embarcadero
Developer Camp

【T4】C++Builderテクニカルセッション

「C++BuilderによるWebサービス&マルチスレッド対応リソースプールの設計」

エンバカデロ・テクノロジーズ
エヴァンジェリスト
高橋智宏

アジェンダ

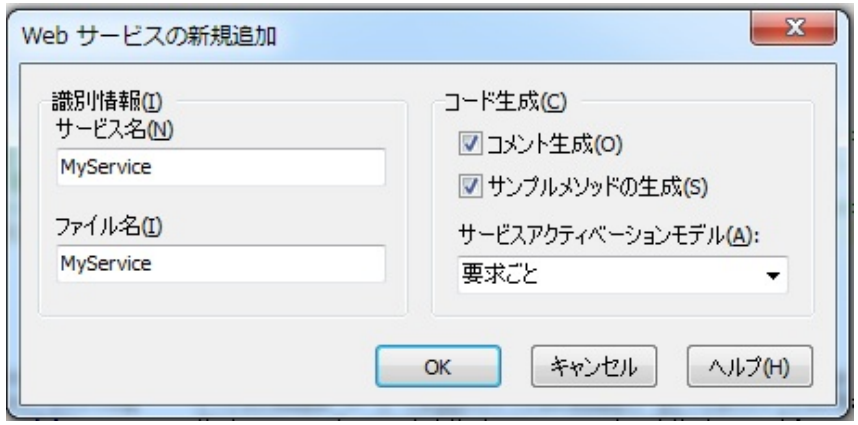
- Webサービス（CGI版）
  - BASIC認証（IIS）
- SOAPクライアント
  - WSDLインポータ, BASIC認証対応
- Webサービス（スタンドアロン版）
  - 独自のBASIC認証を実装
- リソースプール
  - データモジュール（データベース接続）
  - Win32 API
  - POSIXスレッド
  - Boost::Thread

embarcadero
Developer Camp

Webサービスの構築

CGI版のWebサービス

- [C++Builderプロジェクト]-[Webサービス]-[SOAPサーバーアプリケーション]
  - CGI実行形式
- [C++Builderプロジェクト]-[Webサービス]-[SOAPサーバーインターフェイス]
  - サービス名: MyService
  - サンプルメソッドの生成: On



- 注意点1: プロジェクトソース(例:Project1.cpp)が少し壊れることがある
- 注意点2:サンプルのTSampleStructクラスのメンバがAnsiString型

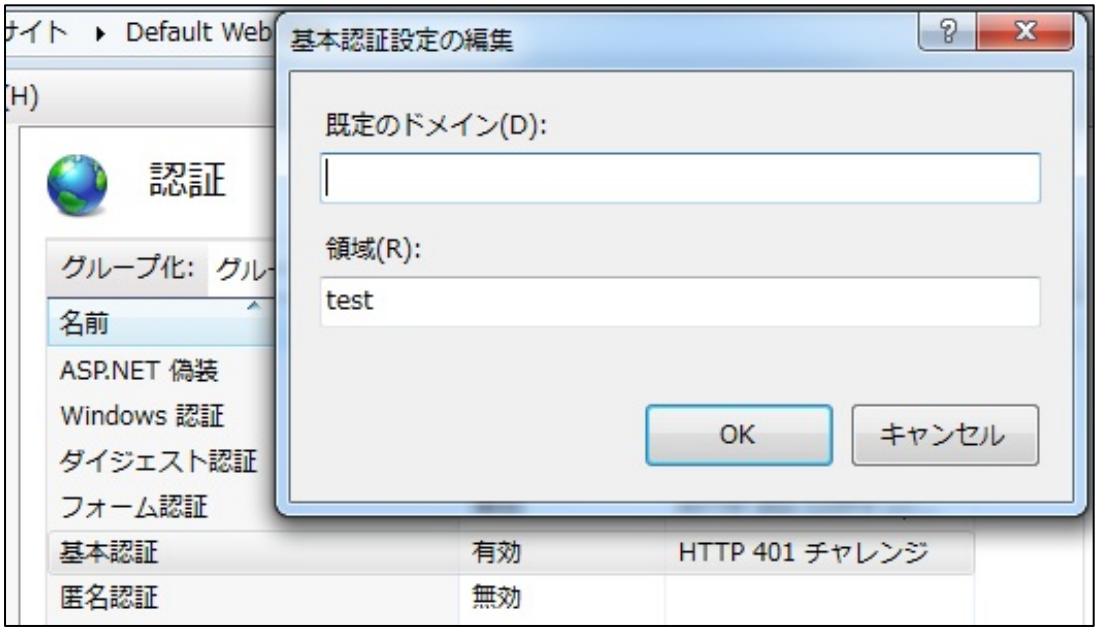
Webサービスメソッドの実装のポイント

- echoStructメソッドの実装
  - TRemotableの派生クラスを処理する
  - クラスのインスタンスのconstポインタを受け取り
  - クラスのインスタンスのポインタを返す
    - returnしたインスタンスは後でdeleteされる
    - NULLを返すのも一応OK
    - 例外はすべて捕捉して伝播させない（SOAPの例外機構は貧弱なので）

```
TSampleStruct* TMyServiceImpl::echoStruct(const TSampleStruct* pEmployee)
{
  TSampleStruct* retval = NULL;
  try {
    retval = new TSampleStruct();
    retval->LastName = pEmployee->LastName;
    retval->FirstName = pEmployee->FirstName;
    retval->Salary = pEmployee->Salary;
  }
  catch (...) {
    ; // do something
  }
  return retval;
}
```

IISにCGIを配布 ー BASIC認証

- CGIを利用できるように設定
  - ハンドラマッピング, CGIの制限 etc…
- WebサイトでBASIC認証を有効にする
  - 基本認証（HTTP 401チャレンジ）: 有効
    - 既定のドメイン:
    - 領域（Realm,レルム）: 例として test を設定
  - 匿名認証: 無効
- WSDLを確認
  - Webブラウザ

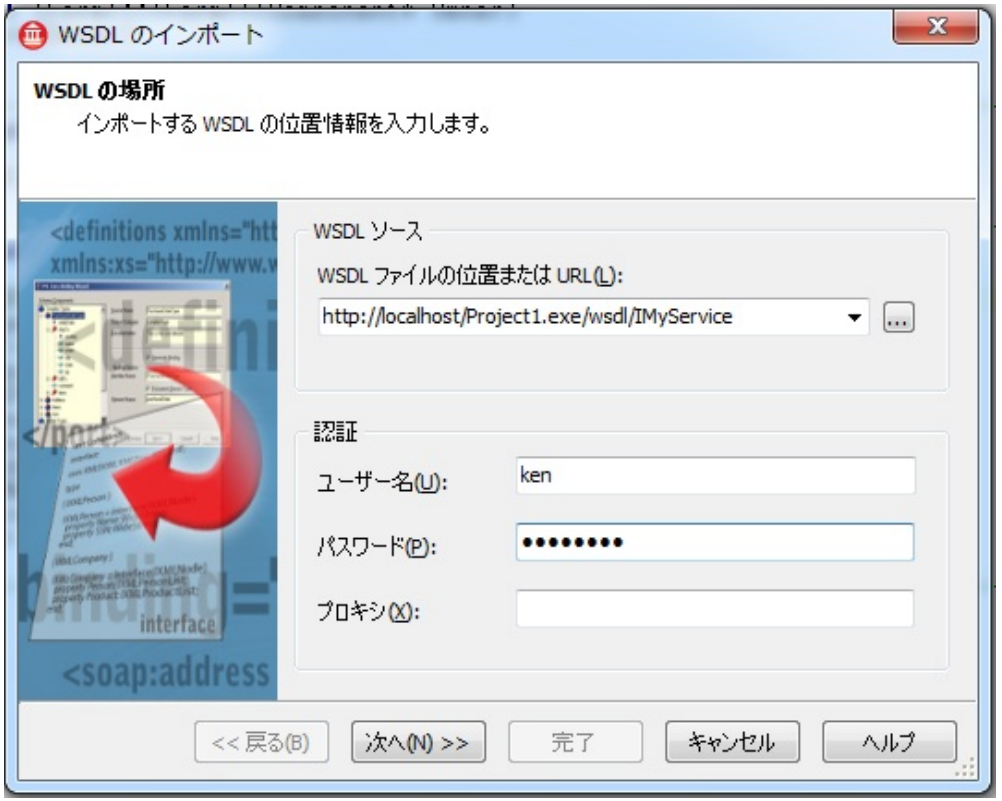


embarcadero
Developer Camp

SOAPクライアントの構築

WSDLインポータ

- C++アプリケーションプロジェクトの作成
- [C++Builderプロジェクト]-[Webサービス]-[WSDLインポータ]
  - WSDLソース
  - 認証: ユーザー名
  - 認証: パスワード
- 自動生成されるファイル
  - IMyService.h
  - IMyService.cpp
- プロキシの取得
  - GetIMyService関数



クライアント側でのBASIC認証への対応

- THTTPRIOクラスのHTTPWebNodeプロパティを使用
  - UserNameプロパティ
  - Passwordプロパティ

```
_di_IMyService GetIMyService(bool useWSDL, System::String addr,
                             Soaphttpclient::THTTPRIO* HTTPRIO)
{
  static const char* defWSDL= "http://localhost/Project1.exe/wsdl/IMyService";
  static const char* defURL = "http://localhost/Project1.exe/soap/IMyService";
  static const char* defSvc = "IMyServiceservice";
  static const char* defPrt = "IMyServicePort";
  if (addr=="")
    addr = useWSDL ? defWSDL : defURL;
  Soaphttpclient::THTTPRIO* rio = HTTPRIO ? HTTPRIO : new Soaphttpclient::THTTPRIO(0);

  // もし第２パラメータがNULLならデフォルトのユーザー名・パスワードを使用
  if( HTTPRIO == NULL ) {
    rio->HTTPWebNode->UserName = "ken";
    rio->HTTPWebNode->Password = "password";
  }

  if (useWSDL) {
```

- 注意: C++Builder XE（Delphi XE）では、必要なら、認証ダイアログを自前で用意して表示させる必要がある
  - 以前のバージョンでは、自動で表示されてしまっていたので要注意
    - QC#91848

クライアントコードの実装のポイント ー その1

- プロキシの取得は1回だけ
  - メモリリークあり: QC#91160
  - THTTPRIOクラスのインスタンスは自前で用意
    - BASIC認証のユーザー名・パスワードを設定
    - deleteは不要

```
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  httprio = new Soaphttpclient::THTTPRIO(NULL);
  httprio->HTTPWebNode->UserName = "ken";
  httprio->HTTPWebNode->Password = "password";
  ws = GetIMyService(false, "", httprio);
}
```

クライアントコードの実装のポイント – その2

- Webサービスメソッドの呼び出しは常に失敗する可能性がある
  - try … catch を忘れずに!!
- Webサービスメソッドの戻り値のNULLチェック
- パラメータと戻り値のそれぞれのインスタンスをdelete
  - try … __finally を忘れずに!!

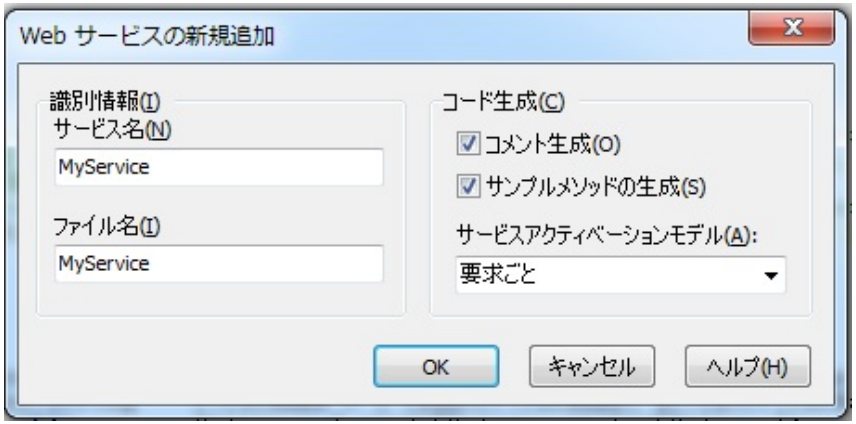
```
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  TSampleStruct* data = new TSampleStruct();
  data->LastName = L"ラスト侑好";
  data->FirstName = L"ファースト侑好";
  data->Salary = 123.456;
  TSampleStruct* retval = NULL;
  try {
    try {
      retval = ws->echoStruct(data);
      if( retval != NULL ) {
        ShowMessage(retval->LastName);
      }
    }
    catch (const ESOAPHTTPException& ex) {
      if( ex.StatusCode == 401 ) {
        ; // BASIC認証に失敗した
      }
    }
    catch (...) {
      ; // do something
    }
  }
  __finally {
    if( data != NULL )
      delete data;
    if( retval != NULL )
      delete retval;
  }
}
```

embarcadero
Developer Camp

スタンドアロン版 Webサービスサーバー

スタンドアロン版のWebサービス

- [C++Builderプロジェクト]-[Webサービス]-[SOAPサーバーアプリケーション]
  - Indy VCL（or コンソール）アプリケーションを選択
- [C++Builderプロジェクト]-[Webサービス]-[SOAPサーバーインターフェイス]
  - サービス名: MyService
  - サンプルメソッドの生成: On



- 注意点1: プロジェクトソース(例:Project1.cpp)が少し壊れることがある
- 注意点2:サンプルのTSampleStructクラスのメンバがAnsiString型

サービスアクティベーションモデル

- 単一プロセスのWebサーバーなので、クラスインスタンスのライフサイクルに注意が必要
- グローバル：単一インスタンスですべて処理（スレッドセーフ必須）

```
static void __fastcall MyServiceFactory(System::TObject* &obj)
{
  static _di_IMyService iInstance;
  static TMyServiceImpl *instance = 0;
  if (!instance)
  {
    instance = new TMyServiceImpl();
    instance->GetInterface(iInstance);
  }
  obj = instance;
}

// ************************************************************************ //
//  以下のルーチンでは、インターフェイスと実装クラスを登録する
//  さらに、インターフェイスのメソッドで使用される型も実装する
// ************************************************************************ //
static void RegTypes()
{
  InvRegistry()->RegisterInterface(__delphirtti(IMyService));
  InvRegistry()->RegisterInvokableClass(__classid(TMyServiceImpl), MyServiceFactory);
}
```

- 要求ごと：処理のたびに new & delete が繰り返される

```
// ************************************************************************ //
//  以下のルーチンでは、インターフェイスと実装クラスを登録する
//  さらに、インターフェイスのメソッドで使用される型も実装する
// ************************************************************************ //
static void RegTypes()
{
  InvRegistry()->RegisterInterface(__delphirtti(IMyService));
  InvRegistry()->RegisterInvokableClass(__classid(TMyServiceImpl));
}
```

スタンドアロン版 ー BASIC認証

- メインフォームで使用されているIndyクラスがポイント
  - TIdHTTPWebBrokerBridge
    - DoCommandGet, DoCommandOther がHTTPリクエストを処理している
- TIdHTTPWebBrokerBridgeの派生クラスに取り替える
  - DoCommandGet, DoCommandOtherをオーバーライド
  - BASIC認証を実装
  - メインフォームのOnCreateイベントで new する

```
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  //FServer = new TIdHTTPWebBrokerBridge(this);
  FServer = new TIdHTTPWebBrokerBridgeEx(this);
}
```

スタンドアロン版 ― BASIC認証（続き）

- TIdHTTPWebBrokerBridgeExクラスの実装

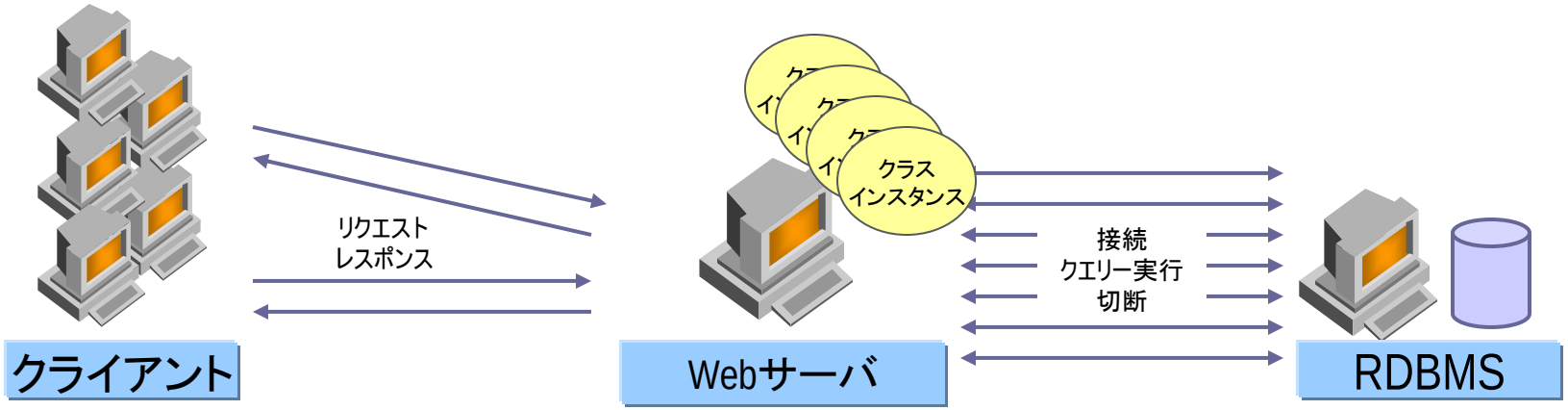
```
#include <IdContext.hpp>
#include <IdCustomHTTPServer.hpp>
class TIdHTTPWebBrokerBridgeEx : public TIdHTTPWebBrokerBridge {
public:
  __fastcall TIdHTTPWebBrokerBridgeEx(TComponent* AOwner) :
        TIdHTTPWebBrokerBridge(AOwner){}
  __fastcall virtual ~TIdHTTPWebBrokerBridgeEx(){}
protected:
  void __fastcall DoCommandGet(TIdContext* AThread,
       TIdHTTPRequestInfo* ARequestInfo, TIdHTTPResponseInfo* AResponseInfo) {
    if( ARequestInfo->AuthExists &&
        ARequestInfo->AuthUsername == "ken" &&
        ARequestInfo->AuthPassword == "password" ) {
        TIdHTTPWebBrokerBridge::DoCommandGet(AThread, ARequestInfo,
AResponseInfo);
        return;
    }
    AResponseInfo->ResponseNo  = 401;
    AResponseInfo->ContentText = "auth error";
    AResponseInfo->AuthRealm   = "test";
  }
  void __fastcall DoCommandOther(TIdContext* AThread,
       TIdHTTPRequestInfo* ARequestInfo, TIdHTTPResponseInfo* AResponseInfo) {
    DoCommandGet(AThread, ARequestInfo, AResponseInfo);
  }
};
```

embarcadero
Developer Camp

マルチスレッドへの対応
リソースプールの設計

有限個のリソースで対応する

- CGIはパフォーマンスの点では劣る
- スタンドアロン版
  - サービスアクティベーションモデルに関係なく、コネクションプール（orデータモジュールのプール）が必要
  - Webサービスメソッド内では…
    1. プールからリソースを借りて
    2. リソースを利用して
    3. プールにリソースを返す



スレッドセーフなプール ― Win32 API版

- CriticalSectionに関するAPI
  - リソースの排他制御（ロック）に使用
    - InitializeCriticalSection(Ex)
    - EnterCriticalSection
    - LeaveCriticalSection
    - DeleteCriticalSection
- ConditionVariableに関するAPI
  - ビジーループや無駄なポーリングを回避するため
  - ただし、Vista / Windows 7 / Windows 2008 でのみ利用可
    - InitializeConditionVariable
    - SleepConditionVariableCS
    - WakeAllConditionVariable
      - WakeConditionVariable

テストコード ー 複数スレッドの起動

- _beginthreadex, WaitForSingleObject, CloseHandle

```
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  InitializeCriticalSectionEx(&cs, 0, CRITICAL_SECTION_NO_DEBUG_INFO);
  InitializeConditionVariable(&cv);

  unsigned tid1, tid2, tid3, tid4, tid5;
  HANDLE t1 = (HANDLE)_beginthreadex(NULL, 0, start, NULL, 0, &tid1);
  HANDLE t2 = (HANDLE)_beginthreadex(NULL, 0, start, NULL, 0, &tid2);
  HANDLE t3 = (HANDLE)_beginthreadex(NULL, 0, start, NULL, 0, &tid3);
  HANDLE t4 = (HANDLE)_beginthreadex(NULL, 0, start, NULL, 0, &tid4);
  HANDLE t5 = (HANDLE)_beginthreadex(NULL, 0, start, NULL, 0, &tid5);
  WaitForSingleObject(t1, INFINITE);
  WaitForSingleObject(t2, INFINITE);
  WaitForSingleObject(t3, INFINITE);
  WaitForSingleObject(t4, INFINITE);
  WaitForSingleObject(t5, INFINITE);
  CloseHandle(t1);
  CloseHandle(t2);
  CloseHandle(t3);
  CloseHandle(t4);
  CloseHandle(t5);

  DeleteCriticalSection(&cs);

  ShowMessage(IntToStr(r)); // r == 1
}
```

テストコード – リソースプールへのアクセス

- try … __finally を忘れずに!!

```
#include <process.h>
static CRITICAL_SECTION cs;    // 排他制御(ロック)
static CONDITION_VARIABLE cv; // Vista/7/2008 でのみ利用可
static volatile int r = 1;     // プール(リソースが1個しかない)
static unsigned __stdcall start(void* arg) {
  EnterCriticalSection(&cs);
  try {
    while( r <= 0 ) {
      DWORD t = 5 * 1000; // 最大待機 = 5秒
      SleepConditionVariableCS(&cv, &cs, t);
    }
    r--; // 借りれた!!
  }
  __finally {
    LeaveCriticalSection(&cs);
  }

  ::Sleep(1000); // リソースを利用

  EnterCriticalSection(&cs);
  try {
    r++; // 返した!!
    WakeAllConditionVariable(&cv);
  }
  __finally {
    LeaveCriticalSection(&cs);
  }
  return 0;
}
```

スレッドセーフなプール ― POSIXスレッド版

- OSに依存しないPOSIXスレッドを利用
  - Open Source POSIX Threads for Win32
    - http://sourceware.org/pthreads-win32/
  - デベロッパーキャンプで紹介済みです

    [gSOAP, Pthreads-Win32, OpenSSLを使ったSOAS C/Sアプリ開発]
    - http://edn.embarcadero.com/article/images/33870/devcamp03_a1.pdf
- 排他制御（ロック）に使用する関数
  - pthread_mutex_init, pthread_mutex_destroy
  - pthread_mutex_lock, pthread_mutex_unlock
- 条件による待機に使用する関数
  - pthread_cond_init, pthread_cond_destroy
  - pthread_cond_timewait, pthread_cond_broadcast

テストコード ー 複数スレッドの起動

- pthread_create, pthread_join

```
void __fastcall TForm1::Button2Click(TObject *Sender)
{
  pthread_mutex_init(&posix_mutex, NULL);
  pthread_cond_init(&posix_cond, NULL);

  pthread_t t1, t2, t3, t4, t5;
  pthread_create(&t1, NULL, execute, NULL);
  pthread_create(&t2, NULL, execute, NULL);
  pthread_create(&t3, NULL, execute, NULL);
  pthread_create(&t4, NULL, execute, NULL);
  pthread_create(&t5, NULL, execute, NULL);
  pthread_join(t1, NULL);
  pthread_join(t2, NULL);
  pthread_join(t3, NULL);
  pthread_join(t4, NULL);
  pthread_join(t5, NULL);

  pthread_mutex_destroy(&posix_mutex);
  pthread_cond_destroy(&posix_cond);

  ShowMessage(IntToStr(r)); // r == 1
}
```

テストコード – リソースプールへのアクセス

```
#include <time.h>
#include "pthread.h"
static pthread_mutex_t posix_mutex; // 排他制御(ロック)
static pthread_cond_t posix_cond;   // 条件による待機
static volatile int r = 1;          // プール(リソースが1個しかない)

#define ADJUST_OFFSET (((__int64)27111902<<32)+(__int64)3577643008)
static void FILETIME_to_timespec(const FILETIME* ft, timespec* ts)
{
  ts->tv_sec = (long)((*(__int64*)ft-ADJUST_OFFSET) / (__int64)10000000);
  ts->tv_nsec = (long)((*(__int64*)ft-ADJUST_OFFSET -
                        ((__int64)ts->tv_sec*(__int64)10000000))*100);
}

static void* execute(void* arg) {
  pthread_mutex_lock(&posix_mutex);
  try {
    while(r <= 0) {
      FILETIME now;
      timespec t;
      ::GetSystemTimeAsFileTime(&now);
      FILETIME_to_timespec(&now, &t);
      t.tv_sec += 5; // 最大待機 = 5秒
      pthread_cond_timedwait(&posix_cond, &posix_mutex, &t);
    }
    r--; // 借りれた!!
  }
  __finally {
    pthread_mutex_unlock(&posix_mutex);
  }

  ::Sleep(1000); // リソースを利用

  pthread_mutex_lock(&posix_mutex);
  try {
    r++; // 返した!!
    pthread_cond_broadcast(&posix_cond);
  }
  __finally {
    pthread_mutex_unlock(&posix_mutex);
  }
}
```

スレッドセーフなプール ― Boost::Thread版

- OSに依存しないスレッドライブラリ
  - C++Builder XE には、Boost 1.39 が付属
  - Boost::ThreadライブラリのDLLが必要
    - リリース用: boost_thread-bcb-mt-1_39.dll
    - デバッグ用: boost_thread-bcb-mt-d-1_39.dll
- 排他制御（ロック）のコードがキレイ
  - boost::unique_lockオブジェクトのスコープ
    - Lockableには、boost::recursive_mutex を使用
- Lockable, Condition Variable の初期化コードが不要
  - boost::condition_variable_anyクラス
    - timed_waitメソッド
    - notify_allメソッド

テストコード ー 複数スレッドの起動

- boost::threadクラス, operator()(), joinメソッド

```
static boost::thread copies_are_safe() {
  callable x;
  boost::thread t(x);
  // x is destroyed, but the newly-created thread has a copy, so this is OK
  return t;
}

void __fastcall TForm1::Button3Click(TObject *Sender)
{
  boost::thread t1 = copies_are_safe();
  boost::thread t2 = copies_are_safe();
  boost::thread t3 = copies_are_safe();
  boost::thread t4 = copies_are_safe();
  boost::thread t5 = copies_are_safe();
  t1.join();
  t2.join();
  t3.join();
  t4.join();
  t5.join();

  ShowMessage(IntToStr(r)); // r == 1
}
```

テストコード ― リソースプールへのアクセス

```
#include <boost/thread.hpp>
static boost::recursive_mutex mutex;          // 排他制御(ロック)
static boost::condition_variable_any cond; // 条件による待機
static volatile int r = 1;                    // プール(リソースが1個しかない)

struct callable {
  void operator()() {
    {
      boost::unique_lock<boost::recursive_mutex> lock(mutex);
      while(r <= 0) {
        boost::xtime t;
        boost::xtime_get(&t, boost::TIME_UTC);
        t.sec += 5; // 最大待機 = 5秒
        try {
          cond.timed_wait(lock, t);
        }
        catch (...) {
        }
      }
      r--; // 借りれた!!
    }

    try {
      boost::this_thread::sleep(boost::posix_time::seconds(1)); // リソースを利用
    }
    catch (boost::thread_interrupted const & ex) {
    }

    {
      boost::unique_lock<boost::recursive_mutex> lock(mutex);
      r++; // 返した!!
      cond.notify_all();
    }
  }
};
```

embarcadero
Developer Camp

デモ
リソースプールの動作検証

embarcadero
Developer Camp

Q & A