Chapter 16. 画像とサウンド、そしてスレッド

１６－１．例外の対処

１６－１－１．例外とtry～catch構文

例外（Exception）とは、プログラムを実行している最中に発生する何らかのエラーのことを指しています。例えば、次のような場合に例外が発生します。これらの例外は、実行最中でないと起こるかどうかわからないものです。

ユーザが強制的に入力を終了させた
指定したファイルが存在しない
ゼロで割った（整数の場合）
配列でインデックスがサイズを超えた
指定されたカラーが存在しない

Javaではプログラム自身で、例外発生時の制御をすることができます。もし例外が発生した場合、次善の措置を記述することができます。次のようなtry～catch構文で記述します。

▼例外発生対処の簡単な書式

```
try {
        何かの処理

} catch ( Exception e ) {
        問題が起こったときにする処理
}
```

この構文の動作は、「何かの処理」を行なっている最中に、もし例外が発生したら、そこで処理を中断して、「問題が起こったときにする処理」に制御を移すためのものです。例外が発生しなければ、そのまま処理を継続することになります。例外発生時には、変数*e*に例外の内容を示すオブジェクトが代入されています。

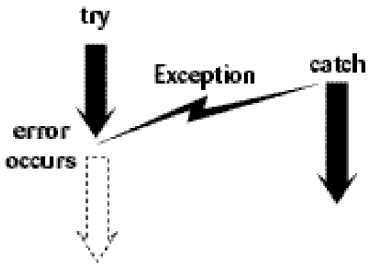

図16-1　例外とtry～catch構文

この構文を使って、例外を発生させるような文を記述してみましょう。次の例は、整数除算で、分母に0を持ってきた場合です（注１）。

```
try {       int x = 0;
            int y = 10 / x;
}
catch ( Exception e ) {
            System.err.println( "整数は０で割らないでください" );
}
```

「問題が起こったときにする処理」としては、上の例のようにエラーメッセージを端末に表示するのが妥当な方法でしょう。通常はSystem.out.printlnや、System.out.printを用いて文字列を表示していますが、これはシステム（Javaを実行している環境）が用意しているSystem.outという標準出力端末（通常は端末画面）に文字列を表示させるものです。System.errはSystem.outと同様に用いることができるのですが、特にエラー情報を中心に表示させるという形で役割分担がされています。

```
try {       int  x = 0;  int  y  = 10 / x;  }
catch ( Exception  e ) {       System.err.println( e );       }
```

エラーが発生した際の例外（Exception）は、変数*e*に代入されますが、この例外は、文字列に変換可能である（printlnメソッドなどの場合は自動的に文字列に変換される）。文字列に変換されたときは、短いエラーメッセージになります。上の例では、これをエラー情報として表示しています。この場合には、次のようなメッセージが端末に表示されることになります。

```
java.lang.ArithmeticException: / by zero
```

また、例外がどこで起こったかをエラー情報として表示させることもできます。この場合には、printStackTraceという名前のメソッドを使用します。

```
try {       int  x = 0;  int  y  = 10 / x;  }
catch ( Exception  e ) {       e.printStackTrace( );          }
```

なお、例外が代入される変数は、次のように別の名前でも構いません。ただし、例外に対処する部分で同じ名前の変数名を使い続けるようにしてください。

```
try {       int  a[ ] = new  int[ 5] ,  x = 8;   a[ x ]  = 10;       }
catch ( Exception  error    ) {            error.printStackTrace( );    }          // errorという変数名で統一
```

従来のプログラムの方法では、何かのエラーが発生した際は、メソッド呼出しの戻り値をif文などを持ちいて調べる必要がありました。これをJavaでは例外処理として、直接記述できるようになったのです。以下の２つの記述はファイルを開いてデータを読み込むための架空の記述（注２）ですが、従来の記述とJavaでの記述の対比をしています。try～catch構文がある方が、流れがわかりやすいのではないでしょうか？

▼従来の記述

```
if  ( (file = openFile(  "sample file"  ) )  ==  ERROR  ) {              // ファイルのオープン
        System.err.println(  "sample file does not exsist." );
        return ERROR;
}
if ( (data = file.readLine( )  ) == ERROR ) {                            // データの読み取り
        System.err.println( "file not contains any data" );
        return  ERROR;
}
file.close( );                                                           // ファイルのクローズ
```

▼Javaでの記述

```
try  {
        File   file = openFile( "sample file"  );
        String   data = file.readLine( );
        file.close( );
}  catch ( Exception  e ) {
        System.err.println( "File Error" + e );
        return ;
}
```

※注１　実数の場合は、0で割るとNaN（Not a Number）という無限大を表す特別な数になります。
※注２　実際の記述方法は後の章で説明します。

１６－１－２．例外とその対処

例外が発生したときに何もしないで、ただ単に次の処理に進みたいときは、次のように、「問題が起こったときにする処理」の部分に何も書かなくても構いません。

```
try {    何かの処理    } catch ( Exception e ) { }
```

しかしながら、エラーが発生したら、そのエラーは後の処理に影響をもたらします。例えば、画像ファイルが存在しない場合のに、画像ファイルを読み込むときに最初の例外が発生しますが、続けて、その画像を表示させたりするときにも例外が発生します。例外が発生しても、上のように記述されていればJavaプログラムは処理を続けます。

AppletViwerなどでも、例外が起こったことだけを伝えて処理を続行させるようにしています。たとえば、カラーの指定方法が間違っていた場合には、次のようにどこで例外が発生したかを例外の種類、短いエラーメッセージと共に出力しながらも、処理は続行されます。この場合は、例外の種類がIllegalArgumentExceptionで、緑（Green）成分の色指定の値が範囲を超えていたことを示しています。ColorQueryというクラスのpaintメソッドで例外が発生したことを示しています。

```
java.lang.IllegalArgumentException: Color parameter outside of expected range: Green
at java.awt.Color.testColorValueRange(Color.java:180)
at java.awt.Color.<init>(Color.java:187)
at ColorQuery.paint(ColorQuery.java:24)
at java.awt.Component.update(Component.java:1083)
```

より高度なプログラミングを行なう場合や、企業商品として使われるアプリケーションやアプレットであれば、想定されるエラーに対して、それによって起こる障害回復を最小限に抑えるように処理を記述する必要があるでしょう。たとえば、画像ファイルがなければ、ユーザにファイルを指定してもらうように処理するようにしなければいけません。

１６－１－３．例外の種類

Javaでは例外もクラスとして定義されていますので、例外にも継承関係があり、細かく分類されています。一番の大もとは、Exceptionです。実行時に言語的に起こり得る例外は、RuntimeExceptionとして分類されていますし、AWTなどのクラスライブラリで起こるものはAWTException、入出力関係で起こるのはIOExceptionというように分類されています。代表的なわかりやすい例外の幾つかを以下の表に階層関係を示しながら挙げておきます。

class java.lang.Throwable	例外のための仕様を定めたクラス
class java.lang.Exception	例外全体
class java.awt.AWTException	AWT関係
class java.io.IOException	入出力関係
class java.lang.InstantiationException	オブジェクトを作るとき
class java.lang.InterruptedException	スレッド関係
class java.lang.RuntimeException	実行時の例外
class java.lang.ArithmeticException	算術演算
class java.lang.ArrayStoreException	配列の要素
class java.lang.IllegalArgumentException	引数の間違い
class java.lang.IndexOutOfBoundsException	配列や文字列のインデックス
class java.lang.NegativeArraySizeException	配列のサイズが負の場合
class java.lang.NullPointerException	オブジェクトがない

図16-2　代表的な例外

このため、より細かな例外から、対処を記述したいという場合があります。このような記述法がtry文の中には

用意されています。

```
try {
        何かの処理

} catch ( より細かな例外のクラス  e ) {
        問題が起こったときにする処理
} catch ( より大きめな例外のクラス２  e ) {
        問題が起こったときにする処理
} finally (
        必ず実行される処理
}
```

１６－１－４．例外を発生させるメソッド

メソッドの処理内容によっては、例外を発生させる可能性があるときは、そのような指定がされています。たとえば、以下はBufferedReaderクラスに定義されていますreadLineというメソッドの仕様なのですが、これにはこのメソッドを利用した場合、IOExceptionを発生させる可能性があるという指定がついています。

```
public String readLine( ) throws IOException
```

例外を発生させる可能性のあるメソッドを用いるときは、必ずtry〜catch構文の中で使う必要あります。たとえば、上のreadLineでしたら、次のように構文を記述します。

```
try { br.readLine( ); } catch ( IOException except ) { System.err.println( except ); }
```

catchの部分には、IOExceptionという指定がされているのですが、これは単にExceptionと記述しても構いません。Exceptionならば、すべての例外の種類が含まれているからです。

１６－１－５．try〜catch構文と変数のスコープ

注意しなければならないのは、try〜catch構文を用いたときの変数のスコープです。tryのブロックの中で宣言して導入した変数は、try構文が終わってしまうと存在しませんので、参照することはできません。たとえば、次のような断片では、コンパイル時にエラーが発生していまいます。

```
try {
        Image  melo = getImage( new URL( "http://www.melodaisuki.com/melo.gif" ) );
} catch ( Exception e  ) {
        System.err.println( e );
}
gc.drawImage( melo, 0, 0, this );                    // 変数meloは既にここでは存在しない
```

もし、try〜catch構文以降も変数を参照したい場合は、次のように変数を構文の外で宣言する必要があります。

```
Image melo;                                          // 変数meloをブロックの外側で定義した
try {
        melo = getImage( new URL( "http://www.melodaisuki.com/melo.gif" ) );
} catch ( Exception e  ) {
        System.err.println( e );
}
gc.drawImage( melo, 0, 0, this );
```

１６－２．イメージの表示

１６－２－１．一般的なイメージの表示方法

AWTクラスライブラリには、GIFおよびPNGやJPEG形式の画像ファイルを扱うことができるImageクラスが用意されており、画像をアプレットの中に表示させることができます。画像の読込みと表示は、次のような３段階で行ないます。まず、変数を宣言し、その変数に対して読込みを行ないます。最後に、変数を使って表示します。

▼画像の読込みと表示

1. イメージ型の変数を宣言する

```
Image        イメージ型の変数;
```

2. イメージを読み込む

```
イメージ型の変数 = getImage( URLの指定 );
```

3. イメージを表示する

```
g.drawImage( イメージ型の変数 , x座標 , y座標 , this );
```

宣言と読込みは同時に行なうことができますので、２段階にすることもできますし、getImageをdrawImageメソッドの引数に入れてしまえば、読込みと表示を１行で表現することもできます。drawImageの引数の４番目のthisは、アプレット自身を示しています。ここにはImageObserverクラスのオブジェクトがはいるのですが、通常はアプレットがImageObserverになっていますので、**this**を指定するのが一般的です。

★イメージを表示させてみる

ここで表示させるイメージは、うさぎのMelo君である。クラスファイルと同じフォルダ（ディレクトリ）に、melo.gifという画像ファイルがあるとします。次のようにアプレットのプログラムを記述します。

```
import  java.awt.*;
import  java.applet.*;

public class MeloViwer  extends Applet {
        public void paint( Graphics g ) {
                Image    melo = getImage(  getCodeBase( ), "melo.gif" );
                g.drawImage( melo, 0, 0, this );
        }
}
```

図16-3　melo.gifに保存されている画像（にんじんを食べるメロ）

★画像ファイルはどこに置くのか？
Xcodeの場合、アプレットは、どこで実行されているのかというと、プロジェクトファイルの置かれているフォルダの下にある、「build」という名前のフォルダで実行されています。ですから、画像や音声、あるいは一般のデータファイルなども、この「build」の下においてください。

１６－２－２．URLの指定について

アプレットではセキュリティの問題から禁止されているのですが、アプリケーションなどで、画像ファイルを読み込む際には、URLの指定をしなければなりません。URLについては、次の３つの指定方法が可能です。ネットワークで接続されているまったく別のコンピュータに画像が置かれているときは、最初の方法で指定します。ローカルに（アプレットではWebサーバ側に）画像ファイルがあるときは、後半の２つのいずれかを指定します。

★URLの文字列をそのまま使う場合

ネットワーク上でのURL（Unified Resouce Locater）を直接指定して、画像ファイルの位置を指定します。この場合、次のような形式でURLを文字列で指定します。

```
new URL( URLの文字列 )
```

この指定の場合は、URLクラスのオブジェクトを作り出すことになります。このとき、例外を発生する可能性がありますので、try〜catch構文を使う必要があります。ここで生成されたオブジェクトは変数に代入しておいても構いません。たとえば、架空のURLアドレスhttp://www.shuwasystem.co.jp/java/pictures/duke.gifに表示したい画像がある場合を考えてみましょう。次のように指定します。

```
Image   duke ;
try {
        duke = getImage( new URL( "http://www.shuwasystem.co.jp/java/pictures/duke.gif" ) );
} catch ( Exception e ) {
        System.err.println( e );
}
```

上の記述では一行で画像の読み込みまでを行なっていますが、これをURLクラスの変数を用いて、次のように分けて書いても構いません。なお、このURLクラスを使う場合は、**import** java.net.* ;をプログラムの最初にいれておく必要があります。

```
Image   duke ;
try {
        URL dukeaddr = new URL( "http://www.shuwasystem.co.jp/java/pictures/duke.gif" );
        duke = getImage( dukeaddr );
} catch ( Exception e ) {
        System.err.println( e );
}
```

ただし、このURLを直接指定する方法は、アプレットではセキュリティ上の問題から禁止されています。ローカルなアプリケーションだけで使うようにして下さい。アプリケーションでグラフィックスを使う方法は、付録の章で説明します。

★classファイルからの相対的な位置指定をする場合

Appletが保存されているサーバあるいはアプリケーションを動かしているコンピュータ上で、Javaの実行形式であるクラスファイルが保存されているディレクトリ（フォルダ）からの相対的な位置をパスで指定する方法です。クラスファイルのディレクトリの位置については、次のメソッドを用いて指定します。

```
getCodeBase( )
```

このメソッドと、相対的な画像ファイルへのパスを文字列で指定することによって、画像ファイルを読み込む

ことができるようになります。以下の例は、クラスファイルが置かれているディレクトリ（フォルダ）の親ディレクトリ（フォルダ）上にあるpicturesという名前のディレクトリ（フォルダ）の中にあるmoon.gifという名前の画像ファイルを指定しています。

```
getImage(  getCodeBase( ),  "../pictures/moon.gif" );
```

相対的な指定をするときは、通常のURLで相対的な指定をするために使われる次のような記号が用いることができます。

.	ピリオド	そのディレクトリ（フォルダ）を示します
.	２つのピリオド	親ディレクトリ（フォルダ）を示します
/	スラッシュ	ディレクトリ（フォルダ）の区切りを示します

★HTMLファイルからの相対的な位置指定をする場合

この方法も、セキュリティ的な問題から、現在はほとんど使えなくなってしまったのですが、アプレットを表示させているHTMLファイル（Webページ）が保存されているサーバ上での、HTMLファイルが保存されているディレクトリ（フォルダ）からの相対的な位置をパスで指定する方法です。HTMLファイルのディレクトリの位置については、次のメソッドで指定します。

```
getDocumentBase( )
```

たとえば、アプレットを表示するためのHTMLファイルが置かれているディレクトリ（フォルダ）にpicturesという名前のサブディレクトリがあり、そのディレクトリの中にalto.gifという画像ファイルがあるとします。そのときは、次のように指定します。

```
getImage(  getDocumentBase( ), "pictures/alto.gif" );
```

１６－２－３．ネットワークセキュリティとアプレット

★外部に公開されていないWebサーバへのセキュリティ

企業などでは、Webサーバであっても外部に公開されているものではなく、企業内部のLAN上のコンピュータからしか閲覧できないものがあります。このような内部向けのWebサーバがあった場合、アプレットでURLを指定して画像などを読み込む方法を使えば、そのような内部向けのWebサーバ上にあって組織内部用に公開されている文書や画像、あるいは音声といったファイルを覗き見することができてしまいます。

アプレットは、まったくの外部のWebサーバから移動してきて、手元のコンピュータで実行される訳ですから、手元のコンピュータが内部向けのWebサーバと繋がっている場合、アプレットを利用して、外部のWebサーバなどに、内部向けのWebサーバに保管されているファイルの内容を逆に転送することができるのです。

アプレットは、基本的にはアプレットが置かれているWebサーバ上の情報しか閲覧することができません。そのため、文書や画像、あるいは音声といったファイルをアプレット上に読み込むときには、getCodeBaseメソッドを使って、アプレットが所属するWebサーバ上の同じフォルダにあるファイルだけを指定するようにします。また、getDocumentBaseも、アプレットが置かれていないWebサーバ上のHTMLファイルを指定してアプレットを実行させた場合に、そのサーバ上のフォルダをみれることができるということで、セキュリティ的な問題があるということで、アプレットでは使わないようにされています。

なお、Internet ExplorerやNetscape Navigator、あるいはSafariといった標準的に使われているWebブラウザ上でアプレットを稼働させたときは、URLを指定して別のWebサーバから画像や音声をダウンロードするようなアプレットは、権限違反として実行されません。

１６－２－４．画像を表示する位置と幅と高さ

drawImageメソッドでは、次のような２つのタイプが用意されています。x座標、y座標は画像を矩形と考えて、その左上の角の位置になります。

```
drawImage( イメージオブジェクト，x座標 , y座標 , イメージオブザーバ );
drawImage( イメージオブジェクト，x座標 , y座標 , 高さ , 幅 , イメージオブザーバ );
```

２つ目の方のdrawImageメソッドでは、画像の幅と高さを指定することができます。この場合は、元の画像をその幅と高さに合うように自動的に拡大・縮小が行なわれます。たとえば、次の記述は幅も高さも128ピクセルに収まるように画像を表示させます。

```
gc.drawImage( melo, 50, 50, 128,  128,  this );
```

読み込んできた画像の元々の幅と高さを得るためには、Imageクラスのオブジェクトに用意されているgetWidthメソッドとgetHeightメソッドを利用します。

```
イメージオブジェクト.getWidth( イメージオブザーバ );
イメージオブジェクト.getHeight( イメージオブザーバ );
```

これらのメソッドは、引数にイメージオブザーバを必要とします。通常は、アプレット自身がイメージオブザーバになっているので、**this**を指定します。次の例は、変数*width*と*height*に画像の幅と高さを求めています。変数の宣言と代入とを同時に行なっています。

```
Image    melo = getImage(  getCodeBase( ), "melo.gif" );
int  width  = melo.getWidth( this );
int  height  = melo.getHeight( this );
System.out.println( "Image width: " + width + "   height: " + height );
```

次の例は、幅・高さとも、半分のサイズで表示させるための記述になっています。

```
gc.drawImage( melo, 0, 0, melo.getWidth( this ) /2,  melo.getHeight( this ) /2,  this );
```

★マウスにあわせて表示を変えていくアプレット

マウスをクリックしたら次の画像を表示するようなアプレットを作ってみましょう。画像ファイルは９つあり、それぞれmelo1.jpg～melo9.jpgというファイル名がついているとします。画像ファイルはWebサーバ上のドキュメントが置かれているディレクトリ（フォルダ）上のサブディレクトリ（フォルダ）である“pictures/”の下に格納されているとします。画像の読込みは、Initメソッドで予め行なっておきます。画像のために配列を用意します。melopicsはImageクラスの配列で、９つの要素を持ちます。変数*current*は、現在配列の何番目の画像を表示すればよいのかを示しています。マウスがクリックされたら、この変数の値を１増やして、再描画させます。

```
import  java.awt.*;
import  java.awt.event.*;
import  java.applet.*;

public class   MeloAnime    extends Applet  implements MouseListener {
        Image    melopics [ ] = new Image[ 9 ];
        int      current;

        public void init( ) {
                addMouseListener( this );
                for ( int  i=0; i<melopics.length ; i++ ) {
                        melopics[ i ] = getImage( getCodeBase( ),
                                "pictures/melo" + (i+1) + ".jpg"  );
                }
        }
```

```
        public void paint( Graphics gc ) {
                gc.drawImage( melopics[ current ], 0, 0, this );
        }

        public void mouseClicked( MouseEvent e ) {
                current = (current + 1) % 9;
                repaint( );
        }
        public  void  mouseEntered( MouseEvent   e ) { repaint( ); }
        public  void  mousePressed( MouseEvent   e ) { }
        public  void  mouseReleased( MouseEvent   e ) { }
        public  void  mouseExited( MouseEvent   e ) { }
}
```

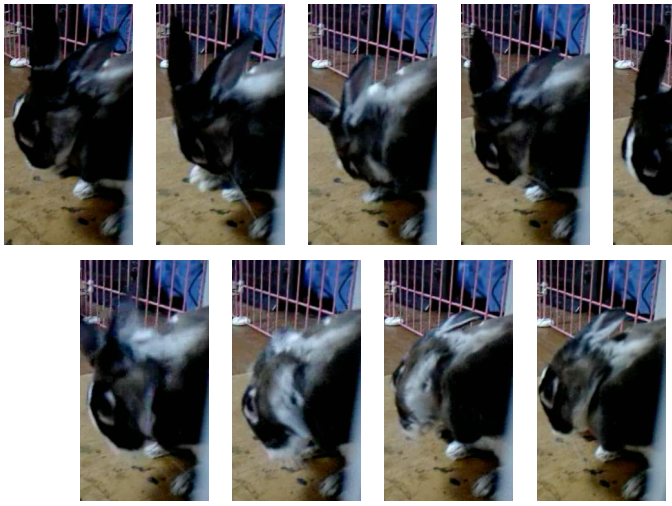

図18-5　画像ファイルmelo1.jpg〜melo9.jpg

１６－２－５．メディアトラッカーを使う

アプレットでは、しばしば画像ファイルが大きすぎて、実際の表示が間に合わないことがあります。それを避けるために、AWTクラスライブラリには画像が読み込まれたかどうかをチェックするためのオブジェクトが用意されています。それはメディアトラッカー（Media Tracker）と呼ばれています。これを利用するには、まずアプレット上でメディアトラッカー・オブジェクトを生成します。

```
MediaTracker  mt = new  MediaTracker( this );
```

次に、画像を示すイメージオブジェクトをaddImageメソッドを使って、このメディアトラッカー・オブジェクトに登録していきます。

```
mt.addImage(  イメージオブジェクト，番号　);
```

この番号には、適当な整数値を与えてください。後で画像がロードされたかどうかチェックするのに用います。

指定されたイメージオブジェクトがロードされたかどうかは、checkIDメソッドを使います。

```
mt.checkID( 番号, フラグ );
```

フラグの部分には、次のような論理値を指定します。

true	ロードがされていなければロードを開始させる
false	ただ単にロードされているかどうか調べるだけ

このcheckIDメソッドは、論理値を返し、**true**ならば、ロードが完了したことを示し、**false**ならばロードが完了していないことを示します。そのため、大抵はif文などの条件として用いられます。次の記述は、３番の画像がロードされているかどうかチェックしています。

```
if ( mt.checkID( 3, true ) ) {      // もし３番の画像がロードされていた場合は、、
        ....                        // ただし、ロードされていなければロードを開始させる
```

登録されたすべての画像がロードされたかどうかを確かめる場合には、checkAllメソッドを使います。checkIDメソッドと同様にフラグを指定します。メソッドの返り値もcheckIDと同じです。

```
if ( mt.checkAll( true ) ) {        // もしすべての画像がロードされたら、、、
```

他にも、指定された画像がロードされるまで待つwaitForIDメソッドや、登録されたすべての画像がロードされるまで待つwaitForAllメソッドがあります。たぶん、こちらの方を良く使うのではないでしょうか。フラグがないだけで、使い方はcheckIDあるいはcheckAllと同じです。なお、waitForIDとwaitForAllは例外を発生する可能性がありますので、try〜catch構文の中にいれておく必要があります。

```
if ( mt.waitForAll( ) ) {    // すべての画像のロードを待ち、それが完了したら、、、、
```

例えば、先ほどの９つの画像がロードされるのを待つようにするために、プログラムの一部を修正してみました。initメソッドの該当部分を書き換えます。

インスタンス変数の宣言：

```
Image    melopics [ ] = new Image[ 9 ];
MediaTracker  mt = new  MediaTracker( this );
```

initメソッドの該当部分の修正：

```
for ( int  i=0; i<melopics.length; i++ ) {
        melopics[ i ] = getImage( getCodeBase( ), "pictures/melo" + (i+1) + ".jpg" );
        mt.addImage( melopics[ i ], i );
}
mt.waitForAll( );
```

メディアトラッカーは、画像をロードしている最中には、その旨を示すような文字列を替わりに表示させるときに用いられることが多いでしょう。たとえば、先ほどの画像ファイルを読み込んで表示するアプレットで、読み込んでいる間は、Loading...とメッセージを出したい場合は次のように修正します。while文でcheckIDを使ってロードが終わったかどうか確認しています。Thread.sleepメソッドは、try文の中でしか使えないのですが、与えられた整数値をミリ秒とみなして、その間アプレットの実行を止め、他の処理をさせるようにさせています。Thread.sleep(300);は0.3秒アプレットが時間経過を待つことを意味しています。

```
import  java.awt.*;
import  java.applet.*;

public class  WaitingMelo  extends Applet {
        public void paint( Graphics g ) {
                try {
                        MediaTracker  mt = new  MediaTracker( this );
                        Image    melo = getImage(  getCodeBase( ),  "pictures/melo.gif"  );
                        mt.addImage( melo, 1 );
                        while ( mt.checkID( 1, true ) == false ) {
                                g.drawString( "Loading...", 40, 40 );
                                Thread.sleep( 300 );
                        }
                        g.drawImage( melo, 0, 0, this );
                } catch( Exception e ){ System.err.println( e ); }
        }
}
```

１６－３．サウンドを鳴らす

１６－３－１．鳴らすことができるサウンドの形式

Java2から、次のような一般的な音声ファイルをサポートすることができるようになりました。このため、多くの音声ファイルは変換する必要はありません。また、MIDI形式のファイルも次のようなファイルをサポートしています。

音声ファイルの形式：　AIFF, WAVE, AU
- 音声ファイルのコーディング方法：　PCM（リニア）, μ-law，その他
- 音声ファイルの標本周波数：　8kHz ～ 48kHz
- 音声ファイルの 量子化ビット数：　8 または 16 bit
- 音声ファイルのチャンネル数：　Mono （＝１）または Stereo（=2）

MIDI形式（楽譜形式）ファイル：
- MIDI Type 0, MIDI Type 1, RMF

Javaでは音声ファイルの再生は、標準では16ビットのステレオで、22kHzの周波数で行なわれます。ただし、実行されるコンピュータで、この品質で再生できないときは、8ビットのモノラル、8kHzでの再生になってしまいます。MIDIファイルに関しては、内蔵のデジタルシンセサイザー音源を使って再生されます。

１６－３－２．アプレット上で直接演奏させる

アプレットでは、これらの形式のサウンドをplayメソッドによって再生することができます。このメソッドは例外を発生する可能性がありますので、try～catch構文の中に入れておく必要があります。

```
try {
        play( URLの指定 );
} catch( Exception e ){
        System.err.println( e );
}
```

★URLの指定について

画像と同じように次のような３種類の指定の仕方ができます。最初のものは、ネットワーク上のURLを指定して直接ロードするものです。これは、セキュリティ的な問題から避けた方が無難でしょう。後の２つはWebサーバ上の音声ファイルを利用するものですが、getCodeBase使うのが良いでしょう。

```
play( new URL( URLの文字列 ) );

play( getCodeBase( ), "ファイル名の指定 " );

play( getDocumentBase( ), "ファイル名の指定 " );
```

１６－３－３．AudioClipクラスを使う

★アプレットに用意されているオーディオ用のクラスAudioClipについて

Appletでは、AUファイル（音声ファイル）を再生するためのAudioClipクラスが用意されています。複数のAUファイルを同時に再生させることができます。AudioClipに関しては次のようなメソッドが用意されています。

●AUファイルをロードする

getAudioClip(URLの指定);

アプレットで、音声ファイルをロードしてAudioClipクラスのオブジェクトを用意してくれるものです。URLの指定は、playメソッドと同様に３種類の指定をすることができますが、Webサーバ上の音声ファイルを指定する２種類のどちらかを使った方が無難でしょう。たとえば、次のように記述します。

```
AudioClip       au = getAudioClip(  getCodeBase( ),  "spacemusic.au" );
```

●AUファイルを再生する

AudioClipクラスのオブジェクトには次の３つのメソッドが用意されています。

play( )	1回だけ再生する
loop( )	繰り返し何回も再生する
stop( )	再生を止める

たとえば、上記の変数auが指し示すAudioClipクラスのオブジェクトを使った場合は、次のように記述します。

```
au.play( );  au.loop( );  au.stop( );
```

★ボタンで再生を制御するアプレット

Webドキュメントのあるディレクトリ（フォルダ）のサブディレクトリにある、"sounds/spacemusic.au"というAUファイルを読み込んで、それを繰り返し再生するようなアプレットを作ってみましょう。ボタンを１つ用意しておき、最初はstopボタンにしておきます。もし、ボタンがおされたら再生を中止して、ボタンのラベルをstartに変えます。次にボタンが押されたら、今度は再生を開始して、ボタンのラベルをstopボタンに変えます。

```
import  java.awt.*;
import  java.awt.event.*;
import  java.applet.*;
import  javax.swing.*;

public class AudioTester extends Applet implements ActionListener {
        AudioClip   mysound;
        JButton b;

        public void init( ) {
                b = new JButton( "stop" );
                b.addActionListener( this );
                add( b );
                mysound = getAudioClip(  getCodeBase(  ),  "sounds/spacemusic.au"    );
                mysound.loop( );
        }

        public void actionPerformed( ActionEvent   e ) {
                if ( b.getLabel( ).equals( "stop" ) ) {  mysound.stop( );    b.setLabel( "start" ); }
                else {                                   mysound.loop( );    b.setLabel( "stop" ); }
                repaint( );
        }
}
```

★アプリケーションで音を鳴らす

Java2（JDK1.2）以降は、画像ファイルの表示と同様に音声ファイルもアプリケーションで再生できるように

なりました。アプリケーションの場合は、ローカルに実行されることが前提になっていますから、セキュリティ的な制約はありません。ローカルにあるファイルも、指定したURLから読み出す場合でも両方問題なく再生することができます。

１６－４．スレッド

１６－４－１．スレッドの存在理由と定義の仕方

スレッド（Thread）とは、アプレットやアプリケーションと同時に動く別のプログラム（プロセス）であり、変数などを共通して用いることができるようなものです。たとえば、アプレットにおいて一定時間ごとにアニメーションが動き、「それとは別に」何らかのアクション（ユーザからの入力）に対応するためには、スレッドを用意しなければなりません。このような場合には、アプレットとスレッド側で次のように動作を分ける設計がよく行われます。

アプレット側	アクションに対応する・スレッドを開始させたり、終了させたりする
スレッド側	アニメーションを動かす

アプレットでスレッドを使うときは、スレッドの開始・終了を制御するため、アプレット側にはstartメソッドとstopメソッド、およびスレッド側には、runメソッドを用意します。スレッド側にはstartメソッドだけが用意されています。スレッドのrunメソッドは、スレッド側のstartメソッドが呼ばれるとすぐにも呼ばれるようになります。スレッド側は参照している変数を**null**値に設定すると実行されなくなります。

start → run → → → → → → → → → → → → →終了

Runnableインターフェースを用いて、一つのクラスがアプレットとスレッドの両方を兼ねる場合が通例です。そこで、インスタンス変数で、スレッド制御用のThreadクラスの変数を作ります。

```
Thread   runner;
```

アプレットのstartメソッドでは、スレッドを新たに作りスタートさせます。

```
public void start( ) {
        if ( runner == null ) {          // nullはスレッドが実行されていないことを示す
                runner = new Thread( this );   runner.start( ); } }
```

アプレットのstopメソッドでは、スレッドに終了を告げます。

```
public void stop( ) { if ( runner != null ) {     runner = null; } }
```

runメソッドでは、スレッド側で実行したいことを行います。ただし、アプレットのstopメソッドによって実行中止の例外が発生するので例外処理をしておきます。また、頻繁に処理をしないときは、Thread.sleep　メソッドを使って休憩し、アプレットに処理を回すようにしておきます。

```
public void run( ) {
        try {
                while( runner == Thread.currentThread( ) ) {
                        したいこと
                        Thread.sleep( ミリ秒数 );
                }
        } catch( Exception  e ) {
                System.out.println( e );
        }
}
```

１６－４－２．スレッドプログラミングの例

★時刻を刻々と変化させて表示させる

スレッドは、runメソッドを見るとわかるように、１秒単位で時間を更新しながら表示していくものです。アプレット自身は、マウスがクリックされたときの処理だけを行なっています。マウスがクリックされた場所に、時間を表示し直しています。

```
import   java.awt.*;
import   java.applet.*;
import   java.util.*;
import   java.awt.event.*;

public class DateModifier extends Applet implements Runnable , MouseListener {
        Thread   runner;
        Date   date;
        int    curx = 100, cury = 100;

        public void init( ) { addMouseListener( this );  }
        public void start(  ) {  if ( runner == null ) {
                 runner = new Thread( this );  runner.start( ); } }
        public void stop(  ) {
                if ( runner != null ) {  runner = null;  }  }

        public  void  paint( Graphics g ) {
                 g.drawString( date.toString(), curx, cury ); }

        public void run( ) {
                try {  while (  runner  == Thread.currentThread( ) ) {
                        date = new Date(  );                    // 現在時刻を得ます
                        repaint(  );    Thread.sleep( 1000 );  }
                } catch( Exception e ) {  } }
        public  void mouseClicked(  MouseEvent  e ) { curx=e.getX( ); cury=e.getY( ); repaint( ); } }
        public  void  mousePressed(  MouseEvent    e ) { }
        public  void  mouseReleased(  MouseEvent  e ) { }
        public  void  mouseEntered( MouseEvent e ) { }
        public  void  mouseExited( MouseEvent e ) { }
}
```

★画像を一定間隔でアニメーションのように見せる

次の例は、非常によく使われる例だと考えられます。最初にメディアトラッカーを使って、画像ファイルを何枚か読みこんでおき、その後に、一定間隔としてアニメーションとして見せる例題です。さきほどの、クリックしたら動くメロの画像を、アニメーションとして記述し直してみました。

```
import  java.awt.*;
import  java.applet.*;

public class MeloAnimetor extends Applet  implements Runnable {
        Image    melopics [ ] = new Image[ 9 ];
        Thread   runner;
        int      current;

        public void init(  ) {
                try {
                        MediaTracker  mt = new  MediaTracker( this );
                        for ( int  i=0; i<melopics.length ; i++ ) {
                                melopics[ i ] = getImage( getCodeBase( ),
                                        "pictures/melo" + (i+1) + ".jpg"  );
                                mt.addImage( melopics[ i ], i );
                        }
                        mt.waitForAll( );
                } catch(  Exception e ){ System.err.println( e ); }

        }
```

```
    public void start( ) {  if ( runner == null ) {
            runner = new Thread( this );  runner.start( ); } }
    public void stop( ) {
            if ( runner != null ) {  runner = null;  }  }

    public void paint( Graphics gc ) {
            gc.drawImage( melopics[ current ], 0, 0, this );
    }

    public void run( ) {
            try {  while (  runner  == Thread.currentThread( ) ) {
                    current = (current + 1) % 9;
                    repaint( );
                    Thread.sleep( 1000 );  }
            } catch( Exception e ) {  }
    }
}
```

１６－５．アプレットとWebページとの連携

１６－５－１．アプレットからWebブラウザにWebページを表示させる

アプレットはWebブラウザに対して、指定したWebページを表示させる機能があります。これは、画像ファイルや音声ファイルを表示させたり、再生させたりするのと同じように行なうことができます。そのためにアプレットには、アプレットコンテキスト（AppletContext）というオブジェクトが付随しており、このオブジェクトでは次のようなメソッドが実行可能です。

getAudioClip(URLの指定);	音声ファイルをロードする
getImage(URLの指定);	画像ファイルをロードする
showDocument(URLの指定);	HTMLファイルをWebブラウザにロードさせる

前者２つのメソッドは、既に紹介しましたが、アプレットにもメソッドが実装されているので、アプレットから直接呼び出せることができます。しかし、showDocumentメソッドだけは、アプレットコンテキスト・オブジェクトに対して実行させる必要があります。

アプレットコンテキスト・オブジェクトを得るには、次のようなメソッドを使います。

getAppletContext( );	現在のアプレットのアプレットコンテキストを得る

このメソッドとshowDocumentメソッドを利用して、Webブラウザに特定のページを表示させることができます。具体的には、次のように記述します。

```
getAppletContext( ).showDocument(  URLの指定  );
```

例えば、http://www.javasoft.comのホームページをWebブラウザに表示させるには、次のように記述します。

```
try {
        URL  javasoft =  new  URL(  "http://www.javasoft.com" );
        getAppletContext( ).showDocument(  javasoft );
} catch( Exception  e  ) { System.err.println( e ); }
```

あるいは、Webサーバ上のクラスファイルがあるフォルダにある、別のproceed.htmlというページを表示させるには次のように記述できます。

```
try {
        URL  nextpage =  new  URL(  getCodeBase( ),  "proceed.html" );
```

```
                getAppletContext( ).showDocument(  nextpage );
        } catch( Exception  e  ) { System.err.println( e ); }
```

もちろん、Webブラウザに別のページを表示させれば、元のAppletは表示されなくなってしまいます（アプレットに用意してあればstopメソッドが実行されます）。他にユーザの対象となるページが移ってしまってよいときに、このように記述します。このshowDocumentは、Webページの表示自体が別のページに移ってしまうことから、セキュリティ的には問題ありません。

この機能を利用すれば、下の図のように複数のAppletと複数のWebページで作ることができます。

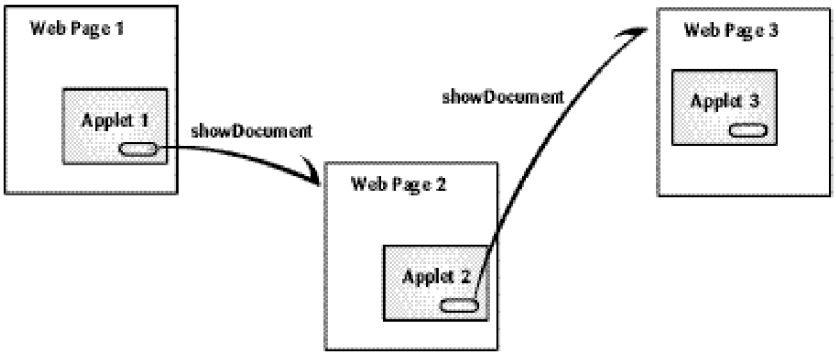

図16-6　アプレットWebページの連携

１６－６．課題

16-1.
マウスクリックに併せて画像イメージの表示を変えていく例題を改良し、マウスを押した位置に表示するようにし、加えて、それぞれに画像にあわせて異なった音を出すように変えてみなさい。たとえば、アプレットを表示させるHTMLファイルが置かれているフォルダのサブフォルダである"sounds/"の中に1.au～9.auという名前の音声ファイルがあるとします。

16-2.
２つのボタンを用意し、１つのボタンが押されたら、画像が拡大し、もう一方のボタンが押されたら、画像が縮小するようなアプレットを作成しなさい。クラス名は、DetailViewerにて。

16-3.
アプレット上にボタンを複数用意し、ボタンが押されたら、指定されたURLのWebページを表示させるようなアプレットを作成してみなさい。クラス名は、WebTripperにて。

16-4.
アプレット上にボタンを２つ用意し、一方のボタンが押されたら、アニメーションが始まり、繰り返し表示し、もう一方のボタンが押されたら、アニメーションが終わるようなアプレットを作成してみなさい。クラス名は、AnimationViewerにて。

16-5.
アプレット上にボタンを用意し、ボタンが押されたら、経過時間が1/10秒単位で表示されるようなストップウォッチのようなアプレットを作りなさい。もう一度ボタンが押されたら、止まるようにします。別のボタンを用意して、経過時間をリセットできるようにしなさい。クラス名は、StopWatchにて。