目次

第 1 章

はじめに

この TJS 活用講座は元々私のブログでひっそりと更新されていた TJS 講座もどきなるものの延長線上にあるものです。正直な話、“ひっそりと更新されていた” という部分は “殆ど更新されていなかった” にほぼ等しいかもしれません。もうあれは忘れてください（＾＾；

対象としているのは主に KAG から TJS へ進んできた人です。一応 KAG をスキップして TJS を吉里吉里でどうやって使うのか興味がある人も大丈夫だと思いますが。

さて、延長線上とは言いましたが TJS 講座もどきで扱った内容も含んでいます。大幅に修正がされていますが。流石にブログの内容をそのままそっくりコピペしてきたわけではありません。因みに今回こういう体裁で文章を書いているのでいつもよりも真面目度が上がっています。表面的にはちょっと堅苦しく見えるかもしれませんが、前の段落で顔文字を使っていたように完璧真面目モードではありやせんぜ兄貴（ぁ

直前の妄言は忘れましょう。兎にも角にもブログの文章よりは真面目に書きます。

私事ですがこの文章は LaTeX を使って書いています。私が LaTeX の扱いに慣れてそこそこ熟練しようという狙いも含みつつお送りします（笑

また、こういう体裁をとっているのでプリントアウトして読んでもかなり読みやすいはずです。ウィンドウの切り替えが煩わしい人等は紙媒体でどうぞ。

1.1 まえがき

ここでこうして教える側にいる私自身、プログラムのプの字も知らないところから TJS を学びました。参考書は吉里吉里付属のリファレンスとネット上のいくつかのサイト、そして TJS を自由自在に使いこなしていた方々のサンプルスクリプトでした。吉里吉里を使い始めたときは KAG も CUI [*1]ゆえにしばしば閉口していましたが、TJS はそれ以上に難しくて挫折しかけました。不安を煽っているわけではありません。ただそれなりの努力を要するということを言いたいのです。他にプログラム言語を齧ったことがあるなら大いに理解の助けとなります。しかし、これを読んでいる人は十中八九そのアドバンテージを持っていないことと思います。そういった人達の理解の一助となれれば嬉しいです。話は逸れますが、他の人の書いたスクリプトはとても参考になります。余裕があればできるだけ参考にしてみて下さい。

1.2 ここで TJS を学ぶ前に

ここで KAG の記述についての説明は基本的にしません。したがって、サンプルとして TJS の記述がされたからといって ks ファイルにそのままコピペして動かないとぼやかないで下さい。ks ファイル内で TJS を記述するとき `[iscript]`〜`[endscript]` を使うことは常識として書いていきます。また TJS の文法については適宜リファレンス等から引用します。ですから、文法のことを説明する章はありません。英語等の言語でもそうですが、文法はやっていて

[*1] Character-based User Interface 要するにキーボードで主な操作をするユーザインターフェース

楽しくないですよね。大切ですけど。そういうわけで必要なときに必要なことを覚えるという方針でいきたいと考えています。

1.3　かまと

絵や図等はちょこちょこと入れていきます。文字ばかりだと嫌になってしまうでしょうから。そういう意味でキャラクターを一人用意しました。余白で何かしていたり、文の合間に出現したりするかもしれません。名前はかまとです。[*2]かま子とでもお呼び下さい。

[*2] キーボードのキーを T・J・S の順にご覧下さい

第 2 章

変数

すべての基礎、変数の話です。大抵プログラム等の講座では一番簡単なものとして最初に Hellow World をやりますが、ここではやりません。文法的なこと等を Hellow World から学べますが、先に述べたように文法の説明は必要に応じて挿入しますし、既に KAG を使っている状況で吉里吉里の起動方法等説明しても釈迦に説法でしょう。変数と言って何のことだか全く分からない人もいないですよね。KAG でも `f.〜`や`sf.〜`を使っていたことでしょう。もしかしたら `mp.〜`を使っていた人もいるかもしれません。なので根本的な変数の説明もスキップします。だたし、ここはどうしても文法的な事項に重きが置かれてしまいます。最初だけ我慢してください。

2.1 TJS での変数

TJS で扱う変数は KAG で使っていた変数とは少し違います。もちろん TJS で使っていた変数を KAG から使うことはできますし、逆もまた然りです。そういう意味での違いではありません。KAG である変数を初めて使うときに変数を宣言した覚えはありますか？普通の使い方をしていた人なら答えは NO のはずです。そもそも宣言って何さ？という人もいておかしくありません。そのあたりから話を始めたいと思います。

本来変数は使いたいなと思ってすぐその場で使えるものではありません。まず使いたい変数を準備しておく必要があります。KAG ではそんな必要が無かったのにどうしてでしょう？その答えはこの先学ぶことになる**配列**で明らかになります。現時点では準備をしなくてはならないものと割り切るしかありません。では、なぜ準備をする必要があるのかという話になります。変数は PC のメモリ上のある場所に名前をつけてその場所に値を入れておく仕組みです。なので変数を宣言してメモリのある場所を確保しなくてはなりません。

2.2 変数の宣言・初期化

では早速変数を宣言して使えるようにしてみましょう。

```
var elem;
```

この一行で `elem` という名前の変数を宣言しました。一行だけですが文法面で学ぶことがいくつかあります。

1. 一つの処理の最後にはセミコロン (;) が必要。
2. 意味が分からなくならない程度のホワイトスペース[*1]を入れることができる。
   逆に、意味が分かるように適宜挿入する必要がある。
3. 変数の名前は予約語以外でなくてはならない。
   ただし、数字は先頭にくることができない。

[*1] 改行やスペース、タブ等のこと

3 について補足すると予約語とは以下のもの達のことです。

```
break continue const catch class case
debugger default delete do extends export
enum else function finally false for
global getter goto incontextof Infinity
invalidate instanceof isvalid import int in
if NaN null new octet protected property
private public return real synchronized switch
static setter string super typeof throw
this true try void var while with
```

これらの予約語はスクリプトの文法を構成する重要な用途として特殊な意味を持っています。ただし、予約語のほとんどはがんばって覚える必要はありません。なぜなら、これらの予約語は TJS を学んでいけば特殊な意味を持った文字列達だと分かるからです。例えば変数の宣言で使った var という文字列は

```
var ○○○;
```

のように書くことで○○○という名前の変数を宣言するという特別な意味を持ちます。このように予約語と呼ばれる文字列は TJS で何か処理をさせる命令という風なものなのです。だからその内に覚えてしまっているというわけです。

因みに変数の名前に全角文字を使うこともできるにはできるのですが一般的ではありません。俺はマイナー道を突き進むぜ！という人を無理に止めはしませんが、正直に言うと人に見られたとき非常に恥ずかしい思いをします。基本的にコメント[*2]以外は半角文字を使いましょう。できれば英語の方が好ましいです。言い忘れるところでしたが、アルファベットの大文字と小文字は区別されます。

さて、冒頭の一行で変数は宣言されました。これで KAG でやっていたように変数を使った加減乗除等の演算ができるようになりました。この段階では変数 elem には何も値が入っていない状態です。言い換えると、この状態での elem のデータ型は void と言って何も表していないことを表しています。いきなりデータ型という単語を引っ張り出してきて混乱した人もいるかもしれませんね。変数は扱う値の種類によっていくつかの “ 型 ” があります。(整数型、実数型、文字列型...etc) しかし、TJS では変数の “ 型 ” が外部的にはありません。C 言語等の他の言語では変数を宣言する際にそれぞれデータ型を指定してやる必要があるのですが、TJS では指定する必要がないのです。しかし、TJS も内部的には型を持っています。(よって型を持たないと言うよりは、型が自動的/動的に扱われると言った方が正確ですね。これは TJS の長所の一つと言えると思います）その内部の型に void、整数型、実数型、文字列型、オブジェクト型[*3]、オクテット型[*4]があります。そして宣言したての変数は void です。

ただし、宣言と同時に何か値を変数に与えてやった場合はこの限りではありません。

```
var elem = 10;
```

これを初期化と言います。この場合、elem には既に 10 という整数が入っています。ここで文法事項の 2 について少し説明しておきます。意味が分からなくならない程度のホワイトスペースとはどういうことでしょう？例として意味が分からない程度にホワイトスペースを入れてみましょう。まず一つ目としては

```
var
elem
=
10
;
```

[*2] //から行末までの部分、もしくは/*から*/で挟まれている部分。
このコメントの部分は TJS に解釈されないので自由にメモを残しておくといいです。
//についてはは KAG の; に似ていますね。

[*3] ここでは気にしなくて OK です。参考までに。

[*4] 同じく参考までに書いたまでです。流して下さい。

というものです。読み辛いことこの上ありません。止めてください（＾＾；

```
v ar e le m = 1 0 ;
```

これが二つ目の例です。勿論これは論外です。正常な解釈ができません。つまるところ、こんなことにならないようにホワイトスペースを入れましょうねってことです。また、意味が分かるように適宜挿入するというのはどいういうことでしょうか。再びやってはいけない例を示します。反面教師にして下さい。

```
varelem=10;
```

最早解説の必要はないでしょうが一応。var が変数を宣言するための特別な文字列です。しかしその後にスペースを設けずに elem と入力してあります。これでは var と解釈されずに varelem として解釈されます。意図した動作はしてくれません。

◆補足◆　数値と文字列

因みに記述する我々としては整数型と実数型はあまり意識する必要がありません。一番気にすることになるのは整数型/実数型と文字列型です。これは KAG で経験済みかもしれませんが一応説明しておきます。数字の 1 に数字の 2 を足すといくつになるでしょう。もちろん 3 ですね。この 3 は数字です。では、文字の 1 に文字の 2 を足すといくつになるでしょう。答えは 12 です。この 12 は文字です。つまり、こういうことです。3 を期待していたのに結果として 12 が得られてしまう、というような困ったことになりますね。こういう問題は必要に応じて文字→数値の変換をしてやることで解決できます。単項演算子 “+” の出番です。加減算演算子の “+” と混同すると危険かもしれません。加減算演算子は左側のものに右側のものを足す、という処理をします。要するに小学生で習う足し算をするんですね。そして単項演算子は右側のものを文字列から数値に変換しようとがんばります。

```
var elem = "10";
var num = +elem;
```

上の 2 行ではまず elem に文字の 10 が与えられ、num には elem を数値に変換して代入されています。文法的なことで一つ確認しておくことが。それは文字列は"と"で囲むということです。KAG を普通に使っていれば学習済みかと思いますが念のため。

2.3　２重宣言

同じ名前の変数を 2 回以上宣言してもエラーが起こることはありません。ただし、宣言された名前の変数は void となります。初期値を指定していればその値になります。スクリプトが大規模化すると無意識のうちに同じ名前の変数を宣言してしまうことが起こりえます。そうするとエラーが出ないため予期せぬ挙動を示すことになりますね。気をつけましょう。

2.4　変数のスコープ

まずスコープの説明の前にブロックというものについて説明の必要があります。ブロックとは{と}で囲まれた部分のことをいいます。

```
var flag = 1;
if(flag == 1)
{
        var elem;
}
```

上のスクリプトで言うと変数 elem が宣言されている部分です。ところで唐突に出現した if(flag == 1) とは何のことでしょう。お察しの通り条件分岐をしています。この==は KAG の if タグで使っていた==と同じものです。TJS ではこのように条件分岐を行います。条件式を評価して真なら後続ブロックを実行し、偽なら後続ブロックを無視します。

ではスコープの説明に入っていきます。上の例では if の後続ブロック内で elem が宣言されています。実はこの elem は if の後続ブロックから出ると消えてしまいます。つまり、この elem はこのブロック内でのみ有効な変数[*5]なのです。ところで、以下の場合を考えてみてください。

```
var flag = 1;
if(flag == 1)
{
        var flag = 2;
}
var elem = flag + 1;
```

elem には何が入っていると思いますか？答えは 2 です。えー (; ´ Д `) と思った人はいますかね。恐らくいると思います。何せ変数の 2 重宣言を説明した直後ですからね。ブロック内で宣言された flag はローカル変数なのでブロックの中でのみ有効です。なのでブロックの外で宣言された flag とは全く別の変数なのです。待てよ、と思った人はいますか？そうです。これではブロックの外で宣言した flag にアクセスできませんね。でもそんな不便な仕様にはなっていません。一番外側で宣言された変数[*6]には global.〜でブロックの中からでもアクセスできます。

```
var flag = 1;
if(flag == 1)
{
        var flag = 2;
        global.flag = flag;
}
var elem = flag + 1;
```

この場合には elem には 3 が入ります。

◆補足◆　単項演算子 “++”

最後のスクリプトの最終行 var elem = flag + 1; は flag に 1 だけ足したものを elem に入れているため var elem = ++flag; と書くことができます。単項演算子 “++” を使いました。便利な演算子なので是非使えるようになりましょう。ただし、注意することがあります。足すものの前につけるか後ろにつけるかで意味が変わってしまいます。例えば

```
var flag = 1;
var elem = flag++;

flag = 1;
var tmp = ++flag;
```

というスクリプトがあったとき、elem と tmp にはそれぞれ違う値が入っているのです。elem は 1 で tmp は 2 です。それではいい加減 “++” について詳しく説明しましょう。そろそろイライラしてきたはずです（笑；

まずは flag++ について。++ が後置された場合、左にあるものに 1 を加算して、全体としては加算する前の値 (左にあるものそのまま) になります。すなわち、elem には加算前の flag の値が入って、flag には 1 が足されるということです。

次は ++flag についてですね。++ が前置された場合、右にあるものに 1 を加算して、全体としては加算した後の値になります。前置の方が分かりやすいですね。flag は 1 が足されて、その 1 足された flag の値が elem に入るということになります。

蛇足かもしれませんが、“++” を使わずに上のスクリプトを書き換えてみます。

```
var flag = 1;
var elem = flag;
flag = flag + 1;

flag = 1;
flag = flag + 1;
var tmp = flag;
```

見比べてみて下さい。flag = flag + 1; の部分は “++” 以外で書き換え可能ですが、それはまた別の機会に。

[*5] これをローカル変数と言います。

[*6] こちらはグローバル変数と言います。

第 3 章

文字の表示

自分の書いたスクリプトで目に見える処理がされると嬉しくなります。特に意図した通りに動いたときには。前章で変数を通して TJS の基礎は学習しました。ここから先、吉里吉里に目に見える処理をしてもらいます。変数を宣言してもディスプレイの前の我々には何も変化していないように見えますね。当然のことですがそれでは面白くありません。まずは前章の内容も踏まえて、変数の中身を吉里吉里に示してもらいましょう。

3.1 メッセージボックス

一番簡単な方法はメッセージボックスを使って変数の中身を表示することです。早速例を示します。

```
var text = "かまと";
System.inform(text);
```

吉里吉里で実行してみて下さい。“かまと” と書かれたメッセージボックスが開きましたね。2 行目が謎なオーラを放っていますがまだ分からなくていいんです。`System.inform(〜);` で〜の内容をメッセージボックスに表示する、という覚え方で構いません。これをちゃんと理解するには**関数**や**クラス/オブジェクト**の知識が必要です。

とにかくこれで変数の内容が表示されてわかるようになりました。前章に戻って例として書いてあるスクリプトを試してみるのもいいと思います。ちょっとだけ応用の例も書いておきます。

```
var text = "かまと";
var item = "バナナ";
System.inform(text + "は" + item + "を手に入れた。");
```

もし分かりにくかったら `text + "は" + item + "を手に入れた。"` の部分を変数で置き換えてみると見やすくなります。

```
var text = "かまと";
var item = "バナナ";
var info = text + "は" + item + "を手に入れた。";
System.inform(info);
```

慣れないうちはこの方がいいかもしれませんね。

3.2 KAG の画面内に描画

吉里吉里の画面と言ったほうがよくわかるかもしれませんが、一応厳密に KAG の画面と書きました。ただの表現の違いというくらい微妙なことだと思うのであまり深く考えません（笑

3.2.1　メッセージレイヤへの描画

勿論 KAG でできることなのでわざわざ TJS を使う必要はありません。ただ、TJS で書いている途中に（例えば KAG プラグイン等）KAG は使いません。できないことではありませんが、普通はそのままの流れで TJS で処理を書きます。では先程のかま子がバナナを入手したことについてメッセージレイヤに描画してみましょう。

```
var text = "かまと";
var item = "バナナ";
var info = text + "は" + item + "を手に入れた。";
kag.fore.messages[0].drawText(0, 0, info, 0xffffff);
kag.fore.messages[0].visible = true;
```

当然ながら変数を宣言するところまでは前のメッセージボックスに表示したときと同じです。4 行目では、表画面メッセージレイヤ 0 の x=0、y=0 の位置に info の内容を白文字で描画しており、次の行で可視状態にしています。折角描画しても不可視状態のままでは見えませんからね。kag.fore.messages[0] の部分を見て、KAG に慣れ親しんでいる皆さんはどのレイヤに描画しようとしているのか、については最早説明を要しないことと思います。裏画面のメッセージレイヤ 1 に描画したければ

```
var text = "かまと";
var item = "バナナ";
var info = text + "は" + item + "を手に入れた。";
kag.back.messages[1].drawText(0, 0, info, 0xffffff);
kag.back.messages[1].visible = true;
```

としてやればいいのです。[endscript] の後でトランジションでもやれば確認できることでしょう。ところで、drawText の後ろのカッコの中身は一体なんでしょう。先述の説明で何となく勘付いている人もいることと思いますが説明します。コンマ “,” で区切られているので便宜的に前から a,b,c,d とします。a は文字を描画する x 座標、b は描画する y 座標、c は描画する文字列、d が描画する色となっています。なので info の内容を表画面メッセージレイヤ 1 の x=40、y=100 の位置に赤文字で描画したければ

```
var text = "かまと";
var item = "バナナ";
var info = text + "は" + item + "を手に入れた。";
kag.back.messages[1].drawText(40, 100, info, 0xff0000);
kag.back.messages[1].visible = true;
```

という具合になります。

◆補足◆　少し掘り下げた drawText の使い方

吉里吉里 2 リファレンスでクラスリファレンスの Layer クラスからメソッド drawText(文字描画) という項目の内容を見てみるとアレ？と思うかもしれません。実際にご覧下さい。構文のところに書かれているものが妙に長たらしいですね。ここで使ったものとは偶然名前が同じだけで別物…というわけではありません。実は今回使った drawText は color 以降の部分を省略したのです。ですから最後の shadowsfy のところまで値を書いても OK です。ただし、注意があります。opa=255 や aa=true 等と書いてあるからといって

```
kag.fore.messages[0].drawText
(
        0,
        10,
        "てすと",
        0xffffff,
        opa=200,
        aa=true,
        shadowlevel=100,
        shadowcolor=0x505050,
        shadowwidth=2,
        shadowofsx=2,
        shadowofsy=4
);
```

と書かないようにして下さい。(長くなってしまい 1 行に収まらないため、カッコの中を 1 つのブロックのように見える書き方をしました。このようなブロックのように見える書き方をするな、と言っているわけではありません。勘違いしないように。) TJS は KAG と違ってマークアップ言語[*1]ではありません。正しくは

```
kag.fore.messages[0].drawText
(
        0,
        10,
        "てすと",
        0xffffff,
        200,
        true,
        100,
        0x505050,
        2,
        2,
        4
);
```

という風になります。では、何故 opa=255 のような表記がされているのでしょう。それは省略した場合赤字で書いてある値を指定したことになる、ということを意味しています。そして、値のみをコンマで区切って書いていくため順番が大切になります。因みにここで出てきたクラス・メソッドという言葉はまだ理解していなくていいのであまり気にしないで下さい。この章で覚えるべきは “どうすれば TJS を使って文字の描画 (表示) ができるのか” です。大量のことを一気に覚えさせようとはしませんので整理しながら進んでいって下さい。でもときには振り返ることも必要です（ぁ

3.2.2　その他レイヤへの描画

まずはメッセージレイヤと同様に前景レイヤに文字を描画しましょう。実はこの時点でできてしまう人もいるかもしれません。例えば表画面の前景レイヤ 0 にかま子がバナナを何たらという文を表示する場合

```
var text = "かまと";
var item = "バナナ";
var info = text + "は" + item + "を手に入れた。";
kag.fore.layers[0].drawText(40, 100, info, 0xff0000);
kag.fore.layers[0].visible = true;
```

という感じになります。そうです、messages が layers に代わっただけです。KAG の名残で messages を message、layers を layer と書かないように気をつけて下さい。そして、表画面の背景レイヤに同様に表示するなら

```
var text = "かまと";
var item = "バナナ";
var info = text + "は" + item + "を手に入れた。";
kag.fore.base.drawText(40, 100, info, 0xff0000);
kag.fore.base.visible = true;
```

非常に簡単ですがこれで終わりです (汗

あらかたのことを先に説明してしまいましたのでこの章はこれで終了です。じゃー次行ってみよー。

[*1] KAG ではタグを使うとき、属性とそれぞれの値を書いていましたがこのような記述をするのはマークアップ言語の場合のみです。

第 4 章

画像の表示

言うまでもなく文字が表示されただけではどうしようもありません。勿論文字さえ表示することができれば画像の代わりに文字を使ってアイディア勝負のゲームを作ることができると言えばできます。ただやっぱり普通にゲームを作るなら画像の読み込みは避けては通れない道です。そこでここでは画像をレイヤに読み込む処理から、できると便利な処理を少しだけ紹介します。あ、言うまでもないと思いますけど何か画像を用意して下さいよ（＾＾；

4.1 レイヤへの読み込み

用意した画像を対象のレイヤに読み込ませましょう。しかし、すでにある KAG のレイヤ[*1]に TJS で記述して読み込ませるのはちょっと面倒です。なので自分でレイヤを用意して、そのレイヤに画像を読み込ませましょう。こっちは簡潔に記述することができます。レイヤを用意するには本来**クラス**や**オブジェクト**といった知識が必要なわけですがここでは詳しく説明しません。後々**クラス**について触れる際にわかることでしょう。では、レイヤを作ってみます。

```
var lay = new Layer(kag, kag.fore.base);
```

完了しました。今のところは “おなじないの一行” で結構です。lay という名前のレイヤを使うことができます。ところで、var が出てきていますね。しかしできたのは変数ではなくレイヤだと言っています、この人は。なんのこっちゃわけわからんわ、となっていますか？先に触れた**オブジェクト**という単語から変数のデータ型の話で出てきた言葉 “オブジェクト型” を思い浮かべた方は勘付いているかもしれませんが、ここで解説することはしません。おまじないなのです。それより画像を読み込みましょう。

```
var lay = new Layer(kag, kag.fore.base);
lay.loadImages("sample.png");
lay.setSizeToImageSize();
lay.visible = true;
```

一行ずつ見ていきます。実際に画像を読み込む処理をしているのは 2 行目だけです。ではそれ以降は何をしているのでしょう。ここは流石におなじないで済ませるわけにはいきません。4 行目は説明無しでも大丈夫かもしれませんね。作ったレイヤ lay を可視状態にしています。折角画像を読み込んでもレイヤが不可視では見えませんからね。3 行目は読んだままですが説明をば。レイヤのサイズを画像のサイズと同じにしています。何故こんなことをしなくてはならないのか。

> 画像サイズがレイヤの表示サイズより小さかった場合を 除いて、レイヤの表示サイズは変更しません。（吉里吉里 2 リファレンスより）

[*1] 例えば前景レイヤ等

要するにレイヤのサイズが小さいのにそれより大きい画像を読み込むとレイヤのサイズ分しか表示されないため、です。一度 3 行目をコメントアウト[*2]して実行してみると違いがわかると思います。そういうことなので、基本的には 2 行目と 3 行目はほぼセットで記述します。

◆補足◆　with ステートメント

実は 2 行目から 4 行目については省略した書き方ができます。それには with を使います。別にあのままでいいんですが、記述するときに何度も lay と打つのは面倒じゃありませんか？これがたった 3 字なのでそれほど面倒さを感じないかもしれませんが表画面の前景レイヤ 0 だったとき等長い場合は断言します、面倒だと。そして修正するときも面倒で仕方ありません。というわけで、これができなくてもゲームに影響はありませんが効率が変わってくるわけです。では、先にスクリプトを示します。

```
var lay = new Layer(kag, kag.fore.base);
with(lay)
{
        .loadImages("sample.png");
        .setSizeToImageSize();
        .visible = true;
}
```

こんな感じに記述することができます。見た目としてはピリオド（.）の前を省略して書くことができそうですね。仕組みは**オブジェクト**について説明する際に一緒に説明することにしましょう。ここでは見た目のままの使い方で構いません。

さて、KAG のレイヤに TJS で画像を読み込ませるのは面倒だと言いました。実際のところ記述自体にはそれほど面倒なことはありません。**配列**の内容を先取りした内容になってしまいますのでここでは割愛しました。

4.2　レイヤへの操作

レイヤは画像を読み込んだり、文字を表示するだけのものではありません。これは KAG を弄っていた人には既にわかっていることでしょう。TJS を使うことで KAG 標準ではできないレイヤ操作をすることが可能です。ここではその一部を取り扱います。基本的なやり方がわかれば、あとは全部その応用です。

4.2.1　塗りつぶし

まずは肩慣らしです。表画面の背景レイヤの一部を真っ白に塗りつぶしてみます。

```
kag.fore.base.fillRect( 50, 0, 300, 450, 0xffffffff );
```

これで完了です。実際に実行してみると各引数[*3]が何を意味しているかわかると思うのであえてその説明はしません。あ、でも最後のものは見慣れない代物かもしれませんね。一応 0xffffffff だけ説明をしておきます。0xffffff の間違いなんじゃないか？と思った方もいるでしょうが、間違いではありません。[*4]0xffffff は 0xRRGGBB という色の指定に使っていました。一方 0xffffffff は 0xAARRGGBB という形式をとっています。AA とは何のことか…これは透明度のことです。ff が完全不透明、00 が完全透明。もう大丈夫ですね。では表画面の背景レイヤ全部を真っ白に塗るのにはどうすればいいでしょう。

```
kag.fore.base.fillRect( 0 , 0, 640, 480, 0xffffffff );
```

ですね。でもちょっと待って下さい。何気なく 640 や 480 という数値を直に記述しましたが、実はあまりいい書き方ではありません。もし画面サイズを変更した場合、当然レイヤのサイズも変わってきます。何せ背景レイヤですからね。その都度手打ちで修正するのは骨が折れます。なのでレイヤ全体を塗りつぶすのであればレイヤの大きさを取得し

[*2] 3 行目の頭に//を記述するなどしてコメント行にしてやること

[*3] カッコの中の , で区切られたそれぞれの数値や文字列のことです。

[*4] それまでの記述如何では 0xffffff と書く方が正しい場合もあります。今の時点では気にしなくていいのですが気になるのであれば吉里吉里 2 リファレンスの Layer クラス－fillRect(矩形塗りつぶし) を参照して下さい。

てそれを利用しましょう。

```
var wid = kag.fore.base.width;
var hei = kag.fore.base.height;
kag.fore.base.fillRect( 0 , 0, wid, hei, 0xffffffff );
```

見てもらった通りです。kag.fore.base.width で横幅を、kag.fore.base.height で縦幅を取得します。サンプルではそれぞれを変数に入れて、各変数を利用しています。別に変数に入れる必要もないので次のように書くこともできます。

```
with(kag.fore.base)
{
        .fillRect( 0 , 0, .width, .height, 0xffffffff );
}
```

この場合中カッコは無くてもいいんですけど、可読性が良くなるので私は滅多に省略していません。邪魔だから無くていいものは書かなくていいじゃないか、と思うのであれば次のように書くとスッキリします。

```
with(kag.fore.base) .fillRect( 0 , 0, .width, height, 0xffffffff );
```

4.2.2 反転

予定

4.2.3 ブラー

予定