J3W入門 for Linux講座

水谷純
http://www.nk.rim.or.jp/~jun/index.html

第6回　カメラワーク

3Dアクションゲームでは、ゲームの演出にいろいろな視点が利用されています。今回は、J3Wを使って「空間の見せ方」を実験してみます。J3Wでは、三次元空間に存在する物体がすべて視点になることができます。視点の「移動」や「切り替え」を実際に試してみましょう。

これまでは、サンプルプログラムのソースコードのリストをすべて掲載していましたが、サンプルが長くなってきたので、J3Wに付属しているコードを利用することにします。共用できる機能をまとめたファイルを用意しておくと、プログラムを作成するときにインポートするだけで、その機能を追加できて便利です。

【表1】J3C付属ライブラリ

ファイル	種別	説明
color.j3c	library	色名の定義
key.j3c	library	キーコード名定義
ssolid.j3c	library	単純な形状を生成
zsolid.j3c	library	Z軸方向に伸びる形状生成
xsolid.j3c	library	X軸方向に伸びる形状生成
empty.j3c	class	何もしない空のクラス
eye.j3c	class	簡単な視点クラス
esckey.j3c	class	ESCキー終了チェッククラス
revolutn.j3c	class	回転体生成クラス
fps_pos.j3c	class	表示速度計算クラス

■付属ライブラリの使用法

J3W付属のサンプルプログラムが共用するライブラリは「j3w-643/j3c_script/lib」以下のディレクトリにあります（表1）。表1のファイルをインポートして、クラスならば、「new(クラス名, データサイズ)」を実行すると機能が追加できます。継承してクラスの機能を拡張することもできます。ライブラリの場合は「ライブラリ名.定数名」または「ライブラリ名.メソッド名」で、どのクラスからでもライブラリに登録されている定数やメソッドが利用できます。

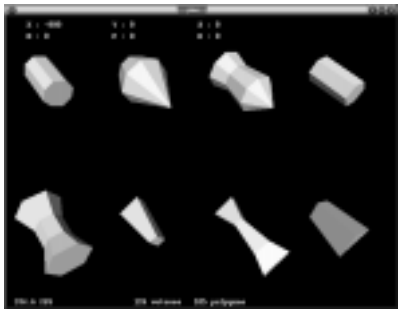
ユーザーのホームディレクトリが/home/jun/で、ここにJ3Wを展開した場合、付属ライブラリのeye.j3cをインポートするにはリスト1のように絶対パスで指定します。また、/home/jun/j3w-643/j3c_script/work/のように、作業ディレクトリをJ3Wを展開したディレクトリに置いた場合、付属ライブラリをインポートするには次のように相対パスで指定できます（本誌付録CD-ROM収録のサンプルプログラムもこの指定方法になっています）。

```
import "../lib/eye.j3c"
```

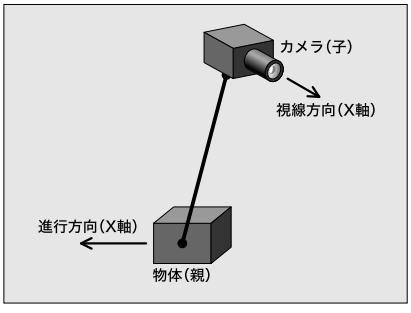
【リスト1】付属ライブラリeye.j3cのインポート

```
import "/home/jun/j3w-643/j3c_script/lib/eye.j3c"
```

【画面1】testsolid.j3cの実行例



【図1】カメラとアーム



画面1は、lib/xsolid.j3cを使った8種類の形状を作成するプログラムの実行例です。付録CD-ROMにソースと実行ファイル（testsolid.j3c、testsolid.j3d）がありますから、実際に実行してみてください。

■カメラワーク

J3Wでは、See()組み込み関数を実行したインスタンスから見ている空間が、画面に表示されます。実体化している（NewObjectを実行するか、childで生成された）インスタンスは、すべて視点となることができます。視線は、物体の原点からX軸方向（前方）に固定されます。視点を持つ物体をここでは「カメラ」と呼ぶことにします。カメラは自由に移動したり、向きを変えることができますが、「歩きながらあたりを見回す」というような人間の目と同じような動きをさせるには、ちょっとした工夫が必要になります。物体に階層構造で子供となる物体（カメラ）を作っておくと（図1）、視点の運動に自由度を持たせることができます。アームの先に付いたカメラが、物体の動作とは独立して色々な方向を見ることができるようになります。

3Dアクションゲームでは「見せ方」をいろいろ工夫しています。Quakeタイプの一人称視点の3Dアクションシューティングとか、格闘ゲームのような第3者の視点、または自キャラ後方からの視点、などがあります。ボクシングの場合を考えると、選手（自分）の視点、相手選手の視点、選手がよく見えるように移動しているレフリーの視点、離れたところで静止して全体を見ている観客の視点、などが思いつきます。

今回は、「視点の切り替え」、「視点の移動」、といったカメラワークに関するサンプルを詳細に見ていきましょう。以下では、プログラムをいくつかの部分に分けて解説しますが、プログラム全体は付録CD-ROMにファイル名camerawk.j3cとして収録しています。

camerawk.j3cでは、5つの付属ライブラリを使用しています。ここでは、J3W643.tar.gzを展開してできたj3w-643/j3c_script/以下にworkディレクトリを作成して、その中にcamerawk.j3cを置いているものとして解説を行います。

リスト2はcamerawk.j3cの先頭部分で、j3w-643/j3c_script/lib以下にあるライブラリとクラスを使うようにインポート宣言しています。そのあと、クラス間で共有して使われる定数を宣言しています。「library Message」はカメラに送る命令で、メッセージ(32ビット整数)に分かりやすい名前を付けています。送る側と受け取る側で利用するため、libraryとして定数のスコープをグローバルにしています。

Floor(リスト3)は最も簡単なクラスです。床を作った後は、単にプログラムの終了まで静止しています。1辺1000の正方形を色を変えて4枚並べています。正方形の登録にsSolidラ

【リスト2】camerawk.j3c(先頭部分を抜粋)

```
import "../lib/key.j3c";
import "../lib/color.j3c";
import "../lib/ssolid.j3c";
import "../lib/zsolid.j3c";
import "../lib/revolutn.j3c";

library Message {                    // メッセージ
    final int LookUp        = 1;
    final int LookDown      = 2;
    final int LookLeft      = 3;
    final int LookRight     = 4;
    final int GoBehind      = 5;
    final int GoHome        = 6;
    final int Select        = 10;
}

library const {
    final int CAMERA = 0;            // インスタンス番号
    final int Ref    = 1000;         // インスタンス番号
    final int PL1    = 3000;         // インスタンス番号
    final int PL2    = 4000;         // インスタンス番号
}
```

Column

J3Wの内部構造の解説

■コンパイラj3c

3Dアニメーション専用の仮想CPUを実装しているj3wの仕組みを5回にわたって概説してきました。今回は、この連載の主役であるJ3C言語のコンパイラ「j3c」を見ていきます。

Linuxでは、多くの言語のコンパイラやインタプリタが無料で利用できるため、言語処理系をあえて作ろうとする人も少ないと思います。しかし、ゲームなど特定の分野の専用言語を考えるのも面白いものです。コンパイラが高価であった20年近く前には、言語処理系の作成は、アマチュアプログラマにとって一般的なテーマであったように思います。

j3cは、J3C言語のソースファイルを読み込んで、アセンブリ言語のソースを1パスで出力するコンパイラです。そのアセンブリソースを2パスアセンブラ(j3dasm)でアセンブルすることでバイナリ(j3dファイル)が生成されます(図A)。

J3C言語は、LL(1)文法(左結合:L、最左導出:L、1トークン先読み:1)という文法を使っています。LL(1)文法の言語は、構文図式(記事末の「J3C構文リファレンス」を参照)に沿って構文解析するプログラム(再帰下降法と呼ぶ)を組んでいくことができます。j3cは、Pascalや Oberon-2を作ったNiklaus Wirthの名著である「アルゴリズム+データ構造=プログラム」(記事末のResource【1】を参照)という本のサンプルのPL/0という小さなコンパイラと同じような構造になっています。

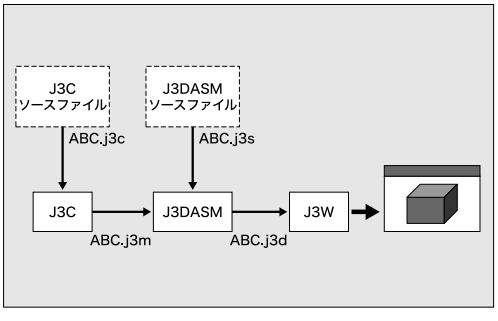
j3cはC++で書かれています。j3cのクラスの大まかな関連を図Bに示します。J3C言語の構文解析は「cParse::parse(**入力ファイル**, **出力ファイル**)」が起点となっています。ソースの読み込みは、cScanクラスが担当しています。数値や予約語、クラスや変数の名前を認識して、cParseクラスが構文解析する単位となるトークン(単語)を返しています。

構文解析に必要とされる変数や定数の名前を管理するシンボルテーブルは4つのクラスで構成されていて、j3cのソースでは最も複雑なパートなっています。クラスの継承、局所変数、ライブラリ、クラスのデータを扱うときに「名前の有効範囲(識別子のスコープ)」を適切に管理するためです。この部分は、シンボルの管理に関する情報を十分に見つけられなかったため、無駄な処理をしているところがあるかもしれません。j3cを開発する上でJ3C言語のシンボル管理が最も苦労した部分です。

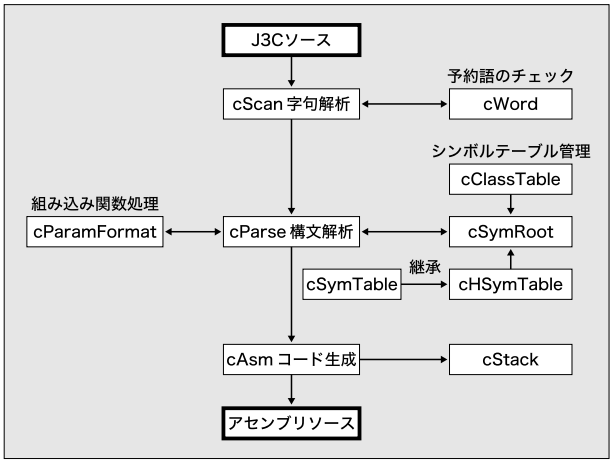
コンピュータ言語の文法は十分に研究されていて、LL(1)文法より柔軟な文法を定義できるLALR(1)文法を解析するプログラムは、yaccやbisonといったコンパイラコンパイラ(コンパイラを作成するプログラム)が、文法を定義するファイルから自動的に作ってくれます。しかし、生成されたプログラムの動きを理解することは困難です。私は人間的な構造のプログラムになる再帰下降構文解析法が好きです。

3Dアニメーション専用の「便利な」高級言語を思い付かなくて、まず「万能な」アセンブラ形式のJ3Wを公開しました。3年近く後になって、J3C言語を追加しました。もっと良いアイデアがあれば、J3W用の別の文法を持つ言語に挑戦してみてはどうでしょうか?　(水谷純)

【図A】j3dファイルができるまで



【図B】クラスの関連図



イブラリのHorizonを使っています。Horizonは、原点を中心とした正方形を生成するため、SetOffset組み込み関数で登録される頂点座標をずらしています。SetOffsetの機能は、続いて登録される頂点座標にRX、RY、RZレジスタ変数の値を加えるというものです。結果として、本来登録されるべき頂点座標がレジスタ変数の分だけ「ずれ」ることになります。原点を中心として作成されるはずの正方形を複数並べることができます。

視点となるCameraクラスは形を持っていません(リスト4)。受信したメッセージに従って回転や移動を行います。視点を持っている(See組み込み関数を実行している)ため、移動や回転に伴って「空間の見え方」が変化します。カメラはHeadクラスの局所座標系を移動します。従って、カメラが動かなくてもHeadクラスが移動すると、カメラも運動することになります。

カメラの移動中にメッセージを受信すると動作が止まります。例えば「見上げてから元に戻

【リスト3】camerawk.j3c（Floorクラス）

```
class Floor {                    // 床を生成
    int INIT() {
        NewObject(20, 20);
        ClearRegisters();
        SetPosition();           // 中心は原点
        Priority(10000);         // 表示優先順位を低く
        RX = -500; RY = -500; RZ = 0;
        SetOffset();             // 中心は (-500, -500, 0)
        sSolid.Horizon(Color.SolidTeal, 500);
        RX = -500; RY = 500; RZ = 0;
        SetOffset();             // 中心は (-500, 500, 0)
        sSolid.Horizon(Color.SolidBrown, 500);
        RX = 500; RY = -500; RZ = 0;
        SetOffset();             // 中心は (500, -500, 0)
        sSolid.Horizon(Color.SolidNavy, 500);
        RX = 500; RY = 500; RZ = 0;
        SetOffset();             // 中心は (500, 500, 0)
        sSolid.Horizon(Color.SolidGray, 500);
    }
    int RUN() {
        Wait();
    }
    int EVENT() {}
}
```

【リスト4】camerawk.j3c（Cameraクラス）

```
class Camera {                   // カメラ
    volatile int type, behind;
    final int rate = 1000;       // 回転時間
    int INIT() {
        RQ = const.CAMERA + RX;
        behind = 0;
        ClearRegisters();
        SetPosition();
        See();
    }
    int RUN() {
        SetCommon(1, RQ);        // 割り込み禁止
        ClearRegisters();
        switch (type) {
            case Message.LookUp :
                RP = 360;        // 上45度
                Move(rate);
                Pause(200);      // 0.2秒停止
                RP = -360;       // 下45度
                Move(rate);
                break;
            case Message.LookDown :
                RP = -360;       // 下45度
                Move(rate);
                Pause(200);      // 0.2秒停止
                RP = 360;        // 上45度
                Move(rate);
                break;
            case Message.LookRight :
                RH = 360;        // 右45度
                Move(rate);
                Pause(200);      // 0.2秒停止
                RH = -360;       // 左45度
                Move(rate);
                break;
            case Message.LookLeft :
                RH = -360;       // 左45度
                Move(rate);
                Pause(200);      // 0.2秒停止
                RH = 360;        // 右45度
                Move(rate);
                break;
            case Message.GoBehind :
                if (behind == 0) {
                    RX = -400;   // 後ろ
                    Move(rate);
                    RX = 0;
                    RZ = -150;   // 上
                    Move(rate);
                    behind = 1;
                } else {
                    RX = 400;    // 前
                    Move(rate);
                    RX = 0;
                    RZ = 150;    // 下
                    Move(rate);
                    behind = 0;
                };
                break;
            case Message.GoHome :
                ClearRegisters();
                SetPosition();
                behind = 0;
                break;
            case Message.Select :
                See();
                break;
        }
        type =0;
        SetCommon(0, RQ);        // 割り込み許可
    }
    int EVENT() {
        type = RL;
    }
}
```

る」という一連の動作を中断させないため、共有メモリ領域のインスタンス番号の位置に、割り込み禁止フラグとして「1」を書き込んでいます。動作が終了したら、「0」を書き込んでメッセージが受信可能であることをKeyInクラスに伝えています。

リスト5は、2人の選手とレフリーのクラスです。体と頭の2つの部分からできています。

体の部分はPlayerクラスと、Playerクラスを継承したPlayer2とRefereeクラスの3つがあり、色と動きが異なっています。Z軸に沿って形状を定義するzSolidライブラリのBar881メソッドを使っています。連結した八角柱2つと八角錐を生成しますが、SetScale組み込み関数を使ってX軸方向を半分に圧縮することで、身体らしい扁平な形状としています。Playerクラス（とサブクラス）は、Headクラスを階層構造の下位に生成しています。Headクラスで生成される頭は、単に回転体を使った球で表しています。Headクラスは階層の下位にCameraクラスを生成しています。

Playerクラスの動きは、中心（床）から距離100だけ離れて、カニ歩きで回るものです。円周の距離を横に進むあいだに、360度ヘッド回転することで実現できます。ここでは1秒間に円周の1/4の距離を90度向きを変えて運動します。90度回転するたびに前進してちょっと横を見て元の位置に戻る、という動きを加えて変化をつけています。Player2クラスは、Playerクラスと対称な位置で同じ動きをしています。Refereeクラスは静止しています。

KeyInクラス（リスト6）は、キー入力されたときに視点を持っているインスタンス（Cameraクラスのインスタンス）に、キー入力に対応するメッセージを送信します。そのために、選択されたCameraクラスのインスタンス番号をcam変数に保持しています。1キー、2キー、3キーのどのキーが入力されたかでインスタンス番号を切り替えています。また今回のサンプルでは、いったんメッセージを受け取ったら「見上げる」、「左を向く」動作のあと、「元の位置に戻る」動作をします。横を向いている間に次のメッセージを受信して動作を中断することがないように、Cameraクラスのインスタンス番号と同じ共有メモリのアドレスを、「メッセージ送信禁止フラグ」として利用しています。カメラ側でメッセージを受け取る準備ができた場合に禁止フラグを「0」にします。KeyInクラスは、禁止フラグが「0」の場合だけメッセージを送信しています。メッセージを受信する側は、メッセージを受け取ると、必ず動作中の命令（移動、回転、停止）を終了して次の命令に移ってしまいます。そのため、都合の悪いタイミングでメッセージを送らないように送信側で

【リスト5】camerawk.j3c（2人の選手とレフリーのクラス）

```
class Head extends Revolution {     // 頭
    int INIT() {
        PushRegisters();
        sphere(30, 16, 16, RX);
        PopRegisters();
        child(Camera, 100, 0, 0);   // カメラを生成
        ClearRegisters();
        RZ = -170;
        SetPosition();
    }
}

class Player {                      // 選手1
    int Initialize(int color) {
        NewObject(100, 100);        // 物体生成
        zSolid.Bar881(60, 40, 40, 40, 30, 50, color);
        RX = 500; RY = 1000; RZ = 1000;
        SetScale();
        RX = color;
        child(Head, 100, 300, 300);
        ClearRegisters();
    }
    int INIT() {
        RQ = const.PL1;
        Initialize(Color.Green);
        RX = 200; RH = 1440;
        SetPosition();              // 位置設定
    }
    int RUN(){
        ClearRegisters();
        RY = 314; RH = -720;        // 周回運動
        Move(2000);
        RX = 150;                   // 前進
        Move(500);
        RotHead(500, 80);
        RotHead(500, -80);
        RX = -150;                  // 後退
        Move(500);
    }
    int EVENT() { }
}

class Player2 extends Player {      // 選手2
    int INIT() {
        RQ = const.PL1;
        Initialize(Color.Red);
        RX = -200;
        SetPosition();              // 位置設定
    }
}

class Referee extends Player {      // レフリー
    int INIT() {
        RQ = const.Ref;
        Initialize(Color.Cream);
        ClearRegisters();
        RX = -500;                  // 初期位置
        SetPosition();              // 位置設定
    }
    int RUN(){
        Wait();
    }
}
```

【リスト6】camerawk.j3c（KeyInクラス）

```
class KeyIn {
   volatile int key, cam;
   volatile int InteruptMask;
   int INIT() {
      cam = const.CAMERA + Color.Cream;
   }
   int RUN() {
      key = InKey();          // キー入力
      // 共有メモリを調べて割り込みOKならメッセージ送信
      InteruptMask = GetCommon(cam);
      if (InteruptMask == 0) {
         switch (key) {
            case Key.CursorLeft :  // [←]
               Send(cam, Message.LookLeft);
               break;
            case Key.CursorRight :// [→]
               Send(cam, Message.LookRight);
               break;
            case Key.CursorUp : // [↑] 見上げる
               Send(cam, Message.LookUp);
               break;
            case Key.CursorDown :  // [↓] 見下ろす
               Send(cam, Message.LookDown);
               break;
            case Key.B :         // [B]  背後視点
               Send(cam, Message.GoBehind);
               break;
            case Key.H :         // [H]
               Send(cam, Message.GoHome);
               break;
            case Key.One :    // [1]  選手1の視点選択
               cam = const.CAMERA + Color.Green;
               Send(cam, Message.Select);
               break;
            case Key.Two :    // [2]  選手2の視点選択
               cam = const.CAMERA + Color.Red;
               Send(cam, Message.Select);
               break;
            case Key.Three :     // [3]  レフリーの視点選択
               cam = const.CAMERA + Color.Cream;
               Send(cam, Message.Select);
               break;
            case Key.Esc :    // [Esc]
               Stop();        // ESCキーで終了
               break;
         }
      }
   }
   int EVENT() {}
}
```

チェックするようにしています。

J3Cでは、常にmainクラスが最初に実行されるクラスとなっています。リスト7のmainクラスでは「グラフィックウインドウを開いて、背景の色を設定、視野角（ズーム）の指定」といった初期設定をした後で、これまで見てきたクラスを順次起動しています。Floorは、床、Refree、Player、Player2という登場人物、キー入力からプログラムの動作を伝えるKeyInクラスといったインスタンスを生成して、後はプログラムの終了を待ち続けます。

では実際に実行してみます。j3w643.tar.gzが展開済みの場合には、j3w-643/j3c_scriptに移動して、付録CD-ROMのsample.tar.gzを展開します。j3w-643/j3c_script/workに移動して、以下のように入力して実行してみましょう（画面2）。

```
j3cc -r camerawk.j3c
```

まずレフリーの視点です。Bキーを入力すると、「後ろへ下がって、上に移動」した視点となります。もう一度Bキーを入力すると、「前進して、下に移動」して元の位置に戻ります。カーソルキーでは、ゆっくり矢印の方向を見て、元に戻ります。1キーでは選手1の視点、2キーでは選手2、3キーでレフリーの視点に切り替わります。

今回のサンプルでは、頭（Headクラス）は静止したままでした。頭にも色々な動きをつけてみてください。頭は、体(Player、Player2、Refree）クラスの局所座標系で運動します。

【リスト7】camerawk.j3c（mainクラス）

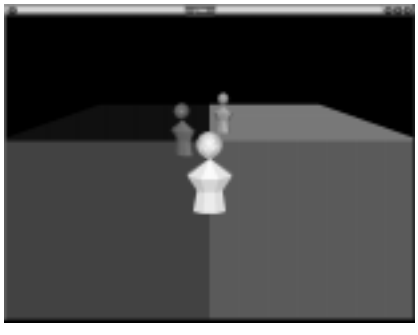
```
class main {                    // メイン
    int INIT() {
        GraphMode();
        BackgroundColor(0x0);
        Zoom(0);
        ClearRegisters();
        new(Floor,   100);      // 床
        new(Player,  100);      // 選手1
        new(Player2, 100);      // 選手2
        new(Referee, 100);      // レフリー
        new(KeyIn,   100);      // キー入力クラス
    }
    int RUN() {
        Wait();
    }
    int EVENT() {}
}
```

■終わりに

3Dアニメーションを眺めるだけではなく、視点の切り替えと移動だけでしたが、少し参加できるプログラムになってきました。

物体を設計する場合に、形状の定義をしながら自分の目で見て確認したくなります。作成中のプログラムの動作とは別に、物体の形を個別に確認するためのツールとしてobjview.j3cが用意されています。次回は、このobjview.j3cの使い方を解説する予定です。

【画面2】camerawk.j3cの実行例



Resource

[1] アルゴリズム＋データ構造＝プログラム

Niklaus Wirth著／片山卓也訳／科学技術出版

原書：Algorithms + Data Structures = Programs, Prentice-Hall, 1976.

J3C 構文リファレンス ①

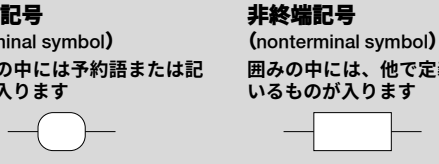
J3Cは、3Dアニメーションキット J3Wのコンパイラです。J3Cは、Java風の文法を持つオブジェクト指向言語として設計されています。今月から3回にわたって、J3Cの構文リファレンスを掲載していきます。

構文図式の見方

■構文規則（syntactic rule）

コンピュータ言語の文法は、構文規則によって決定されます。構文は、左から右へ、上から下へ矢印をたどって読んでいきます。

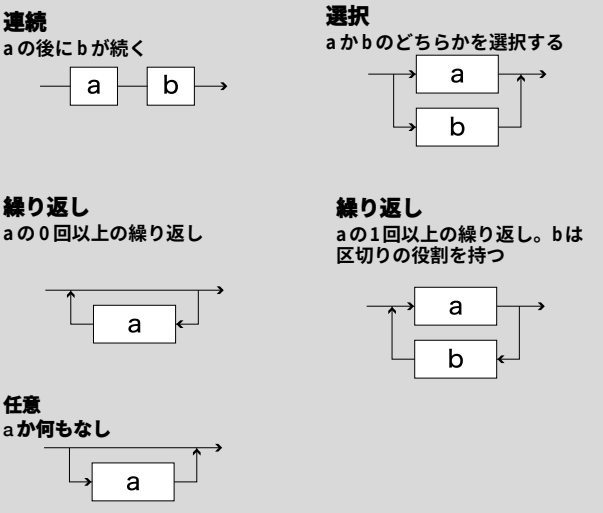
構文規則の左側にある名前のことを、「非終端記号(nonterminal symbol)」と呼びます。また、classやforなどの予約語、＝や＋などの記号で表される単語そのもののことを「終端記号(terminal symbol)」と呼びます。終端記号と非終端記号は、構文図では次のように表します。



構文規則を順次適用していくと、すべての非終端記号は終端記号に行き着きます。

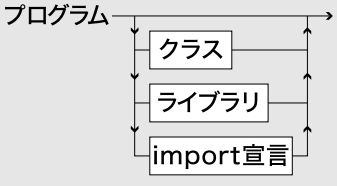
■構文図式の基本形

構文図式の基本形は次のようになっています。「連続」、「選択」、「繰り返し」を組み合わせて文法を記述します。



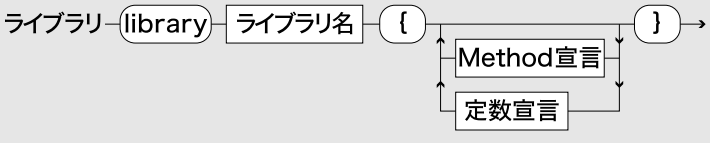
プログラム

J3Cのプログラムは、「クラスの定義」、「ライブラリの定義」、「外部ファイルの挿入（import宣言）」の繰り返しから構成されます。



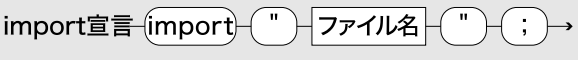
ライブラリ

ライブラリは、クラスと異なりインスタンス化することはありません。スコープはグローバルで複数のクラスからライブラリに属するメソッドと定数を参照することができます。クラスのようにデータメンバ（クラス変数）を持つことはできませんが、定数を宣言することはできます。



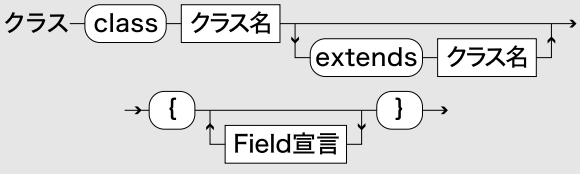
import宣言

importは、別に保存されているソースファイルを指定位置に挿入（インクルード）します。インクルードされたファイルはそこでコンパイルされます。同一ファイルが複数回インクルードされた場合、2度目からは無視されます。



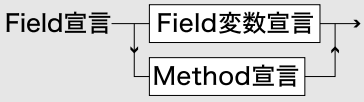
クラス

クラスは、インスタンス化されると「スレッド」として独立して動作します。クラスは独立しているため、すべてのデータメンバとメソッドのスコープはプライベートであり、他のクラスから参照することはできません。extends節で宣言済みのクラスを指定した場合、extends節で指定したクラスからデータメンバとメソッドを継承します。



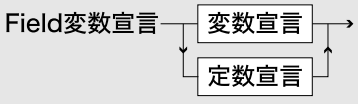
Field宣言

クラスにはメソッド関数の宣言と変数または定数の宣言を置きます。



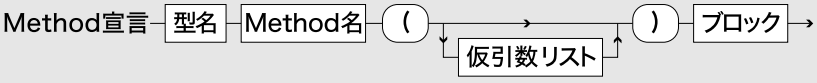
Field変数宣言

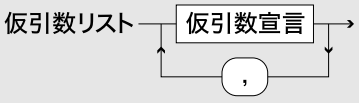
クラスに属する変数、または定数を宣言します。

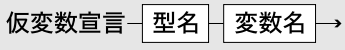


Method宣言／仮引数リスト／仮引数宣言

クラス内で使用する関数を定義します。型名はintのみです。メソッドは使用前に定義されている必要があります。すべてのクラスには`INIT`、`RUN`、`EVENT`メソッドが必須（継承も可）です。



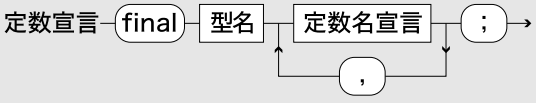


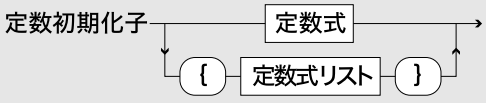


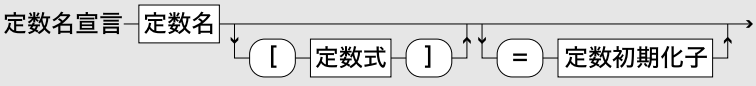
定数宣言

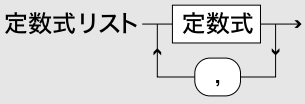
変数は、`final`宣言されたもの（定数）のみ初期化が可能です。変数の配列は、一次元配列のみ宣言することができます。配列を利用する場合、Javaと異なり、配列の大きさを指定する必要があります。

`final`宣言された定数は、定数式中に使用することができます。なお、初期化された配列の参照時のインデックスに変数は使用できません。





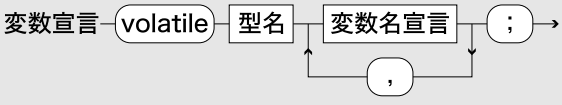


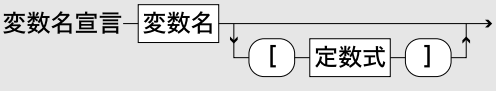


変数宣言

変数の型は`int`型（32ビット整数）のみです。値を変更する通常の変数は、前に「`volatile`」を置きます。変数は初期化できません。

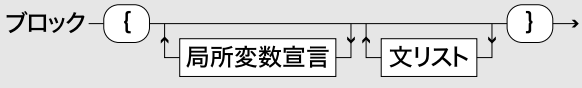
変数は配列とすることもできます。定数式の部分には配列の要素数を指定します。配列の添字は必ず「0」から始まります。





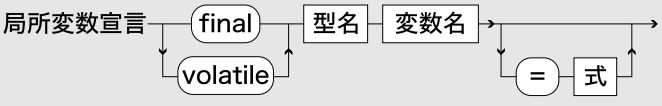
ブロック

「{」と「}」で囲まれた部分は「ブロック」と呼ばれ、ブロック内だけで有効な変数、定数を宣言できます。変数、定数の宣言は文より前に置く必要があります。



局所変数宣言

ブロックの最後まで有効な局所変数を宣言できます。型はintのみです。ブロック内で文リストの前に宣言します。局所変数も定数のみが初期化できます。



文リスト

文と文の間はセミコロンで区切ります。セミコロンは空文（何もしない文）または、文と文の区切りとして作用します。従って、ブロックの最後の文にセミコロンを付ける必要はありません。