



The Design and Evolution of “Ghost”

# Ghostの 設計と進化

Phase 01 : 通信仕様から見る構造と設計

Chameleon Ponapalt (ぽな@ばぐとら)

<http://ssp.shillest.net/>

Rev.1

# 自己紹介

- 何かデータランタイム “ベースウェア” SSPの「現」開発者
  - <http://ssp.shillest.net/>
  - 「現」があるからには前任者もいます
- 通称 “所長” “バグのひと”
  - 配布サイト “ばぐとら研究所” の代表なのでそりゃ所長だろうと（自称）
  - もはやエンバグが持ちネタになるほどひどいソフトウェア品質に全米が泣いた。

# 本日のためにゆう

## ■ 簡単に言うと

- 「現在の何かの実装とその理由をガチプログラマーレベルまで解説してその傾向と対策をばりばり再考してみようぜ！」シリーズ
- レベル: ★★ ★ ★ (3段階評価で4)

## ■ 今回のネタ

- 「通信」に焦点を当てます。
- 内部・外部両方の通信実装とその理由

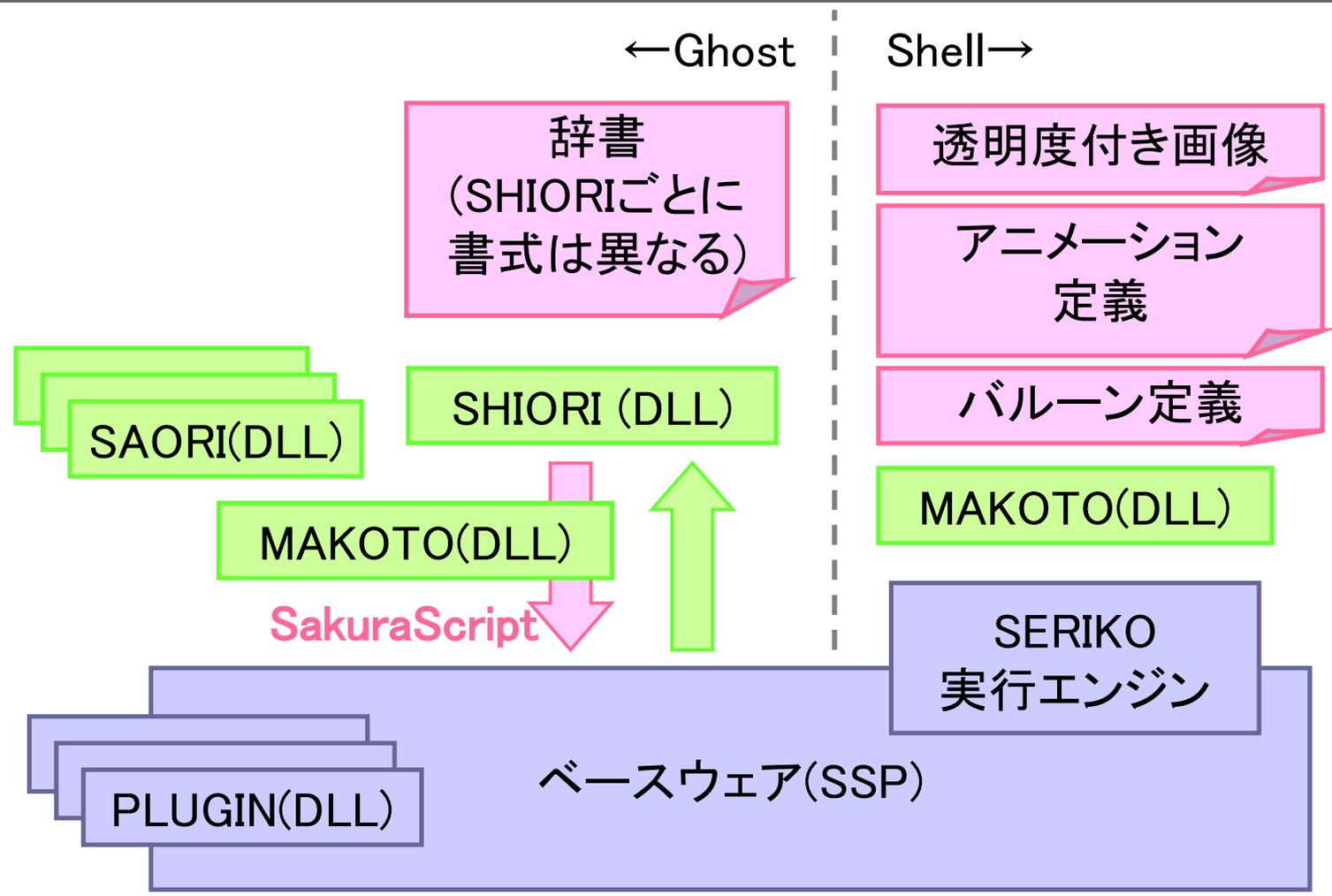


# 1. 全体構造の概観

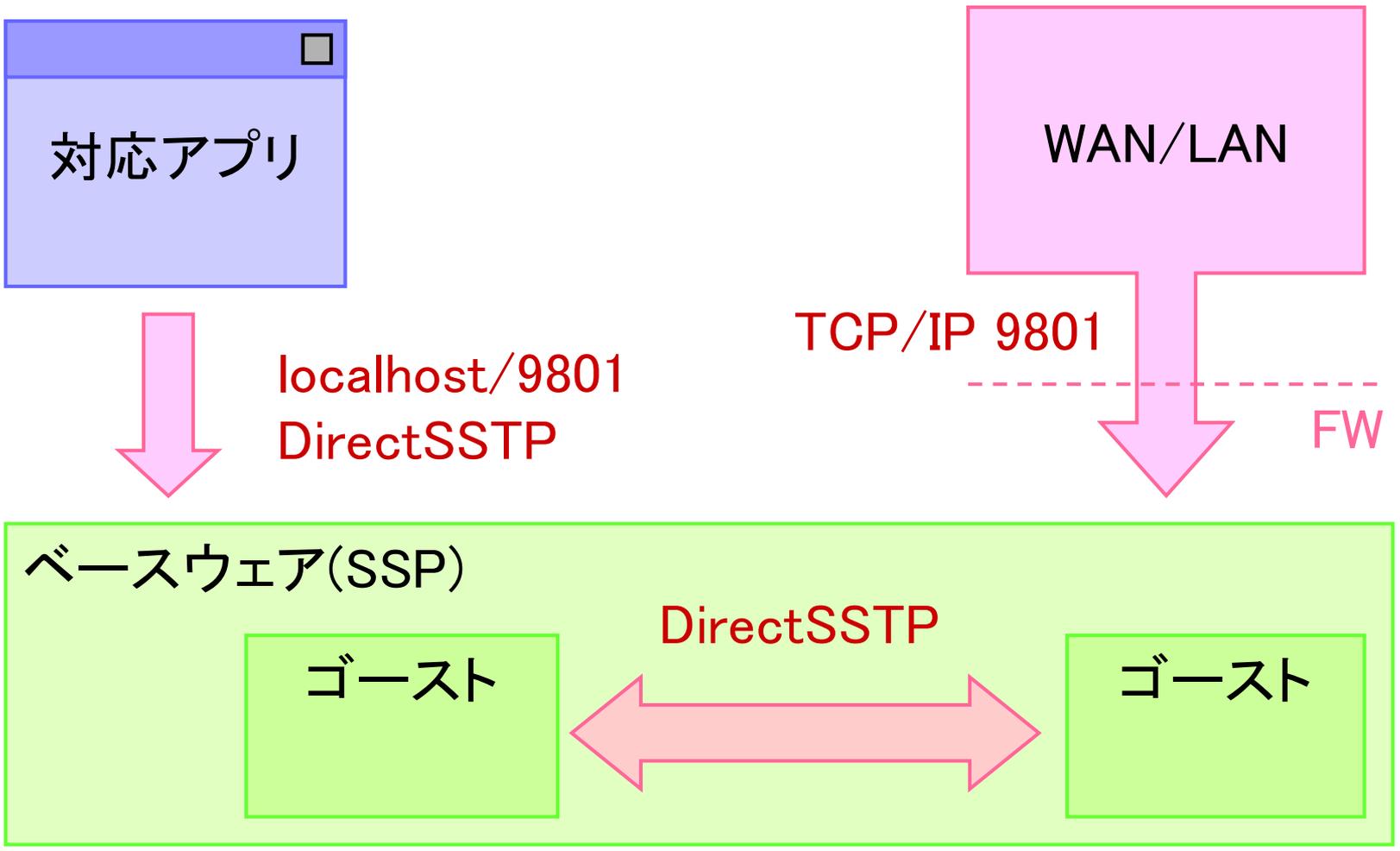
あるいは  
「おさらい」ともいう。



# 構造概観 > 内部構造

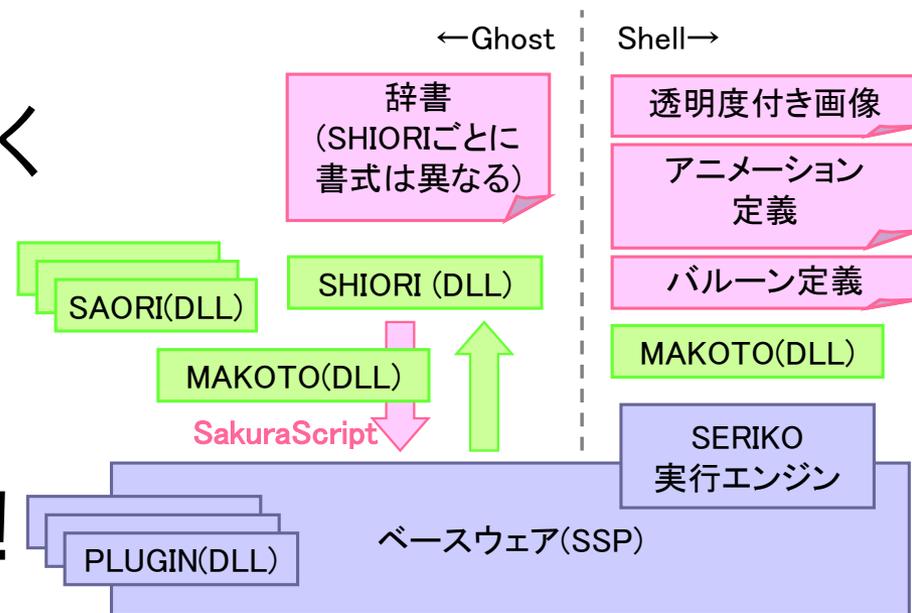


# 構造概観 > 外部通信



# 構造概観> 結合がゆるい？

- 見た目”Shell”とキャラ”Ghost”の分割
  - 互いの制御は”本体”を介さないといけない  
制御用 “SakuraScript”
  - それぞれ非同期  
…「好き勝手に」動く  
特に”Shell”の  
アニメーション
  - 細かく制御できず  
不満!なんとかしろ!



8 / 39  
構造概観 > 結合がゆるい。

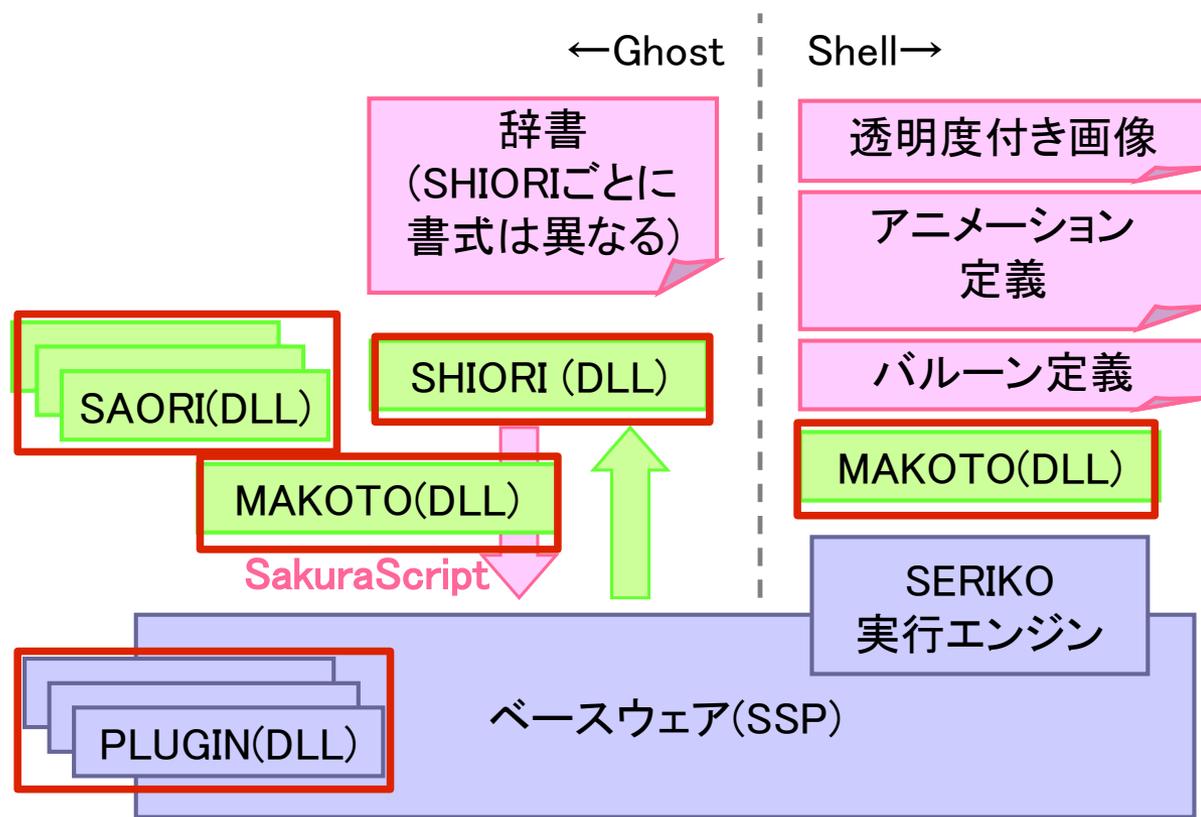
- 昔は”Shell”しかなかった
  - 元々は”Shell”が複数、”Ghost”は1つ
  - “Ghost”を複数持てるようになって今の構造になった
- 実は”Ghost”も1キャラあたり複数持てた
  - 1キャラクターに複数の”Ghost”：制御部
  - 実装も含め複雑で誰も使いこなせなかったので廃止になった

# 構造概観> ゆるい結合の内幕

- 今回は「ゆるい結合」自体のおはなし
  - 今の結合の仕方と問題点を見ることでこの先どうするか考えよう!
- 開発想定環境
  - 使用言語: C / C++
  - 「生の」Win32 API
  - 実行環境: materia、またはSSP上

## 2.内部処理用実装

- SHIORI
- SAORI
- MAKOTO
- PLUGIN



11 / 39

内部実装 > **ただのDLLです。**

```
extern "C" _declspec(dllexport) ...
```

```
//読み込み
```

```
BOOL _cdecl load(HGLOBAL h, LONG len)
```

```
//開放
```

```
BOOL _cdecl unload(void)
```

```
//イベント通知
```

```
HGLOBAL _cdecl request(HGLOBAL h, LONG *len)
```

- 基本はこの3関数のみ
- ほとんどrequestで全てをまかなう

# 内部実装 > load / unload

- BOOL load(HGLOBAL h, LONG len)
  - hの中身は「データを読み込むパス」  
通常DLLと同じディレクトリを指示
  - lenで長さを指定、ゼロ終端ではない
- BOOL unload(void)
  - 開放時に呼び出される・パラメータ無し
  - 失敗(返り値FALSE)でも誰も面倒見ません



# 内部実装 > request

- HGLOBAL request(HGLOBAL h, LONG \*len)
  - 渡されたHGLOBALはDLLがGlobalFree
  - 返り値はDLL側で改めてGlobalAllocしてさらにLONGへのポインタの先にそのサイズを代入
  - 行き来するメモリの中身はHTTPヘッダもどき
  - ゼロ終端ではない

```

GET SHIORI/3.0
Charset: UTF-8
Sender: SSP
SecurityLevel: local
ID: OnMouseDoubleClick
Reference0: 115
Reference1: 313
Reference2: 0
Reference3: 0
Reference4: atfield
Reference5: 0

```

# 内部実装 > 利点欠点

- 利点:とにかく超高速
  - プロセス間通信やソケットやその他複雑な仕組みは一切無しなので高速
  - 実装もずいぶん楽
- 欠点:子亀こけたら親亀も…
  - DLL内で不具合があればアプリごと落ち
  - 別プロセス分離も試したけど互換性が…  
>CROWはこのアプローチ

# 内部実装> その他詳細

- なぜGlobalAlloc / Freeなの？
  - mallocやらnewやらはDLL / EXEの境界をまたぐため使用は不可  
>ライブラリの管理境界外
  - 本当はLocalAllocやHeapAllocで良い  
「共通のメモリマネージャ」が使えるなら  
それこそ何でも良い

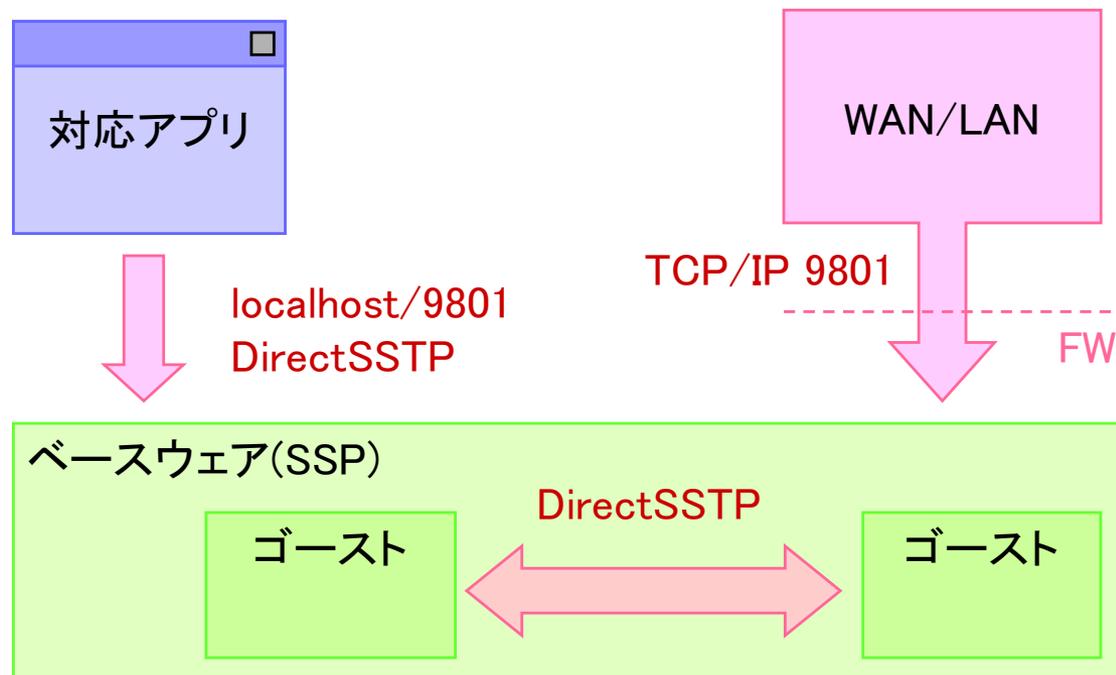


# 内部実装> その他詳細

- なぜゼロ終端ではなく長さ指定なの？
  - 正直ゼロ終端で構わないと思う…
  - PASCAL文字列？Delphiでも使ってたの？
- なぜ `__cdecl` なの？
  - 正直 `__stdcall` でも良いかと…
- Windows以外ではどうしてるの？
  - 動的ライブラリ読み・メモリやりとりは同じ
  - 丸々実装している処理系も

# 3.外部との通信

- (Socket)SSTP
- DirectSSTP
- FMO
- Mutex



# 外部通信 > SSTP

- **S**akura (あるいはSinatiku) **S**cript **T**ransfer **P**rotocol
- 動作中のプロセス「外」から制御用スクリプトを送り込むための仕組み
  - Interprocess Communication (IPC) プロセス間通信
  - TCP/IP上に乗せる (Socket)SSTPと OSのIPC機構上に乗せるDirectSSTP

## 外部通信 &gt; SSTP

- 外から通信するか内部通信するかの違いで、中身はほとんど違いがない
  - HTTPもどきの形式でSakuraScriptをヘッダ内に渡すあたり全然変わらない

SEND SSTP/1.2

Charset: UTF-8

Option: notranslate

Ghost: 所長たん

Sender: Emily

Script: ¥0¥s[7]エンバグ屋言うな！¥e

# 外部通信 > なぜDirectSSTP?

- (Socket) SSTPは、先程のをとりあえず9801番ポートに叩き込めば動きます
- TCP/IPでの通信の欠点
  - TCP/IPを使うことによるコスト  
※ヘッダ文字列解析に比べたら大したことはない
  - ポート9801しか窓口がない  
しかもファイアウォールで蹴られる…  
(Windows標準のFWですら警告が出る)

# 外部通信 > なぜDirectSSTP?

- DirectSSTPの利点
  - OS内部でしか使えない:セキュリティ向上  
ファイアウォールも関係ない
  - 比較的軽い負担
  - 「ポート番号」に縛られないので  
窓口が増やせる(”Ghost”ごとに)
- DirectSSTPの欠点
  - OSべったりの実装になりがち
  - 当然信用できるLAN内でも使えない

# 外部通信 > DSSTPの仕組み

- IPC用ウィンドウメッセージを打ち込む  
SendMessage(相手先hwnd, WM\_COPYDATA,  
自分のhwnd, COPYDATASTRUCTポインタ);
- WM\_COPYDATA
  - dwData: 9801
  - cbData / lpData: SSTPそのもの



# DirectSSTP>hwndの取得

- hwnd (ウィンドウハンドル) って?
  - 特定のウィンドウを示すIDのようなもの「ウィンドウ」を「ハンドル(扱う)」もの
  - 通常、ID=0番 (¥0側) のウィンドウを指すウィンドウハンドルと通信
- どうやって取るの?
  - ごりごりFindWindow → ダメ! 処理系依存
  - “FMO” 内に載ってます



# DirectSSTP>FMOって?

- File Mapping Object
  - ファイル(File)をメモリ空間上に割りつける(Mapping)ためのもの(Object)
- ファイルに関連付けられて「いない」FMOも作れる
  - OS内の「プロセス間共有領域」が作れる
  - キャラクターのhwnd「カタログ」にぴったり



# FMO > 実際の中身

最初の4バイト=DWORD値でFMOのサイズ 最後=空行(ゼロじゃないよ!)

ssp\_fmo\_header\_00001680\_000604c4.hwnd¥1394436 ←

ssp\_fmo\_header\_00001680\_000604c4.name¥1Emily

ssp\_fmo\_header\_00001680\_000604c4.keroname¥1Teddy

ssp\_fmo\_header\_00001680\_000604c4.path¥1D:¥ssp¥

ssp\_fmo\_header\_00001680\_000604c4.sakura.surface¥15000

ssp\_fmo\_header\_00001680\_000604c4.kero.surface¥110

ssp\_fmo\_header\_00001680\_000604c4.kerohwnd¥13277970

ssp\_fmo\_header\_00001680\_000604c4.ghostpath¥1D:¥ssp¥ghost¥emily4¥

a117baf658f123f4336d01b766475eeb.path¥1D:¥materia¥

a117baf658f123f4336d01b766475eeb.hwnd¥1132926 ←

a117baf658f123f4336d01b766475eeb.name¥1さくら

a117baf658f123f4336d01b766475eeb.keroname¥1うにゅう

a117baf658f123f4336d01b766475eeb.sakura.surface¥11

a117baf658f123f4336d01b766475eeb.kero.surface¥110

¥1はバイト値1

## FMO&gt; 詳細実装

## ■ 作成

```
CreateFileMapping(INVALID_HANDLE_VALUE,  
NULL,PAGE_READWRITE,0,32768,"Sakura");
```

## ■ 参照

```
MapViewOfFile(hFMO,  
FILE_MAP_ALL_ACCESS,0,0,0);
```

## ■ 同時書き込みロック用に “SakuraFMO” Mutex

## ■ 書き込まれたタイミング通知用に

```
sakuraAPIMsg = RegisterWindowMessage("Sakura");  
SendNotifyMessage(HWND_BROADCAST  
,sakuraAPIMsg,1024,プロセスID);
```

# 外部通信>現状の問題点

- セキュリティ問題
  - SakuraScriptの危険なタグ
    - >一応実行不能
  - FMOの中身が壊れたら…
    - >共有メモリ自体が良くない実装
- Windowsべったり+古い方法
  - 似たような仕組みはどんなOSにもある?
  - ウィンドウメッセージ経由の方法は”古い”
    - >他に簡便な仕組みは無い

## 4. 伺か以外の例

- キャラリナ (ペルソナウェア)
- Apricot

# 別の例> キャラリナ

- 「見た目定義部」は一応ある…が。
  - 画像ファイル+アニメーション定義のみ  
hpg/hp2/hp3 (専用ファイル形式)  
hps (アニメーション定義用テキスト)
  - 非同期アニメーション実行も可能
  - “Shell” のようなデータセットはない  
> 見た目部分を独立して扱えない
- 使用言語: 「綾織」のみ
  - SHIORI仕様のように取り替え不可

## 別の例&gt; キャラリナ

## ■ かなり密結合気味

- そもそも「内部通信」って何食えるの? 構造
- 表情変更: 綾織スクリプト内部で描画命令  
画像をウィンドウに直接描ける・加工も可
- その分「なんでも好きなだけできる」

```
PersonaCG( "tsumu" );  
Talk( "...¥w...¥w...¥w" );
```

LoadBitmap  
DisplayImage(2)

```
PersonaCG( "normal3" );  
Talk( "んニヤ?¥w¥w" );  
Talk( "ひさしぶりだニヤー?¥w¥n" );
```

「トーク」  
見覚えのあるような  
ないような…!?

# 別の例> キャラリナ

- 外部通信機能もそれなりに
  - プラグインによる拡張
  - 「インフォデリ」「キャラデリ」
- SSTPのような自由な制御手段は無い
  - 構造上同等のことを行うには「外部から綾織スクリプトを流しこむ」という超危険な仕様が要る…



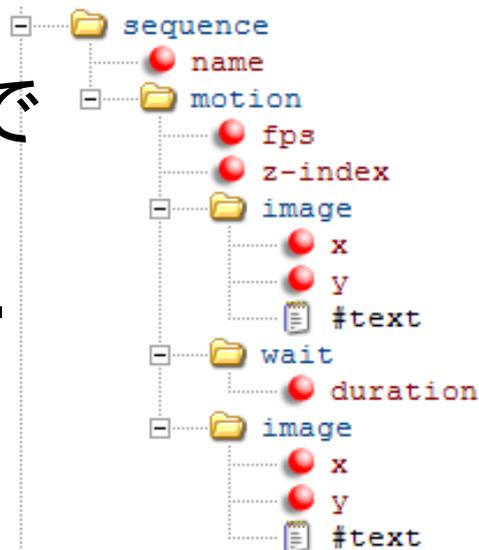
# 別の例 > Apricot

## ■ XMLファイル+画像によるシンプル定義

- script.txt時代  
=SHIORIが無かった頃を思い出す…

- 喋りもアニメーションも、“sequence”  
=手順リストで逐次実行

- イベント定義・  
条件分岐  
(制限あり)



```

Say
12
4
202
62
Images\Hachune-Face-03.png
4000
199
65
Images\Hachune-Face-01.png
  
```

# 別の例>Apricot

- シンプルな構造なので「通信」自体不要
- その分「できないこと」も多い
  - そもそも主目的が違う:「テキスト情報を自動解析して喋る」のが主
  - キャラクターを丸々作る環境とは違う
  - 代わりに「絶対に枯渇しない自動生成ランダムトーク」がたっぷり

# 5.そして今後…

- 機能強化
- ゆるい結合の利点
- 疎結合の利点を損ねない拡張

# 今後>機能強化

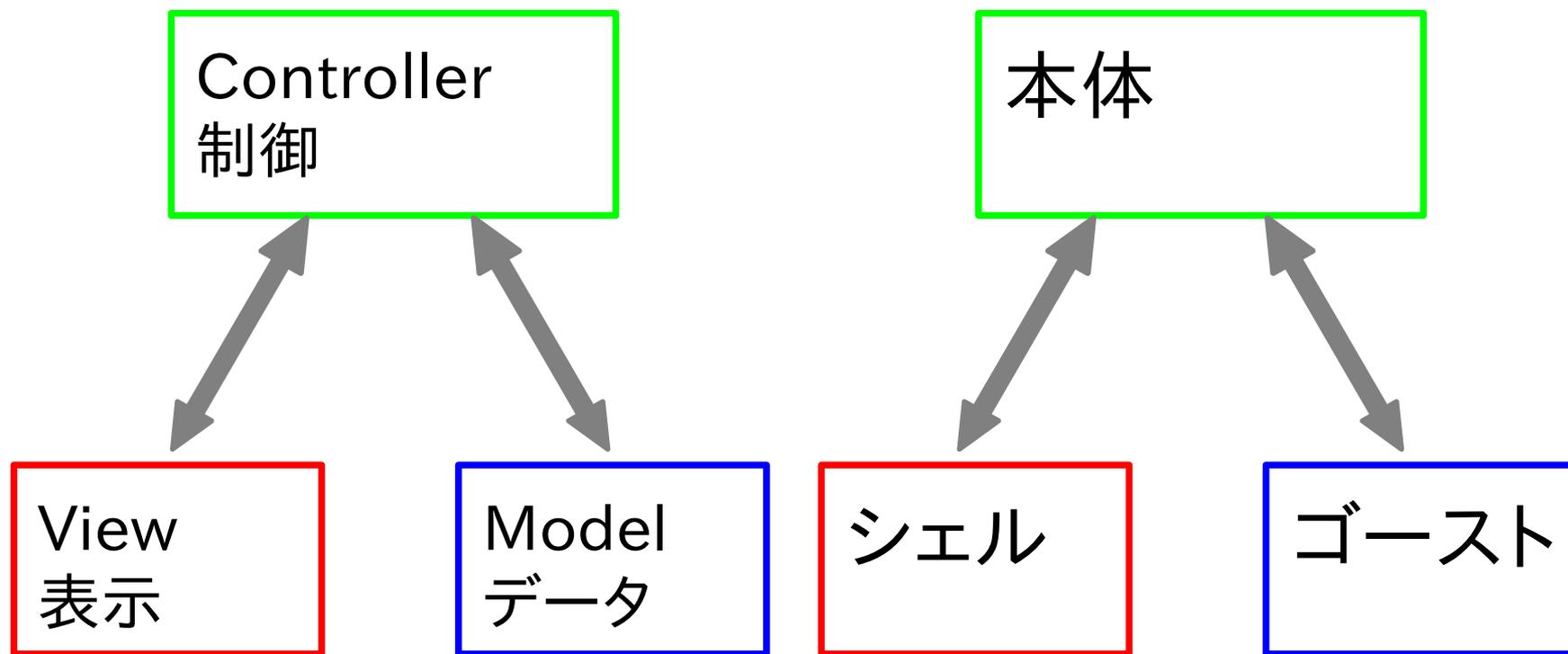
- 表現力強化
  - 「吹き出し」アニメーション
  - いっそ吹き出し内をHTML並みに
  - “Shell” 内にセリフを
- 制御強化
  - 動的な “Shell” 定義
  - いっそ “Shell” に SHIORI もどきを



# 今後> Re:ゆるい結合

- 柔軟なデータの変更
  - 追加”Shell”なんてこの構造がないと…
- 開発が楽な「まるなげ」構造
  - 「イベント駆動」「Model-View-Controller」  
とか聞いたことありますか？  
→ 開発を楽にするための一般的構造
  - まるなげ構造は「オブジェクト指向」の  
根幹にも通じる

## Re:ゆるい結合 &gt; MVC



# 今後> Re:機能強化

- 初めに挙げたような拡張を「見た目」「キャラクター制御」が密に結合しない構造を保ったまま行う必要がある
  - 動的“Shell”定義機能あたり危険
  - 吹き出しへの過度な拡張もちょっと危険
- 密と疎のバランス…難しいよね!
  - 今のところ偶然良さげなバランスなのでぜひ維持していきたいが…

# 今日はここまで!

ご清聴ありがとうございました