx.addMusicListener(new MyMusicListener());
  cmx.playMusic();
}

void draw() {

}
```

6 MIDIイベントをリアルタイムでやりとりする

- MIDI イベントなどの情報をリアルタイムで処理する際には,「モジュール」というものを使います.「モジュール」とは，特定のデータを入力すると何らかの処理を行って出力する小さな部品プログラムです.

- 「モジュール」は，原則としてSPModuleというクラスのサブクラスになっています.

- 「モジュール」を利用する大まかな流れは，次のとおりです.
  (1) 利用したい「モジュール」のインスタンスを生成する.
  (2) 生成したインスタンスを「登録」する.
  (3) どのモジュールの出力をどのモジュールに入力させるか，というデータの流れを定義する.
  (2) は CMXController クラスの addSPModule メソッド，(3) は同クラスの connect メソッドを用います．(1) はモジュールによって異なりますが，いくつかのモジュールは，CMXController クラスにメソッドが用意されています.

Processing:

```
import jp.crestmuse.cmx.processing.*;
import jp.crestmuse.cmx.amusaj.sp.*;

CMXController cmx = CMXController.getInstance();

void setup() {
  MidiInputModule vkb = cmx.createVirtualKeyboard(this);
  MidiOutputModule midiout = cmx.createMidiOut();
  cmx.addSPModule(vkb);
  cmx.addSPModule(midiout);
  cmx.connect(vkb, 0, midiout, 0);
  cmx.startSP();
}

void draw() {

}
```

7 MIDIデバイスを選択する

- CMXControllerクラスのshowMidiInChooserメソッド，showMidiOutChooserメソッドを呼び出すと，MIDI入力デバイス，MIDI出力デバイスの選択ダイアログが表示されます.
- 選択後，CMXControllerクラスのcreateMidiInメソッド，createMidiOutメソッドを用いることで，選択されたMIDI入力デバイスからのMIDI信号を受け付けるモジュール，選択されたMIDI出力デバイスにMIDIイベントを送信するモジュールを取得できます.

Processing:

```
import jp.crestmuse.cmx.processing.*;
import jp.crestmuse.cmx.amusaj.sp.*;

CMXController cmx = CMXController.getInstance();

void setup() {
  cmx.showMidiInChooser(this);
  MidiInputModule midiin = cmx.createMidiIn();
  cmx.showMidiOutChooser(this);
  MidiOutputModule midiout = cmx.createMidiOut();
  cmx.addSPModule(midiin);  cmx.addSPModule(midiout);
  cmx.connect(midiin, 0, midiout, 0);
}

void draw() {

}
```

8 マイク入力のリアルタイム処理

- CMXController クラスの createMic メソッドを用いると，マイクから入力された音声を受け付けるモジュールを取得することができます.
- このモジュールは，マイクから入力された音声波形を短い断片に切り出して（「窓をかける」という）double 型の配列（正確には DoubleArray オブジェクト）として出力する，と言う処理を繰り返し行います.
- 音声処理のための設定パラメータを記述してファイル（config.xml）を別途用意する必要があります.

Groovy:

```
import jp.crestmuse.cmx.processing.*
import jp.crestmuse.cmx.amusaj.sp.*
import jp.crestmuse.cmx.math.*
import static jp.crestmuse.cmx.math.Operations.*

class MyApplet extends CMXApplet {
  class MicPrint extends SPModule {
    void execute(Object[] src, TimeSeriesCompatible[] dest) {
      background(255)
      rect(0, 0, (int)(max(abs(src[0])) * width), height)
    }
    Class[] getInputClasses() { [DoubleArray.class] }
    Class[] getOutputClasses() { [] }
  }
  void setup() {
    size(640, 60)
    fill(255, 0, 0)
    readConfig("config.xml")
    def mic = createMic()
    def micpr = new MicPrint()
    addSPModule(mic)
    addSPModule(micpr)
    connect(mic, 0, micpr, 0)
  }
  void draw(){}
}
CMXApplet.main("MyApplet")
```

Processing:

```
import jp.crestmuse.cmx.processing.*;
import jp.crestmuse.cmx.amusaj.sp.*;
import jp.crestmuse.cmx.math.*;

CMXController cmx = CMXController.getInstance();

class MicPrint extends SPModule {
  void execute(Object[] src, TimeSeriesCompatible[] dest) {
    background(255);
    DoubleArray wav = (DoubleArray)src[0];
    double amp = Operations.max(Operations.abs(wav));
    rect(0, 0, (int)(amp * width), height);
  }
  Class[] getInputClasses() {
    return new Class[]{DoubleArray.class};
  }
  Class[] getOutputClasses() {
    return new Class[0];
  }
}

void setup() {
  size(640, 60);
  fill(255, 0, 0);
  cmx.readConfig(createInput("config.xml"));
  WindowSlider mic = cmx.createMic();
  MicPrint micpr = new MicPrint();
  cmx.addSPModule(mic);
  cmx.addSPModule(micpr);
  cmx.connect(mic, 0, micpr, 0);
  cmx.startSP();
}

void draw() {

}
```

9　行列計算 (1) 平均・共分散・特異値分解

- Groovy の演算子オーバーロード機能を用いると，行列をあたかもプリミティブな変数として加減乗除を行うことができます.
- jp.crestmuse.cmx.math パッケージの Operations クラスや MathUtils クラスを mix-in することで，これらのクラスに用意されているメソッドを，あたかも行列を表す変数自身が持っているメソッドかのように扱うことができます.

```
import jp.crestmuse.cmx.math.*
import static jp.crestmuse.cmx.math.MathUtils.*

DoubleArray.mixin(Operations)
DoubleArray.mixin(MathUtils)
DoubleMatrix.mixin(Operations)
DoubleMatrix.mixin(MathUtils)

def x = createDoubleMatrix([[1.0, -1.0], [0.5, -1.5],
                            [0.4, -0.8], [0.8, -0.9]])
println('元の行列：' + x.toString())
println()
println('列毎の平均値：' + x.meanrows().toString())
println('共分散行列：' + x.cov().toString())
println()
println('特異値分解')
(U, S, V) = x.svd()
println("U = ${U.toString()}")
println("S = ${S.toString()}")
println("V = ${V.toString()}")
println()
```

10　行列計算 (2) 固有値計算

- Groovy の演算子オーバーライド，Mix-in 機能を活用することで，行列計算がかなりしやすくなっています．下の例は，2 × 2 型行列に対して固有値を求めています．

```
import jp.crestmuse.cmx.math.*
import static jp.crestmuse.cmx.math.MathUtils.*

DoubleMatrix.mixin(Operations)
DoubleMatrix.mixin(MathUtils)

def y = createDoubleMatrix([[1.0, -5.0], [-5.0, 1.0]])
println("元の行列：${y.toString()}")

(V, D) = y.eig()
println("V = ${V.toString()}")
println("D = ${D.toString()}")
```