

あのキーボード用の キーリマッパ

中間報告と開発テクニック

* キーリマップとは

* キーコードの入替えを行うソフト

* なぜ新たに作るのか

* 2つのキーボードの入力を1つに
出来ないから

進捗報告

	実証用コード	本番用コード
kext	2/3 くらい	未着手
スタートアップ アイテム	未着手	未着手
設定 アプリ	ON/OFF だけ実装	未着手
バックグラウンド アプリ	未着手	未着手
インストーラ	未着手	未着手

作り方

基本方針

- * コントロールパネルは使わない
- * 複雑な**UI**になると予想されるから
- * **kext**に入れるコードは最小限にする
- * デバッグが困難なため
- * 複数ユーザーには対応しない

作るコード達

- * 設定アプリ
- * スタートアップアイテム
- * kext
- * バックグラウンドアプリ
- * インストーラ & アンインストーラ

設定アプリ

- * ON/OFF

- * `IOConnectCall_XXXX_Method`でkext
と通信する

- * カスタマイズ機能の提供

- * バックグラウンドアプリと
`UserDefault`経由で通信する

スタートアップ項目

- * kextのload&unload

- * バックグラウンドアプリの起動と監視

kext

- * フックを掛ける
- * キー入力をバックグラウンドアプリケーションへ渡す
- * バックグラウンドアプリから変換結果を貰ってキーボード入力とする

バックグラウンドアプリ

- * **kext**から受け取ったキーコードを変換して**kext**のキューに積む
- * **UserDefault**経由で設定を読み込み
- * **IOConnectCall_XXXX_Method**で**kext**と通信

インストーラとか

* **PackageMaker**で作るか設定アプリの中に仕込むかは検討中

* *.dmgで配布か*.zipで配布かは検討中

何で作るか

設定アプリ	通常の Cocoa アプリ
スタートアップアイテム	シェルスクリプトで作る。
kext	Embedded C++
バックグラウンドアプリ	Cocoa アプリだが、エージェントと呼ばれる形式になるハズ
インストーラ等	PackageMaker + シェルスクリプト

Kextの作り方

- * デバッグは**IOLog**が基本
- * **c++**なので**NULL**ポインター禁止
- * **libkern**で定義されるコレクションクラスを使う。**STL**は使えない。
- * クラス設計は**Objective-C**っぽい
- * **CoreFoundation**を使ったプログラミングに似てる
- * **auto release pool** は無いので気をつける!!

Embedde C++

- * **template**無し、例外無し、**RTTI**無し
- * **RTTI**の代わりにマクロあり
- * **STL**が使えないのでコレクションクラスを使う
- * 例外無しは気合いで乗り切る
- * なぜか名前空間が使える。
- * **operator new**はメモリ確保に失敗した時に例外を投げるので本来使ってはダメなはず。

サブクラス化の方法1

```
#ifndef __minimal01__minimal01__ // インクルードガード
#define __minimal01__minimal01__ // objcを書いてると忘れる

#include <IOKit/IOService.h>
#include <IOKit/IOLib.h>

// クラス名が長いのは逆DNS名を使うから。カーネル空間で名前が衝突しないようにします
class com_mindto01_minimal01_driver : public IOService
{
// これがRTTIのマクロです。
    OSDeclareDefaultStructors(com_mindto01_minimal01_driver)
public:
// 指定イニシャライザみたいな物です。
    virtual bool        init(OSDictionary *dictionary = 0);
// デストラクタ
    virtual void        free(void);
protected:
};
// 長い名前を使いたくないのでtypedefすると楽。でもマクロには使えないので気をつける
typedef com_mindto01_minimal01_driver SKDriver;

#endif /* defined(__minimal01__minimal01__) */
```

最初期のobjectシステムに似てる(TCL, ET++, MacAPP....)

サブクラス化の方法2

```
#include "SKDriver.h"
#include "KE_Log.h"

// RTTIマクロの為の設定
#define super IOService
OSDefineMetaClassAndStructors(com_mindto01_minimal01_driver, IOService)

bool
SKDriver::init( OSDictionary *dict )
{
    bool theResult;

    theResult = super::init(dict);

    // KE_Enterマクロの中で、メタクラス名 (this->getMetaClass()->getClassName()) を使っているため、
    // super::init();呼び出し前は呼出すとOS丸ごと落ちる。
    KE_Enter();
    KE_Exit();

    return theResult;
}

void
SKDriver::free(void)
{
    KE_Enter();

    // KE_Exitマクロの中で、メタクラス名 (this->getMetaClass()->getClassName()) を使っているため、
    // free();呼び出し後は呼出すとOS丸ごと落ちる。
    KE_Exit();

    super::free();
}
```


ログを出す

IOLogが基本でも使いにくいので適当にマクロを作る

```
#define KE_Enter()    IOLog("%s(%p)::%s Enter\n", this->getMetaClass()->getClassName(), this, __FUNCTION__);
#define KE_Exit()     IOLog("%s(%p)::%s Exit\n", this->getMetaClass()->getClassName(), this, __FUNCTION__);

#define KE_Trace_Int(inValue) IOLog("%s(%p)::%s | %s = %d\n", \
                                   this->getMetaClass()->getClassName(), this, __FUNCTION__, #inValue, inValue);

#define KE_Trace_Hex(inValue) IOLog("%s(%p)::%s | %s = 0x%x\n", \
                                   this->getMetaClass()->getClassName(), this, __FUNCTION__, #inValue, inValue);

#define KE_Trace_Bool(inBool) IOLog("%s(%p)::%s | %s = %s\n", \
                                   this->getMetaClass()->getClassName(), this, __FUNCTION__, #inBool, \
                                   inBool ? "TRUE" : "FALSE");

#define KE_Trace_Str(inStr) IOLog("%s(%p)::%s | %s = %s\n", \
                                   this->getMetaClass()->getClassName(), this, __FUNCTION__, #inStr, inStr);

#define KE_Trace_Ptr(inPtr) IOLog("%s(%p)::%s | %s = %p\n", \
                                   this->getMetaClass()->getClassName(), this, __FUNCTION__, #inPtr, inPtr);

#define KE_Trace_Msg(inMsg) IOLog("%s(%p)::%s | %s\n", \
                                   this->getMetaClass()->getClassName(), this, __FUNCTION__, inMsg);

#define KE_Trace_Block(inBlockPtr, inBlockLen) _KE_Trace_Block( this->getMetaClass()->getClassName() , this, \
                                                                __FUNCTION__, #inBlockPtr, \
                                                                inBlockPtr, \
                                                                inBlockLen);
```


コレクションクラス

- * `NSArray`, `NSDictionary`, `IOSet`....がある。
- * `NSString`, `OSSymbol`, `OSNumber`....がある。
- * `NSObject`のサブクラスだけ入れられる
- * `OSSymbol`は同じ文字列ならば常に同じポインター値になる。
`NSDictionary`に使う。
- * コレクションに追加するとリテイン、削除するとリリース。
- * コレクションで`flushCollection()`で全てのオブジェクトをリリース。
- * コレクションクラスをリリースしても、所有オブジェクトはリリースされていないようだ←自身がない。
- * `plist`にシリアライズ&アンシリアライズ出来る

イディオム

- * まずは**NULL**チェック
- * しないと**OS**道連れで落ちる **onZ**

```
SKRemapperMgr* theRemapperMgr = SKRemapperMgr::sharedInstance();  
  
if( theRemapperMgr ) // どんなときでもNULLチェック  
{  
    theRemapperMgr->setGlobalSwitch(true);  
}
```


イディオム

- * `OSObject`にはリテンカウントがある
- * でも `auto release pool`はない

```
// NULLチェック後にリリースして変数にNULLを入れるマクロ  
OSSafeReleaseNULL(symbol);
```

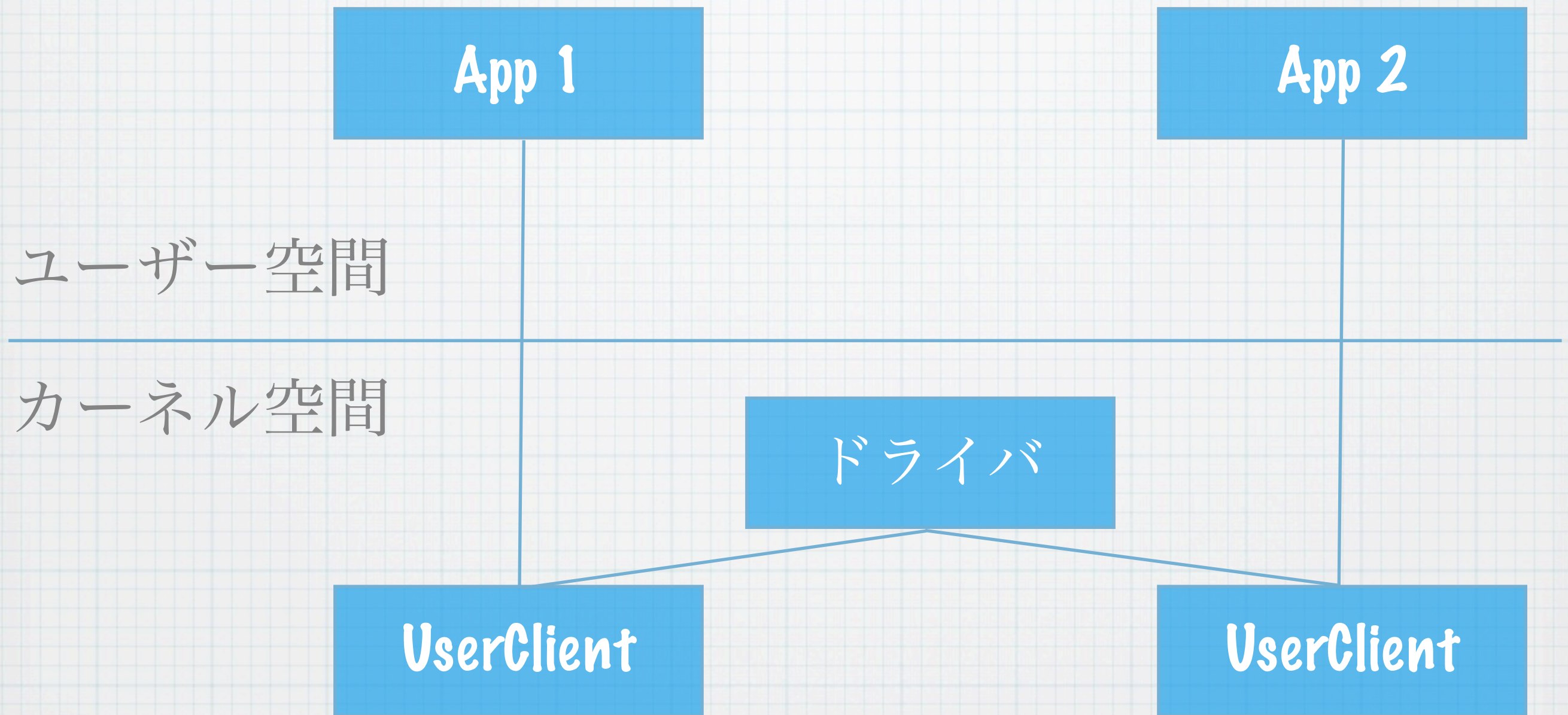
- * `autorelease`は無いので、所有権の取得と放棄は手書き

```
// クラス名::withXXXXで作成した時にリテンカウントは1  
theObject = SKKeyboardRemaper::withKeyboard(theHIKeyboard);  
  
// コレクションクラスに追加したらリテンカウントは2  
keyboardDict->setObject( theSymbol, theObject );  
  
// コレクションに追加したのでリリースして、リテンカウントを1にする  
theObject->release();
```


Kextとuser空間の通信

- * **UserClient**のサブクラスを作る
 - * **API**を適当に作る
 - * デスパッチテーブルを作る
- * **info.plist**に登録
- * **NSInvocation**を手書きて書いてる感じ

アプリとkextの通信



* **userClient**が通信経路毎にインスタンスが作られる

アプリ側から見てこんな**API**つくりたい

-(BOOL)open	通信経路を開く
-(void)close	通信経路を閉じる
-(void) setGlobalSwitch:(BOOL)	全機能を ON/OFF
-(BOOL) globalSwitch	全機能の ON/OFF を取得

SEL型値に対応する物をつくる
kextと**user App**で共有する数値です

```
// User client method dispatch selectors.  
enum  
{  
    kSKUserClientOpen,  
    kSKUserClientClose,  
  
    kSKUserClientGlobalSwitch,  
    kSKUserClientSetGlobalSwitch,  
  
    kMyScalarIStruct0Method,  
  
    kNumberOfMethods // Must be last  
};
```


アプリ側はIOConnectCall_XXXをつかって呼出す

```
- (void) setGlobalSwitch:(BOOL)inYesNo
{
    if( [self isOpen] )
    {
        uint64_t scalarI_64[1];

        scalarI_64[0] = (inYesNo == YES);

        IOConnectCallScalarMethod(_ioConnect,
                                   kSKUserClientSetGlobalSwitch,
                                   scalarI_64,
                                   1,
                                   NULL,
                                   NULL);
    }
}
```

```
- (BOOL) globalSwitch
{
    if( [self isOpen] )
    {
        kern_return_t theResult;

        uint64_t scalar0_64;
        uint32_t outputCount = 1;

        theResult = IOConnectCallScalarMethod(_ioConnect,
                                                kSKUserClientGlobalSwitch,
                                                NULL,
                                                0,
                                                &scalar0_64,
                                                &outputCount);

        return ( scalar0_64 != 0 );
    }

    return NO;
}
```


kext側はIOUserClientのサブクラスを作る

```
class com_mindto01_minimal01_userClient : public IOUserClient
{
    OSDeclareDefaultStructors(com_mindto01_minimal01_userClient)
protected:
    SKDriver*          fProvider;
    task_t             fTask;
    static const IOExternalMethodDispatch sMethods[kNumberOfMethods]; <--- ここで
public:
    .
    .
    .
protected:
    virtual IOReturn externalMethod(uint32_t selector,
                                     IOExternalMethodArguments* arguments,
                                     IOExternalMethodDispatch* dispatch,
                                     OSObject* target,
                                     void* reference); <--- ここに注目

    //
    static IOReturn sGlobalSwitch(com_mindto01_minimal01_userClient* target,
                                   void* reference,
                                   IOExternalMethodArguments* arguments);

    virtual bool    globalSwitch(void);

    static IOReturn sSetGlobalSwitch(com_mindto01_minimal01_userClient* target,
                                      void* reference,
                                      IOExternalMethodArguments* arguments);

    virtual void    setGlobalSwitch(bool inBool);
};

typedef com_mindto01_minimal01_userClient SKUserClient;
```

API関数へのポインタの配列を定義する

```
//
const IOExternalMethodDispatch SKUserClient::sMethods[kNumberOfMethods] =
{
    { // kSKUserClientOpen
      (IOExternalMethodAction) &SKUserClient::sOpenUserClient,
      0,
      0,
      0,
      0
    },
    { // kSKUserClientClose
      (IOExternalMethodAction) &SKUserClient::sCloseUserClient,
      0,
      0,
      0,
      0
    },
    { // kSKUserClientGlobalSwitch
      (IOExternalMethodAction) &SKUserClient::sGlobalSwitch,
      0,
      0,
      1,
      0
    },
    { // kSKUserClientSetGlobalSwitch
      (IOExternalMethodAction) &SKUserClient::sSetGlobalSwitch,
      1,
      0,
      0,
      0
    },
};
```


先ほどの配列を利用してそれぞれの関数へ飛ばす

```
IOReturn
SKUserClient::externalMethod(uint32_t selector,
                              IOExternalMethodArguments* arguments,
                              IOExternalMethodDispatch* dispatch,
                              OSObject* target,
                              void* reference)
{
    if (selector < (uint32_t) kNumberOfMethods)
    {
        dispatch = (IOExternalMethodDispatch *) &sMethods[selector];

        if (!target)
        {
            target = this;
        }
    }

    return super::externalMethod(selector, arguments, dispatch, target, reference);
}
```

静的関数の先で**this**ポインタ経由でメソッドへ

```
IOReturn
SKUserClient::sSetGlobalSwitch(com_mindto01_minimal01_userClient* target,
                                void* reference,
                                IOExternalMethodArguments* arguments)// これはstaticメソッドです
{
    IOReturn theResult = kIOReturnSuccess;

    SKUserClient* theObject = OSDynamicCast(SKUserClient, target);// targetがobjectなので

    if( theObject != NULL )
    {
        bool theBool = !(arguments->scalarInput[0] == false);

        theObject->setGlobalSwitch(theBool);// こうやって、メソッドへ渡す
    }

    return theResult;
}
```

```
void
SKUserClient::setGlobalSwitch(bool inBool)
{
    SKRemapperMgr* theRemapperMgr = SKRemapperMgr::sharedInstance();

    if( theRemapperMgr )
    {
        theRemapperMgr->setGlobalSwitch(inBool);
    }
}
```


終わり