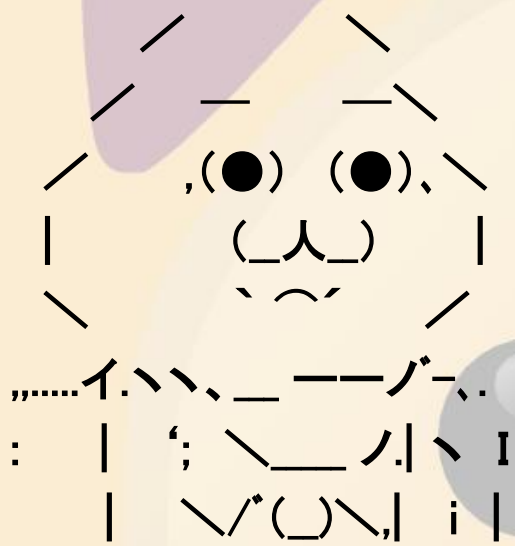
匠の伝承w

マルチな時代の設計と開発

パート2

スピーカー自己紹介



ゆーちです。

ハンドル名です。

本名は、内山康広といいます。

47歳です。

おっさんです。_|￣|○

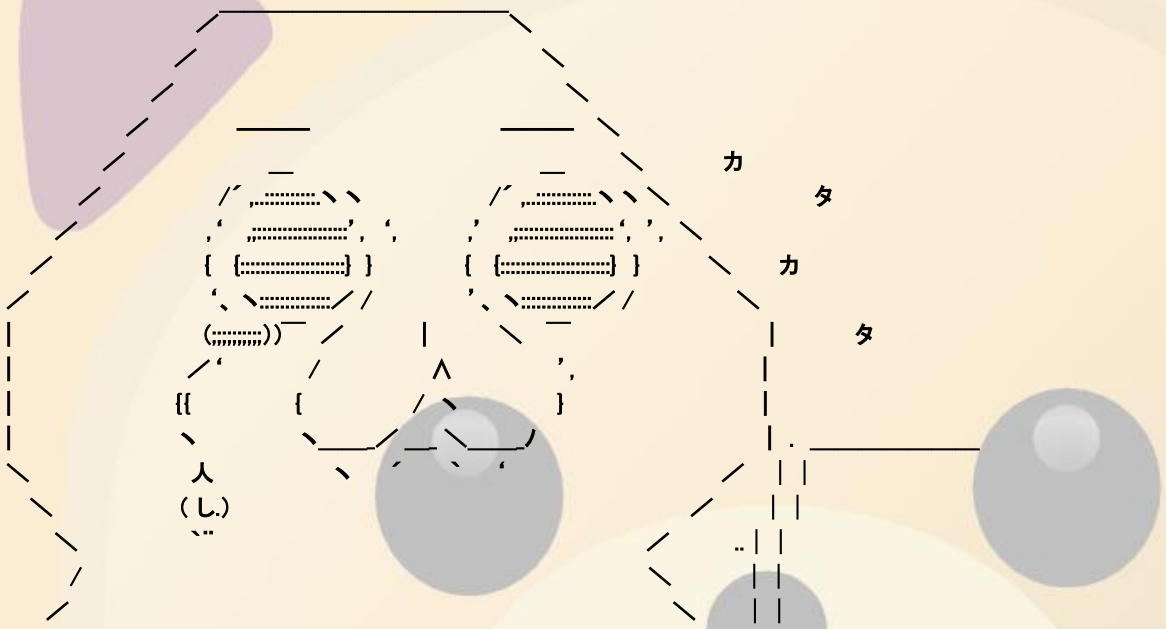
株式会社シーソフト代表取締役です。

現役のエンジニアです。プログラム書いてます。

メールソフト Becky! 用の BkReplyer という作品が微妙に有名らしす。
2ちゃんねらーではありません。

Special thanks for 2ch.

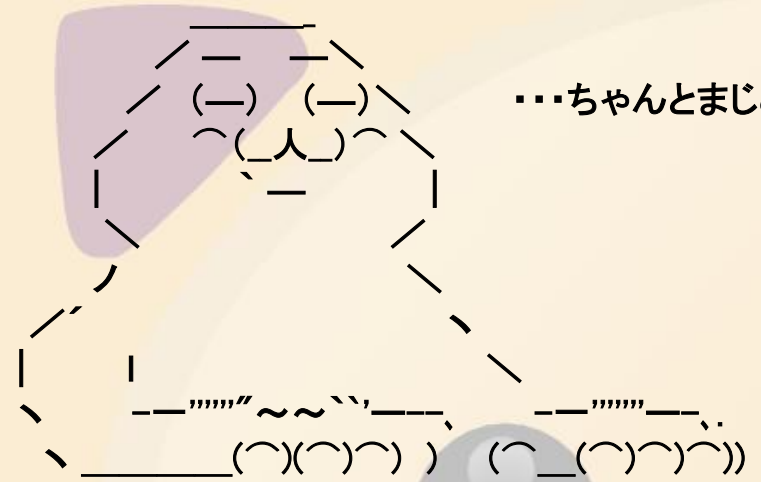
前回のおさらい



デスマーチプロジェクトばっかりやってきた。

しごとください。

そーじゃなくて。



・・・ちゃんとまじめな話をしたんだお。

開発者はプロセス指向にとらえがち。

オブジェクト指向は『モノ』をとらえる。

『モノ』に対する時間軸のイベントを列挙。

時間軸へのイベントが『状態』を作る。

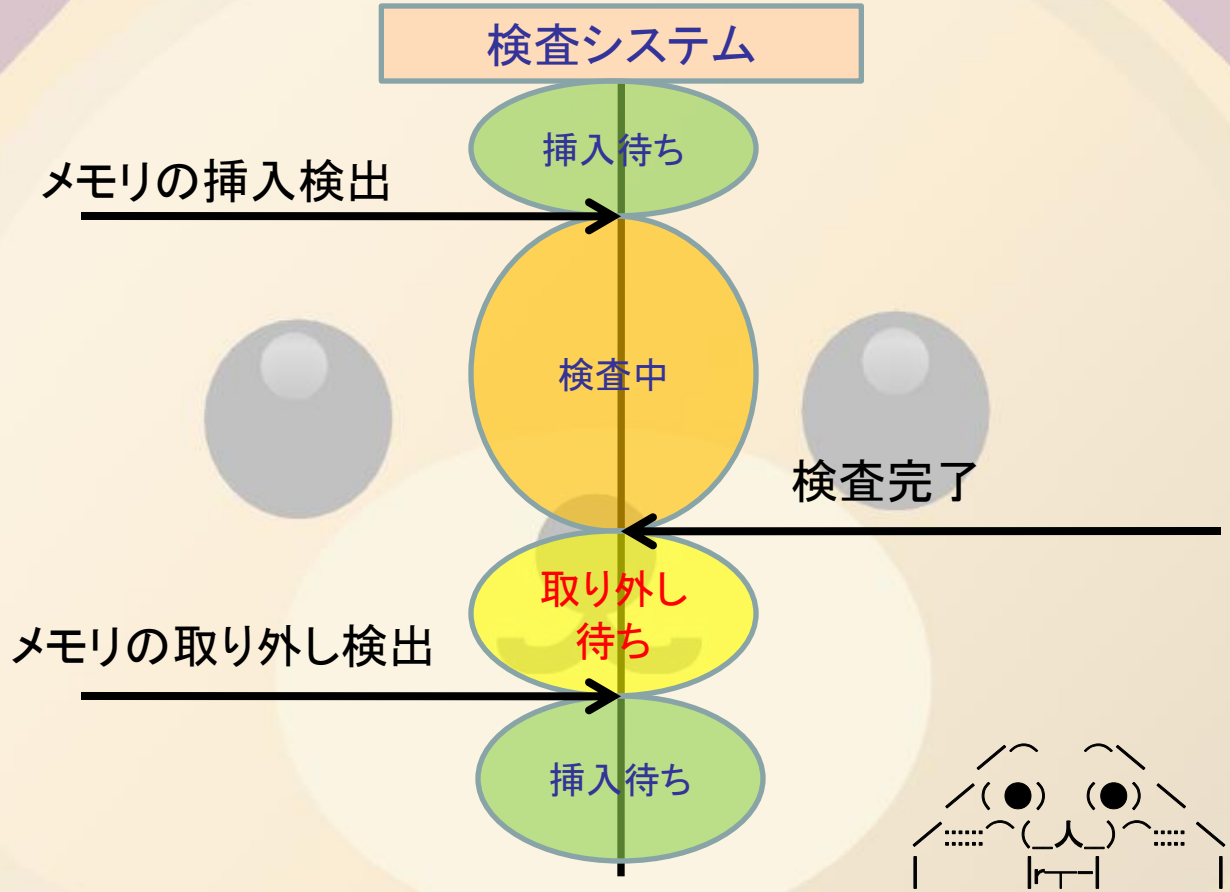
開発は『状態』別に分けて考える。

USBメモリの検査

要件

1. USBメモリの製造工程で製品検査
2. 転送速度が規定値の範囲内かを検査
3. 検査が完了したら抜き差しで次の製品

イベントトレースと状態の関係



いっぽん！？

状態遷移表の見方、考え方

状態 イベント	メモリ挿入待ち (0)	検査中 (1)	取り外し待ち (2)
メモリ挿入検出	検査を開始 状態を検査中に →(1)	エラー表示 検査中断 →(2)	ログ記録 検査を開始 →(1)
検査完了通知	ログ記録 →(0)	検査結果を表示 →(2)	ログ記録 →(2)
メモリ取り外し検出	ログを記録 →(0)	エラー表示 検査中断 →(0)	次の検査準備 →(0)

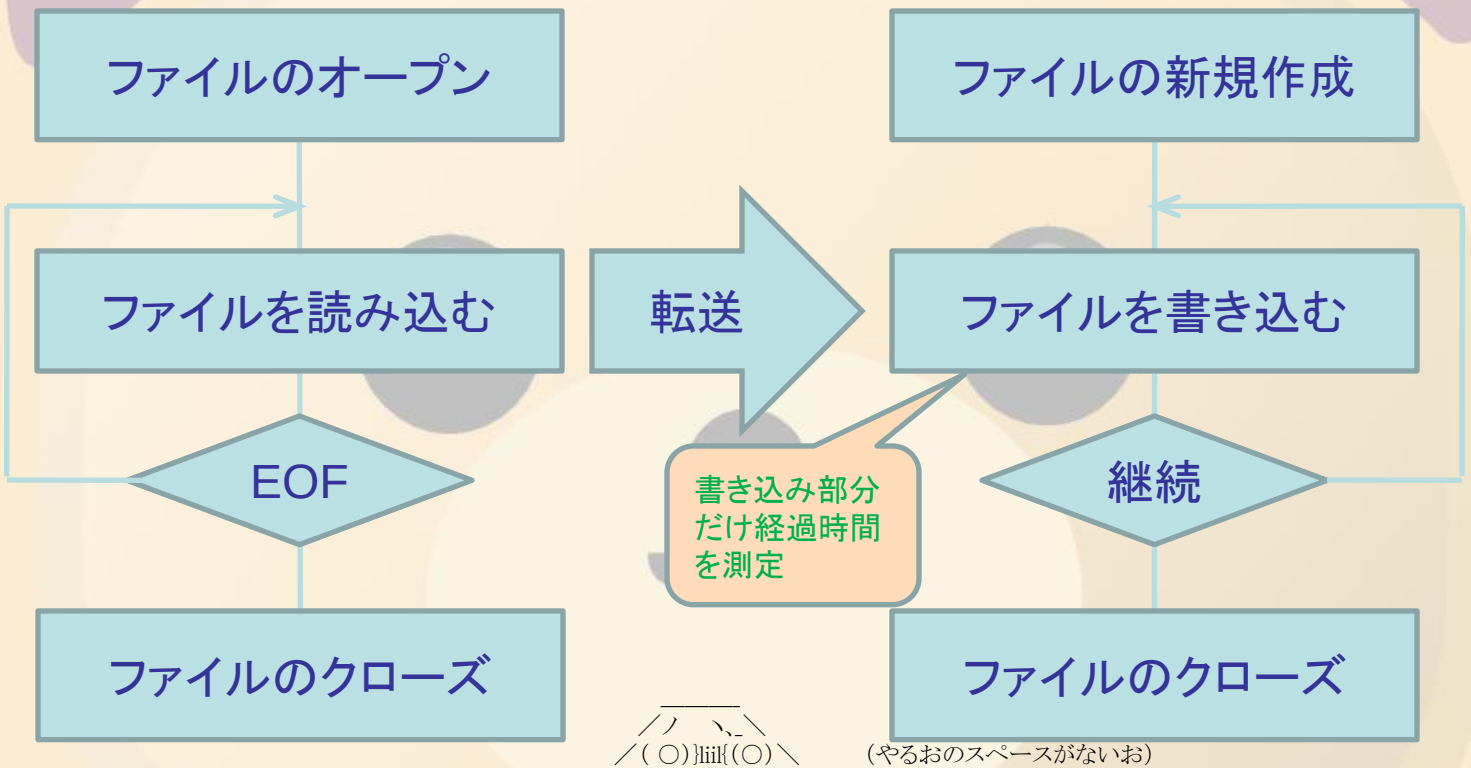
状態別にイベント発生時の処理を書く

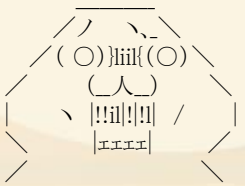
本日のテーマ

同期処理から非同期処理へ

え？ステートパターンじゃないの？

ファイル転送部分の考察

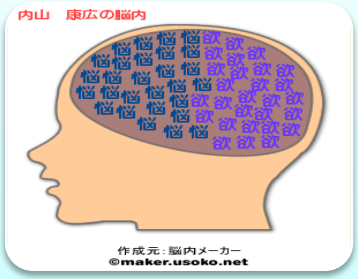




（やるおのスペースがないお）

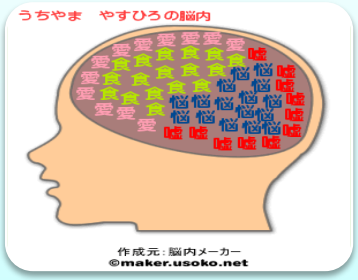
※ファイルの転送では厳密な転送速度を測定できませんが、この業務ではある程度の性能があることがわかれば、OKとのことでした。

ファイルを転送し、書き込みの時間を測定する



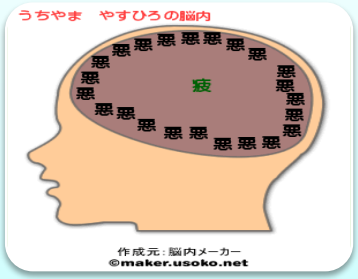
ファイルの準備

- 読み込み用のファイル1をオープンする
- 書き込み用のファイル2をオープンする



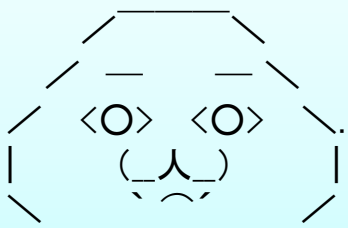
ファイルの転送

- ファイル1を読み込む
- ファイル2に書き込む（ここで時間測定）



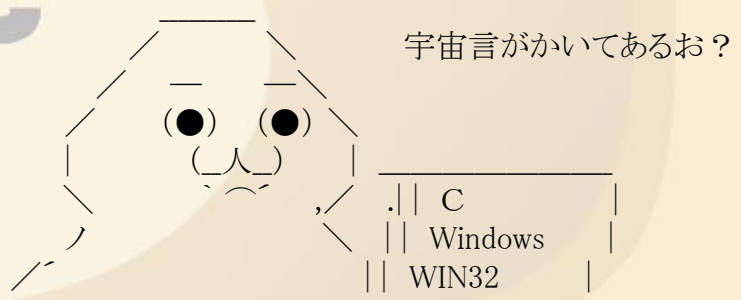
ファイルを閉じる

- ファイル2を閉じる
- ファイル1を閉じる



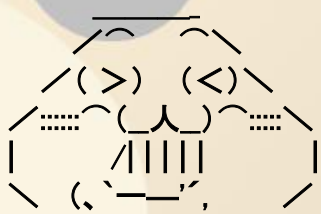
ファイルの準備

```
static const int BUF_SIZE = 32768;
bool FileCopy( const char *src_, const char *dst_, DWORD *msec_ )
{
    HANDLE src = CreateFile( src_, OPEN_EXIST, ... );
    if( src オープンできない )
    {
        return false;
    }
    HANDLE dst = CreateFile( dst_, CREATE_ALWAYS, ... );
    if( dst オープンできない )
    {
        CloseHandle( src );
        rerturn false;
    }
```



ファイルの転送

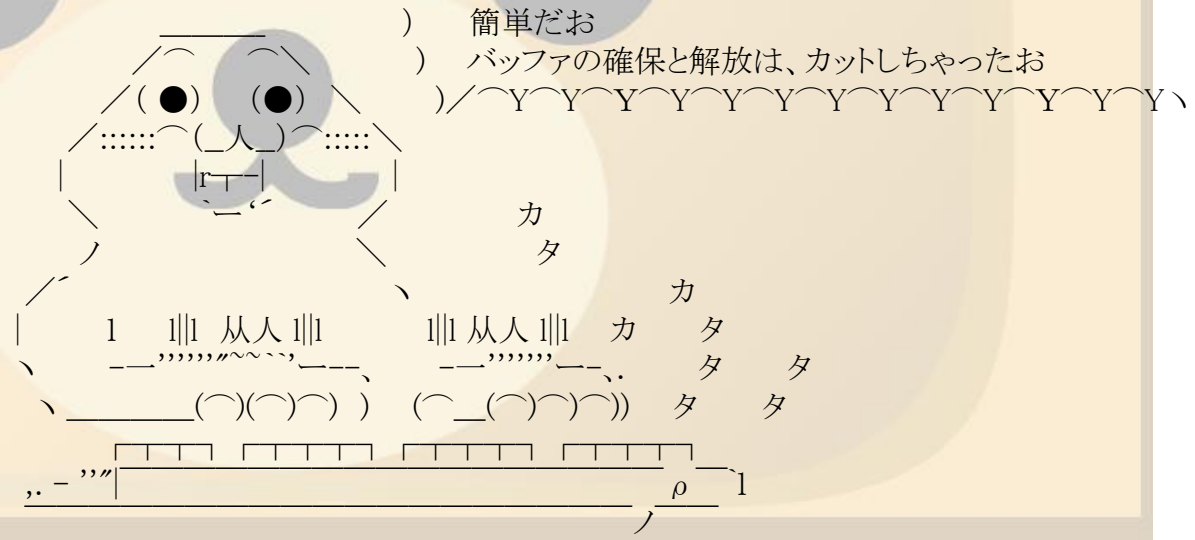
```
bool rc = true;
DWORD total_msec = 0;
try
{
    DWORD   result;
    while( ( rc == true )
        && ( ReadFile( src, buffer, BUF_SIZE, &result ) == TRUE )
        && ( 0 < result ) )
    {
        DWORD   stt = GetTickCount();
        DWORD   write_result;
        rc = WriteFile( dst, buffer, result, &write_result );
        total_msec += ( GetTickCount() - stt );
    }
}
```



GetTickCount() だと、
ホントはうまくいかないお

ファイルを閉じる

```
    if( rc == true )
    {
        *msec_ = total_msec;
    }
    CloseHandle( dst );
    CloseHandle( src );
    return rc;
}
```



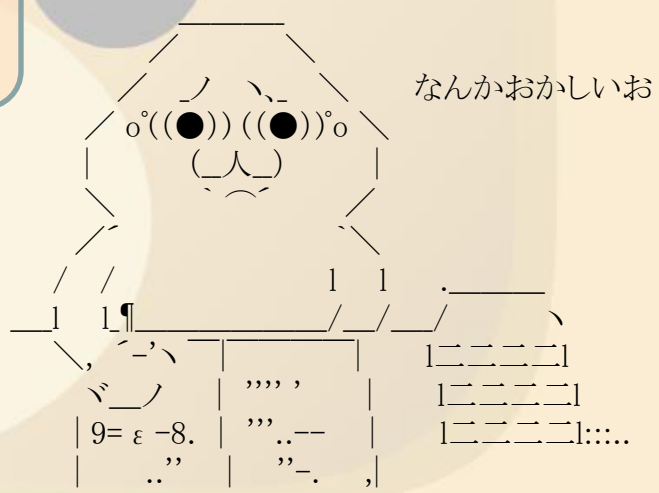
注目

bool FileCopy(const char *src_, const char *dst_, DWORD *msec_)

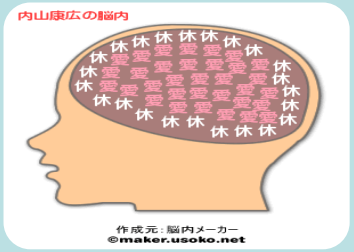
書き込み時間を格納する

書き込むファイル名

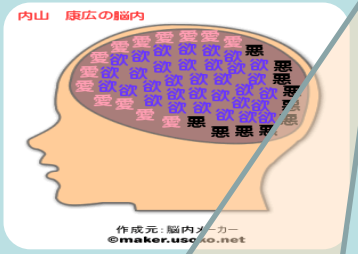
読み込むファイル名



違和感あるよね？

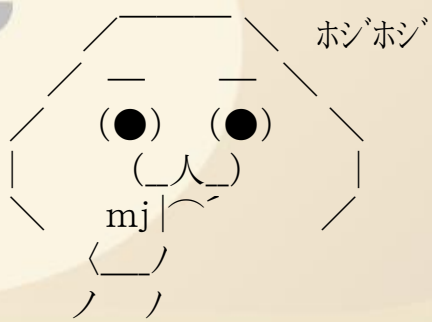


ファイルを転送する



書き込み時間を計測する

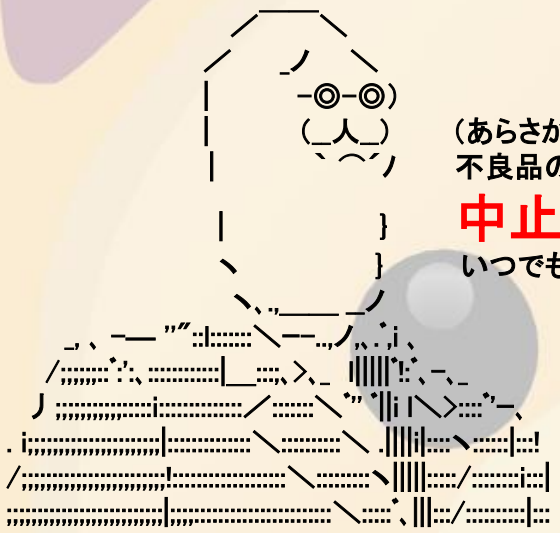
そもそも別々のことなのに、
一度にしようとしている件。



ん？また脳内？

出番が少ないお。

追加要件



（あらさがし、あらさがし・・・）
不良品のUSBメモリをさすと、ダンマリになりますね。

中止ボタンを出して

いつでも中止できるようにしてください。

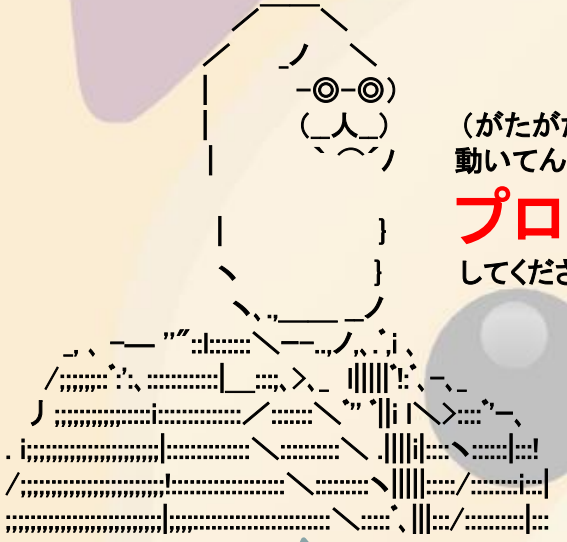
プギャー！！！
聞いてねぇよ！

パラメータ増やして対応・・・

```
bool FileCopy( const char *src_, const char *dst_, DWORD *msec_ ,
               bool& canceled_ )
```

コピー中にこの変数が変化したときは、中断を意味しますよ、とか。

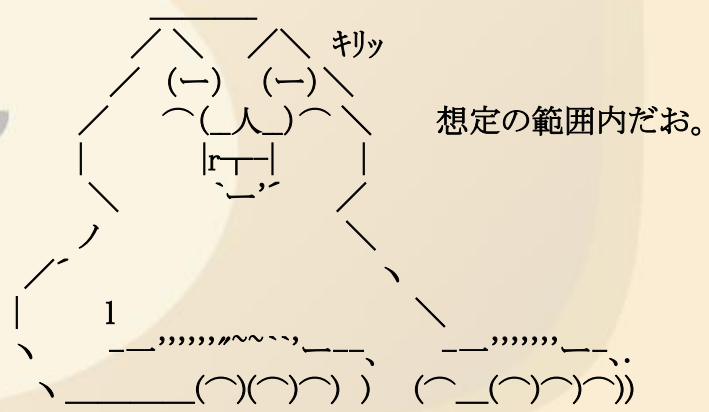
追加要件



あ。納期2日ね。

（がたがた言ってると、もう仕事 流さねぇぞ！）
動いてんのか、死んでんのかわかりませんね。

プログレスバーを出すように
してください。

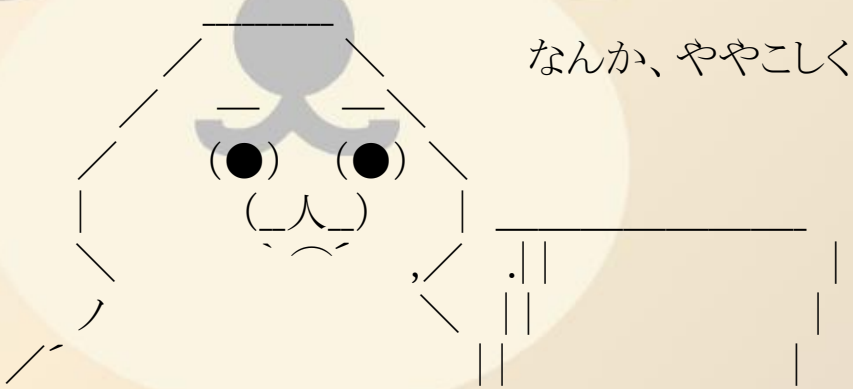


想定の範囲内だお。

パラメータ増やして対応・・・

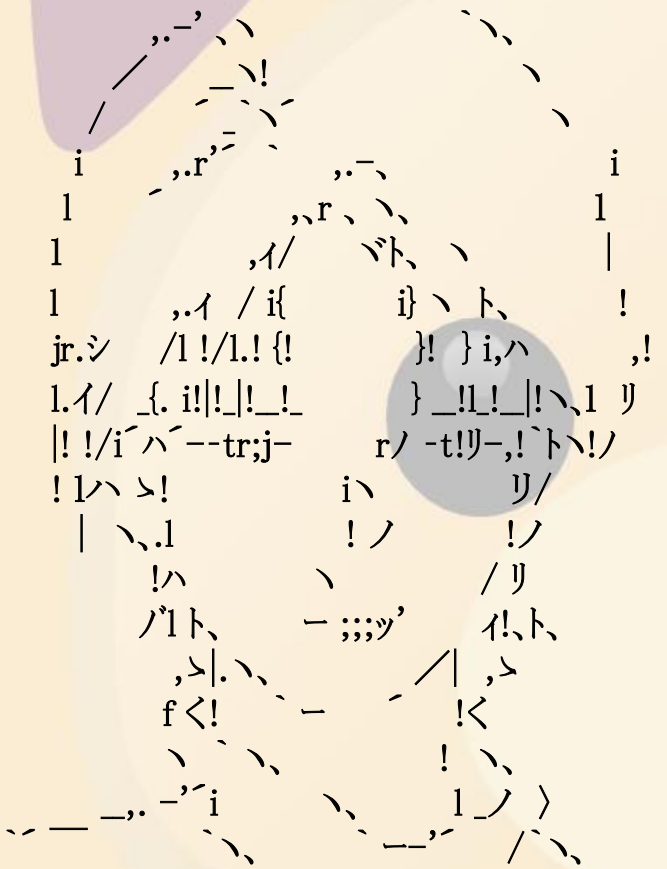
```
bool FileCopy( const char *src_, const char *dst_, DWORD *msec_ ,
               bool& canceled_ , CALLBACK _FUNC*on_progress_)
```

コピーの経過中に、この関数、呼び出してね。



なんか、ややこしくなったお。

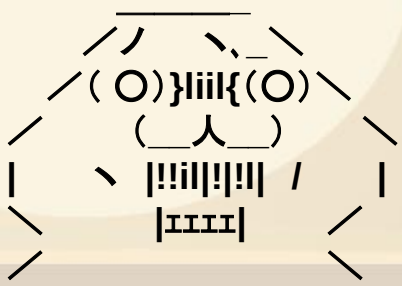
再利用できる？



別の仕事でファイルのコピーが必要なの。

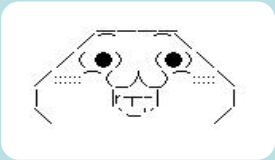
バッチのコマンドラインだからUIいらないわよ

たしか、作ってたわよね。5秒でちょうだい。



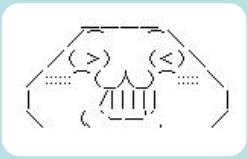
コピペで同じようなのつくるお。

何かが間違ってるはず・・・



ファイルを転送する

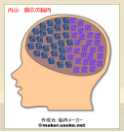
書き込み時間を計測する



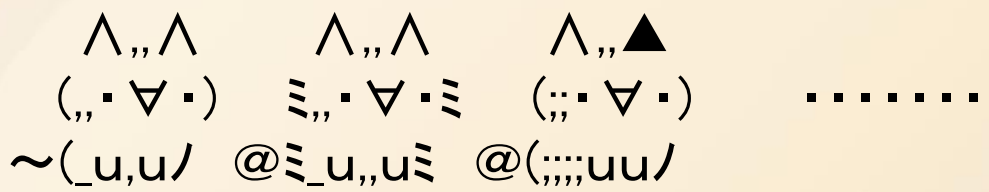
処理を中断する

プログレスバーで経過表示する

アニメーションを表示する・・・とか



わたしの場所とるなYO


∧,,∧
(,,・∀・)
～(_u,uノ
∧,,∧
ミ,,・∀・ミ
@ミ_u,,uミ
∧,,▲
(;;・∀・)
@(;;;;uuノ
・・・・・・・

『モノ』と『動き』の関係を考えよう

オペレータ

USBメモリ

検査システム

測定結果表示

検査結果表示

プログレスバー

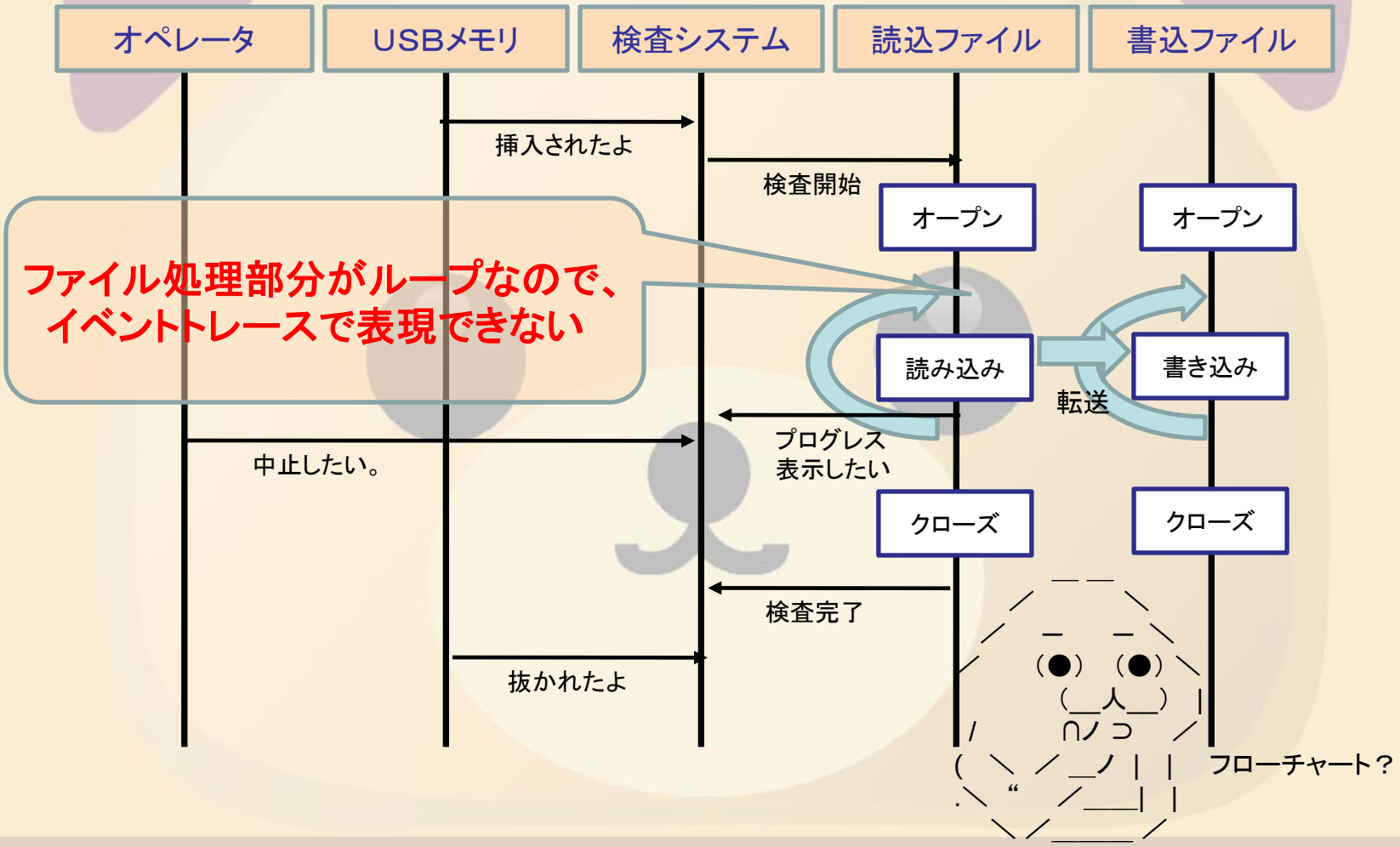
中止ボタン

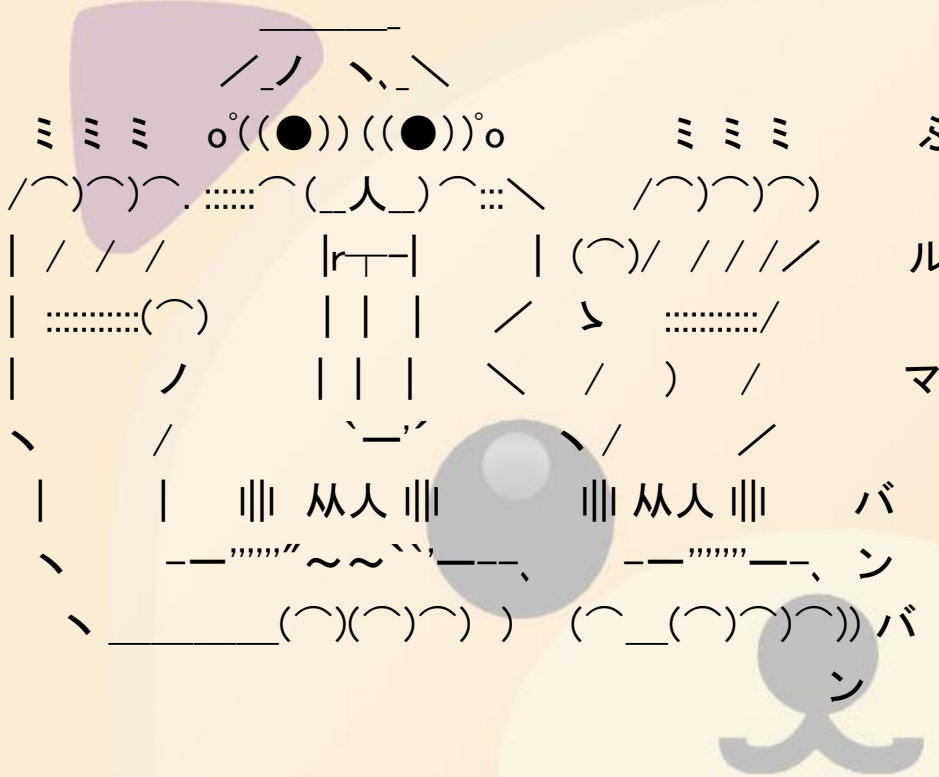
読込ファイル

書込ファイル

一度にファイルを読み書きしてしまうと、途中での中断ができないので、分割して読み書きを繰り返す方法を選択します。

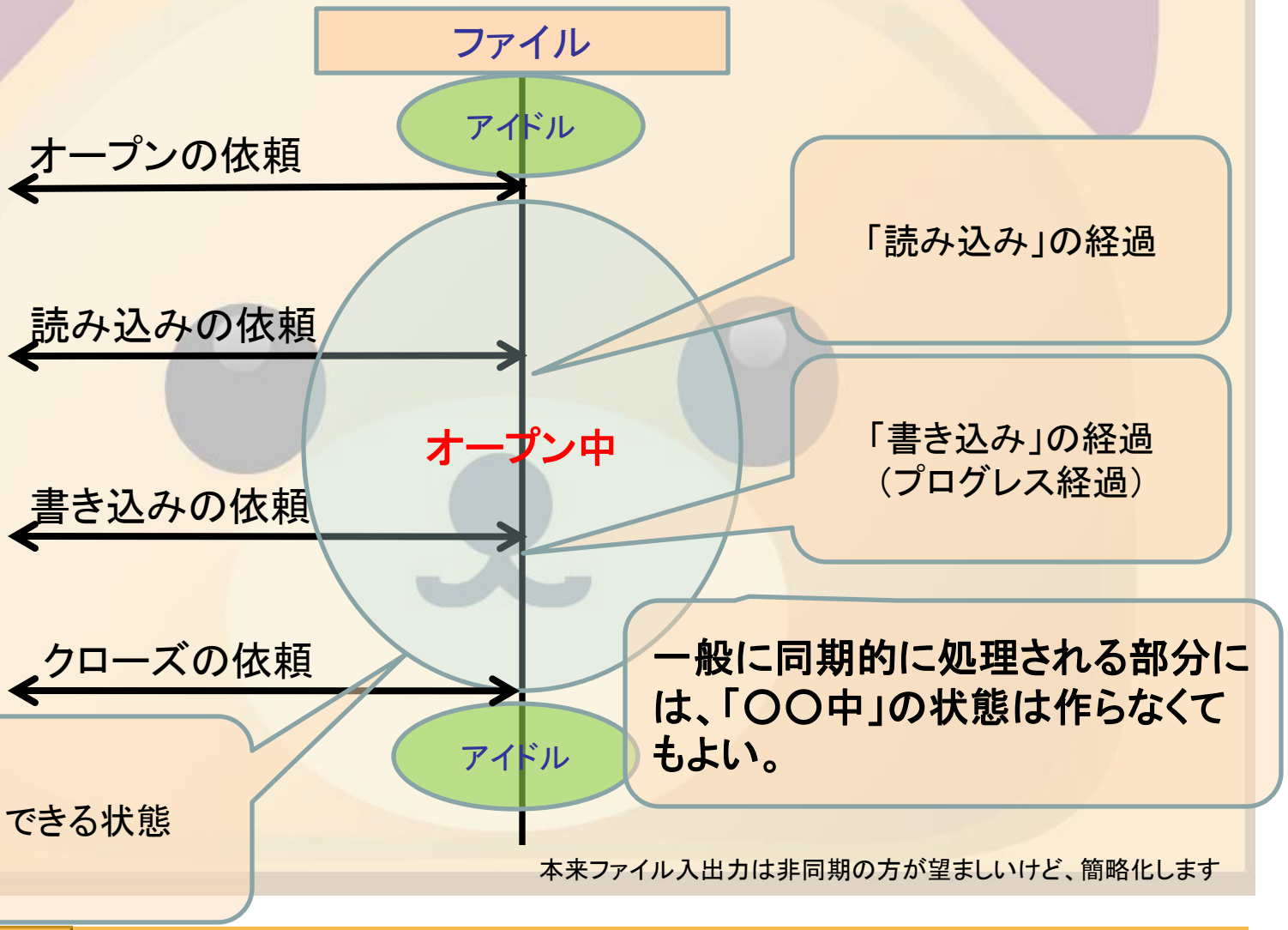
イベントトレースで考えてみる





ループをぶったぎる！

ファイルクラスとオブジェクトの状態



本来ファイル入出力は非同期の方が望ましいけど、簡略化します

イベントトレースを書いたら？

出番が少ないお

状態遷移表書きましょ。

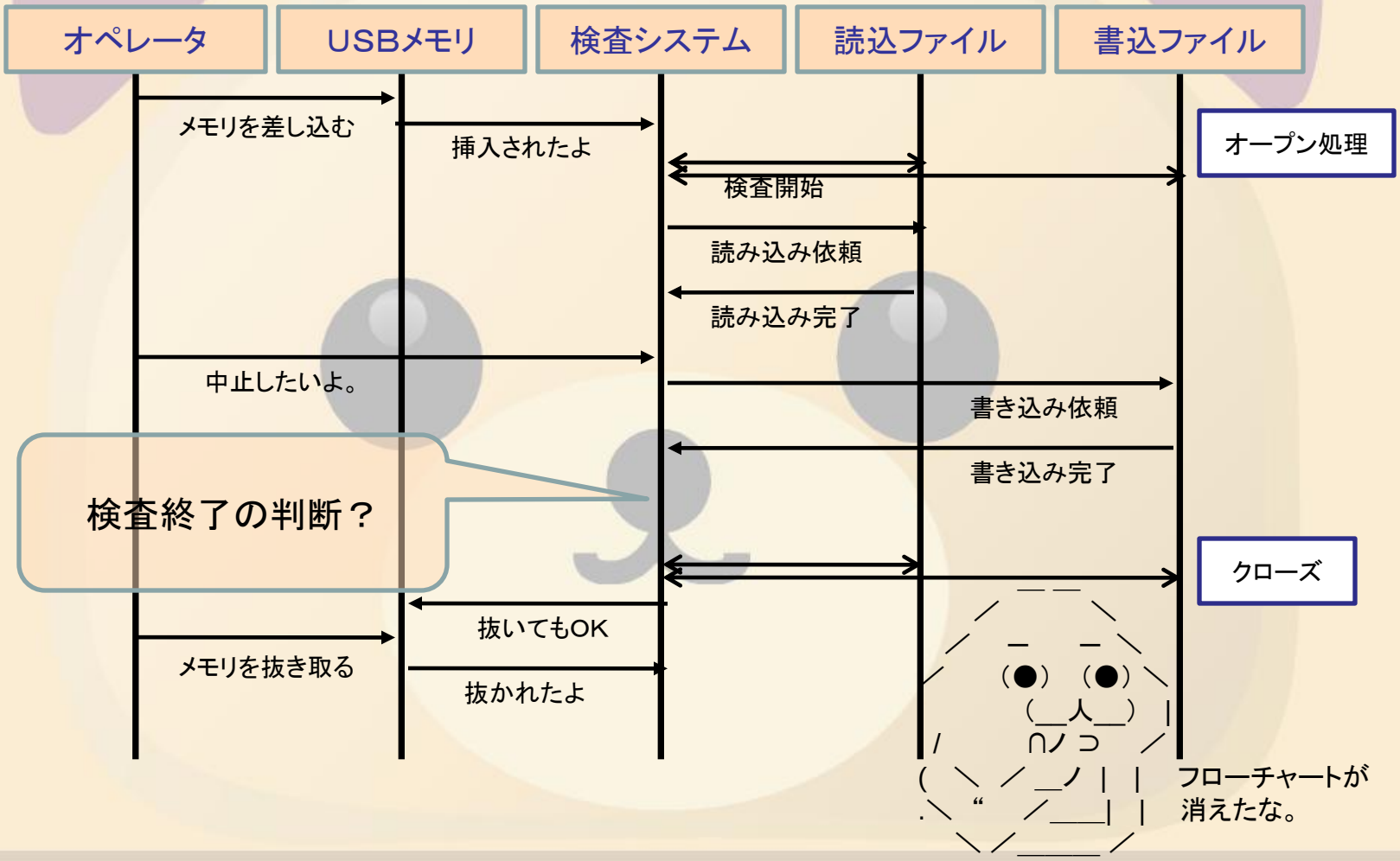
状態 イベント	アイドル (0)	オープン中 (1)
オープン依頼	ファイルを開く成功→(1) 失敗→(0)	ログ記録 →(1)
クローズ依頼	ログ記録 →(0)	ファイルを閉じる →(0)
読み込み依頼	ログを記録 →(0)	読み込み処理 →(1)
書き込み依頼	ログを記録 →(0)	書き込み処理 →(1)
中止依頼	→(0)	ファイルを閉じる →(0)

ファイルのクラス。

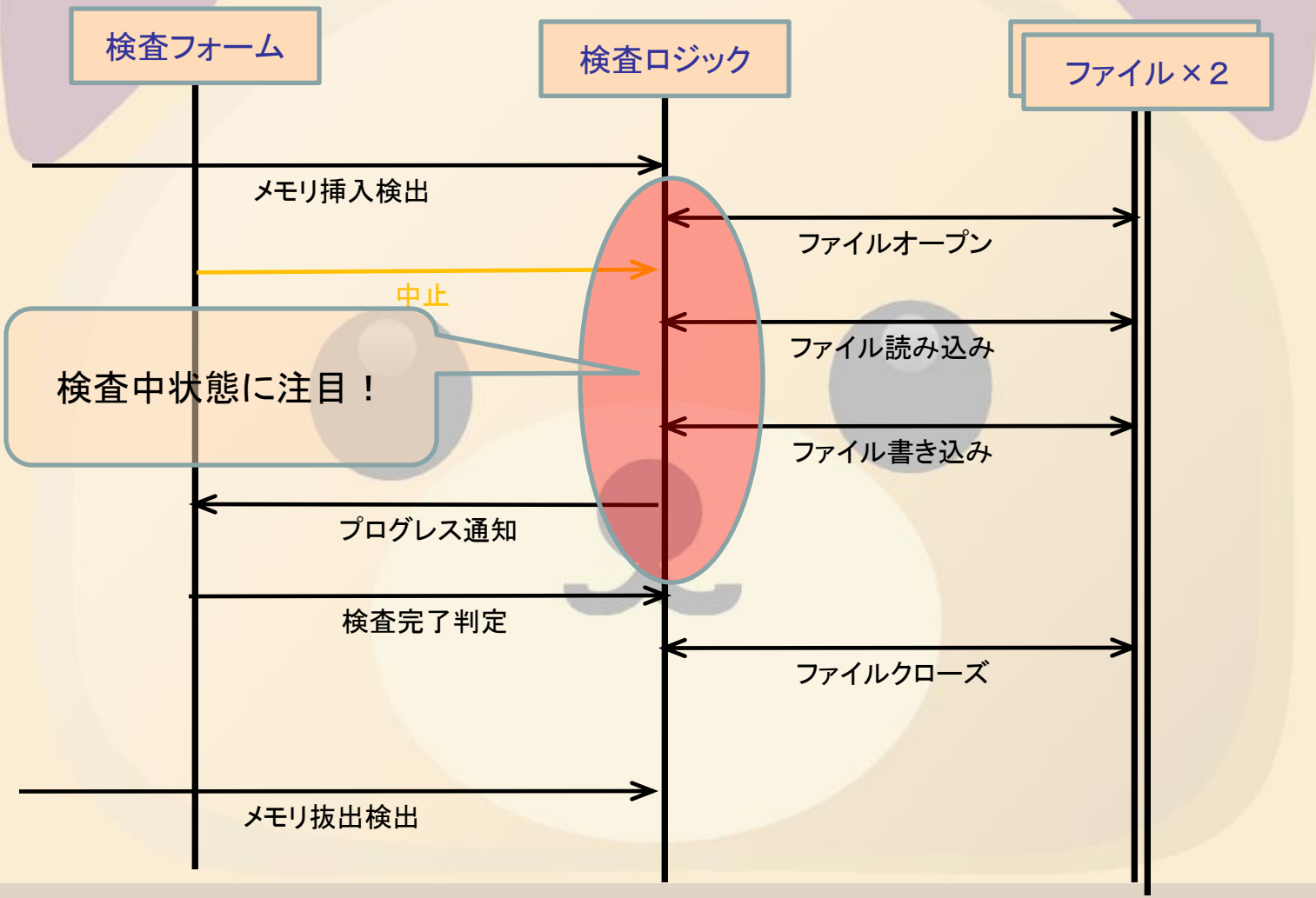
```
#include <windows.h>
namespace  NUsbChecker {
class File
{
        HANDLE  Handle;
public:
        File();
        ~File();
        bool Open( const char *filename_,…);
        bool Close();
        bool Read( char *buffer, DWORD readbytes, DWORD *result );
        bool Write( char *buffer, DWORD readbytes, DWORD *result );
        bool Cancel();
};
} // endof namespace NUsbChecker
```

こんなかんじ。
（検証してませんw）

イベントトレースで考えてみる



検査システムの部分。


検査フォーム
検査ロジック
ファイル×２
メモリ挿入検出
ファイルオープン
中止
検査中状態に注目！
ファイル読み込み
ファイル書き込み
プログレス通知
検査完了判定
ファイルクローズ
メモリ抜出検出

イベントトレースを書いたら？

状態遷移表でしょ？

また書くの？

めんどくさいので、飛ばしましょう。

検査ロジックのクラスを考える。

```
class UsbCheckerLogic
{
	;
public:
	 UsbCheckerLogic();
	~ UsbCheckerLogic();
	bool OnConnectUSB();
	bool OnDisconnectUSB();
	bool OnReadFile();
	bool OnWriteFile();
	 bool OnCancel();
};
```

こんなかんじ？

もうちょっと考察

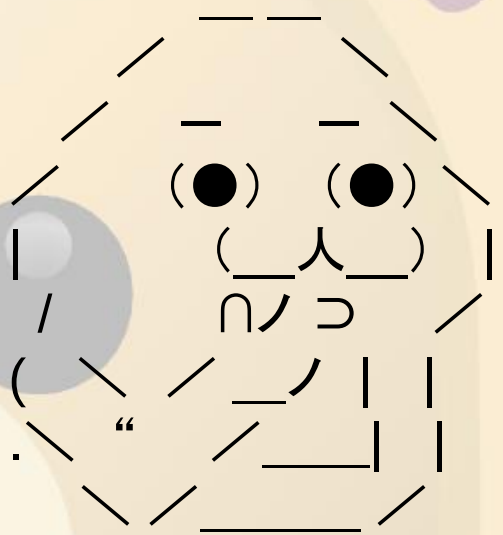
```
bool USBCheckerLogic::OnConnectUSB()
{
    // ここで、検査を開始する・・・
}
bool USBCheckerLogic::OnReadFile()
{
    // ここで、読み込み処理を実施する？
}
bool USBCheckerLogic::OnWriteFile()
{
    // ここで、書込処理を実施し、プログレス経過を通知する？
}
```

どのように受け取って来る？

誰に対して？フォームはどこ？

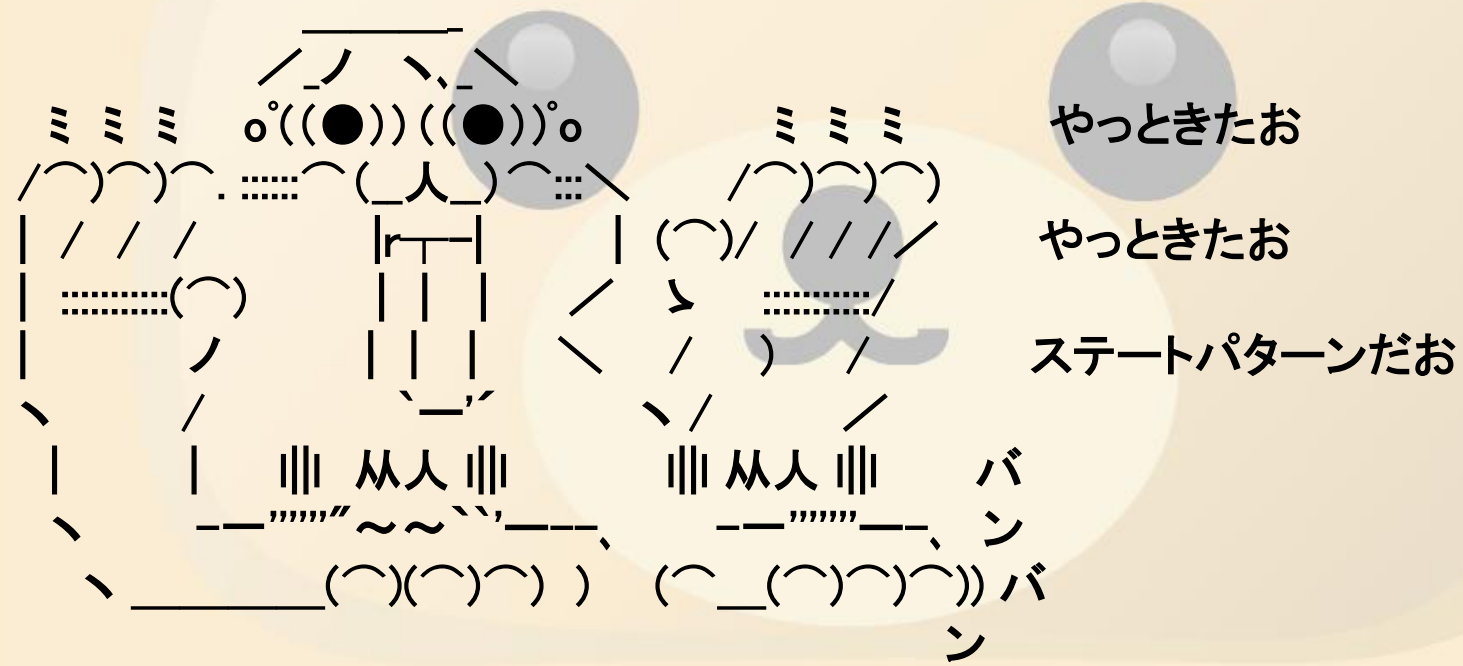
検査中状態に注目。

- 分断したループはどこへ・・・？？？
- ファイルを読み込んだ・・・
  - 中断しているか？
- ファイルを書き込んだ・・・
  - 中断しているか？
  - プログレス経過表示
  - まだ続きを読み込む必要があるのか？

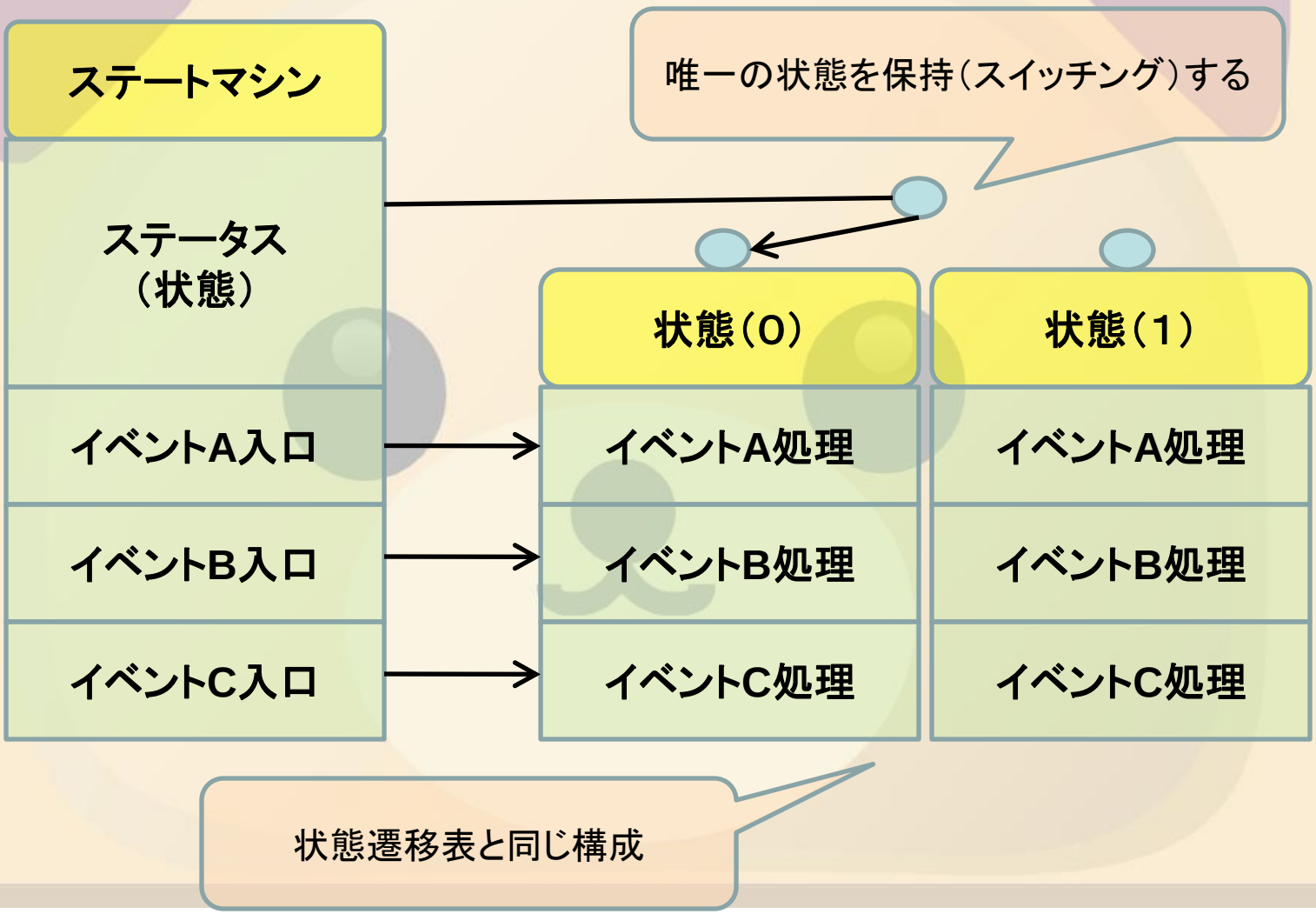


状態遷移の実装方法

ステートパターンをつかう。



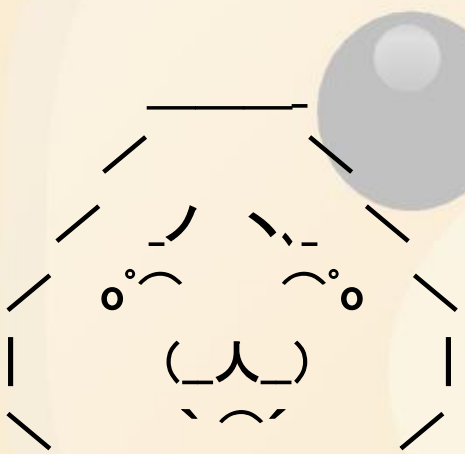
ステートパターンの構成要素



状態遷移表を確認しておきましょう

状態 / イベント	メモリ挿入待ち (0)	検査中 (1)	取り外し待ち (2)
メモリ挿入検出	ファイルオープン 状態を検査中に →(1)読込通知	エラー表示 検査中断 →(2)	ログ記録 ファイルオープン →(1)
ファイル読込	ログ記録 →(0)	ファイル読込 書込通知→(1)	ログ記録 →(2)
ファイル書込	ログ記録 →(0)	経過通知 完了なら完了通知 未完なら読込通知 →(1)	ログ記録 →(2)
検査完了	ログ記録 →(0)	検査結果を表示 ファイルクローズ →(2)	ログ記録 →(2)
メモリ取り外し検出	ログ記録 →(0)	エラー表示 検査中断→(0)	次の検査準備 →(0)
中断	→(0)	ファイルクローズ→(2)	→(2)

コードの例



うれしいお。いただきますお。

状態遷移用の基底クラス

```
class USBCheckerStateMachine;
class USBCheckerState
{
    const class USBCheckerStateMachine *StateMachine;
public:
    USBCheckerState(USBCheckerStateMachine *state_machine_ )
        : StateMachine( state_machine_ ) { }
    virtual ~USBCheckerState (){}

    virtual void OnConnectUSB();        //メモリ挿入検出時の処理
    virtual void OnDisconnectUSB();     //メモリ抜出検出時の処理
    virtual void OnReadFile();          //ファイル読み込み完了時の処理
    virtual void OnWriteFile();         //ファイル書込完了時の処理
    virtual void OnCheckCOmplete();     //検査官了検出時の処理
    virtual void OnCancel();            //中断されたときの処理
};
```

メモリ挿入待ち状態

```
#include “USBCheckerState.h”
class USBWaitingState : public USBCheckerState
{
  // コンストラクタ、デストラクタ、および仮想関数の一覧
};
void USBWaitingState::OnConnectUSB()
{
   if( OpenFiles() == true ) // ファイルをオープンする
   {
      // ステートマシンに検査中状態に変化するように指示
      StateMachine->ChangeState( STATUS_CHECKING );
      // 変化した状態に対して、ファイルの読み込みを通知
   }
   else
     //略
}
```

検査中状態

```
#include “USBCheckerState.h”
class USBCheckingState : public USBCheckerState
{
  // コンストラクタ、デストラクタ、および仮想関数の一覧
};

void USBCheckingState ::OnReadFile()
{
   if( ReadFile() == true ) // ファイルを読み込む
   {
      // ステートマシンにファイルの書込要求を通知
   }
   else
   {
       //エラー処理
   }
}
```

```
void USBCheckingState ::OnWriteFile()
{
   if( WriteFile() == true ) // ファイルを書き込む
   {
      if( IsEndOfFile() == false )
      {
         // ステートマシンに次のブロックを読み込むよう要求通知
      }
      else
      {
         // ステートマシンに完了を通知
      }
   }
   else
   {
       //エラー処理
   }
}
```

以下、略。

ステートマシン側のコード概略

```
class USBWaitingState;
class USBCheckingState;
class USBReleaseWaitingState;
class USBCheckerStateMachine
{
  static  USBWaitingState          *WaitingState;
  static  USBCheckingState         *CheckingState;
  static  USBReleaseWaitingState   *ReleaseWaitingState;

  USBCheckerState  *Status;

  void OnConnectUSB()
  {
     Status->OnConnectUSB();
   以下略
```

状態別のインスタンスをひとつずつ持つ

現在の状態のインスタンス
（上記のいずれかになる）

現在の状態のインスタンスにイベントを通知

検査ロジックは、だいたいこうなる。

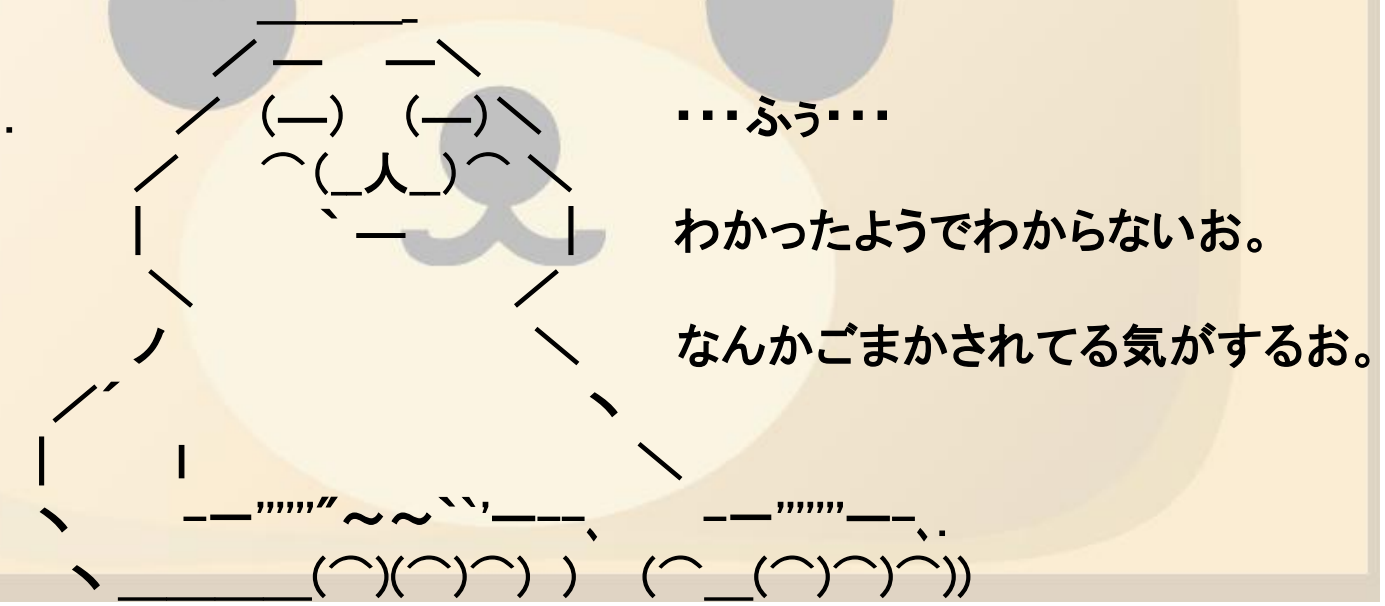
```
class UsbCheckerLogic
{
        USBCheckerStateMachine *StateMachine;
            File *Source, *Dest;

            以下略（略すぎだっつうの（笑）
```

こんなかんじ

と、いうことで。

検査システムのクラスは、
USBCheckerStateMachine

のインスタンスを持つことで、状態遷移を管理できるようになります。



ところで。

次のファイル読み込みとか、書き込み、検査の完了あるいはキャンセルの要求って、どのようにして通知したり受け取ったりすればいいのでしょうか？

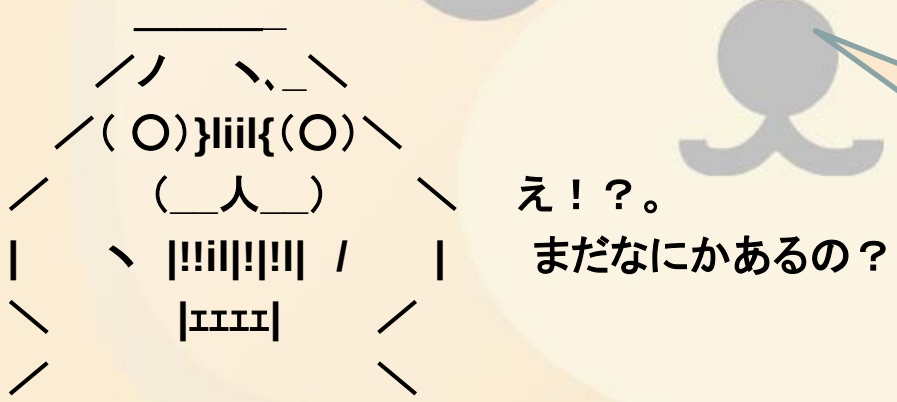
時間軸に対するイベントによって状態を変化させるオブジェクトには、通知の受け口が必要になります。

イベントで状態遷移するクラスには

オブザーバを置く

他のオブジェクトに通知するなら

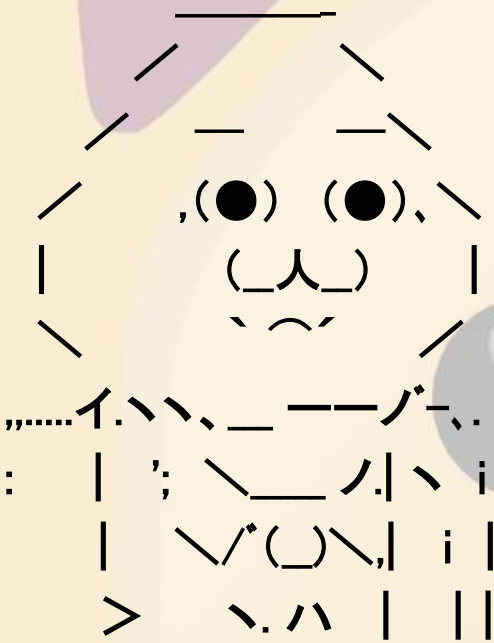
オブザーバコレクションを置く



オブザーバ・パターン

オブザーバ・パターン

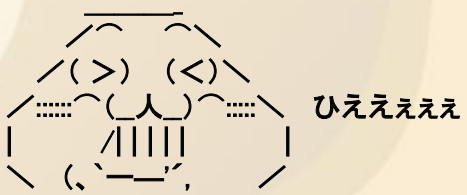
いやな予感がしてきたお。



残念ながら、時間が来たようです。

次回、再利用可能なクラスの作成と状態遷移の

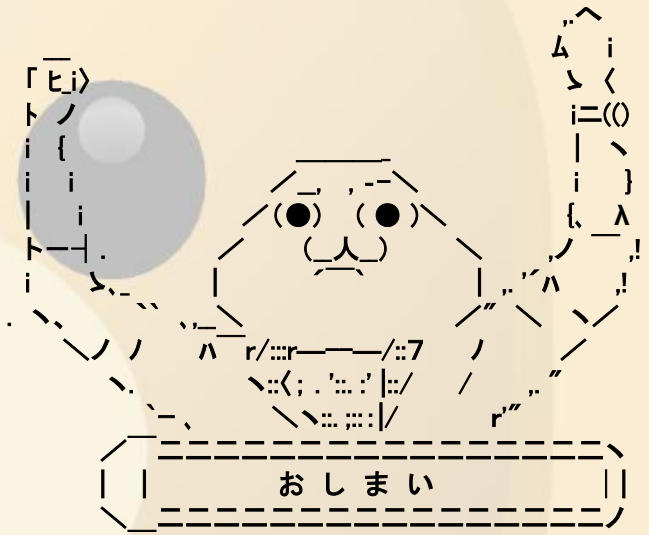
実装の全貌をご覧に入れましょう。



ご静聴ありがとうございました。

m(_._)m

また今度だお



単体のコピーはどうなったのかについて（w

Special thanks for Yaruo charactors