【A4】C++テクニカルセッション

# 「暴れん坊のC++をどう使いこなすか」

## 多人数開発現場におけるC++言語の活用の問題点と対策

Java読書会　代表
高橋　徹（torutk@02.246.ne.jp）




CODE GEAR FROM Borland
Where Developers Matter
Integrated Development Environments for Windows, Java, and Web Developers
DEVELOPER CAMP

# 暴れん坊のC++　アジェンダ

- 発表者とJava読書会とC++
- 暴れん坊C++の姿
- 暴れん坊C++に蹂躙される開発現場
- 暴れん坊C++の押さえどころ
- まとめ

# 発表者とJava読書会とC++

発表者のC++との出会いと関わり
Java読書会について






## 発表者略歴

- C++との出会い
  - 1991年、学生時代にファーストコンタクト
    - それまでは、Basic、アセンブラ、C言語、Pascal言語に触った経験のみでオブジェクト指向プログラミング言語は初
- 就職後、仕事としては
  - Java言語中心（20世紀）
  - C++言語中心（21世紀）
    - 大規模なソフトウェア開発プロジェクトをサブリーダークラスの役割で転々としており、自分でコードを書くことはほとんどない
- 会社とは別に技術活動
  - 個人Webページで技術発信（http://www.02.246.ne.jp/~torutk/）
  - Java読書会

# Java読書会

- 活動概要
  - 東京近郊でJavaに関する技術書籍を読む会を開催
  - 1998年12月より現在に至るまで毎月1回土曜日開催
    「いい本があるが、独りで読むのは難しい」が発端
  - 毎回十数人参加
    - 仕事でJavaを使っていない人もある程度の割合で参加
      → C++技術ネタで盛り上がることがしばしば

URL

http://www.javareading.com/bof/




CODE GEAR FROM Borland
Where Developers Matter
Integrated Development Environments for Windows, Java, and Web Developers
DEVELOPER CAMP

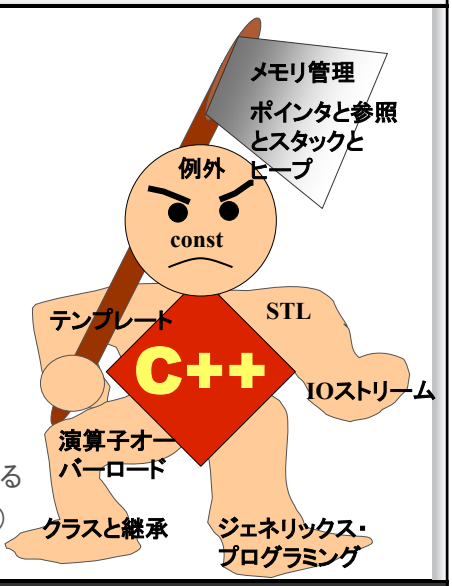
# 暴れん坊のC++の姿

C++言語の特徴
C++言語の難しいところ
C++言語の学習

本文書の一部または全部の転載を禁止します。本文書の著作権は、著作者に帰属します。

# 暴れん坊のC++の姿

- C++超簡略歴史
  - 1983年「C++」命名
  - 1990年ARM出版
  - 1998年ANSI/ISO標準化
- C++の今
  - 標準化対応がひと段落し、効率的な使用方法の蓄積が進んでいる
  - 進歩的な使い方が提唱されている（テンプレート・メタプログラミング）



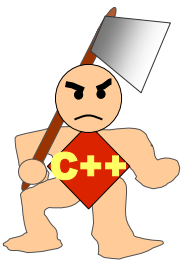






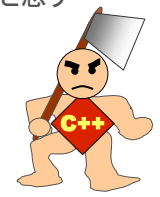
# 暴れん坊C++の姿

- C++言語の習得でぶつかったこと
  - 実体（スタック）とポインタと参照の使い分け
  - コピーと代入を自前で定義するクラス
  - constの記述がよくわからない
  - STLのコンテナ・イテレータは複雑
  - テンプレートは難関



# 暴れん坊C++の姿

- C++言語の学習
  - 15年間経っても熟練には程遠い発表者
  - これを読めばという定番の書籍が見当たらない
    - 「Effective C++」は最近読める知識レベルに達したと思う
      → 初心者に薦められる本ではない。
  - 独学には難しい言語ではないか
  - やさしいコミュニティがあればいいな









# 暴れん坊C++に蹂躙される開発現場

少人数・小規模な開発ではほとんど問題にならないが、
多人数・大規模開発で発生する困難な問題がある。

# 暴れん坊C++に蹂躙される現場

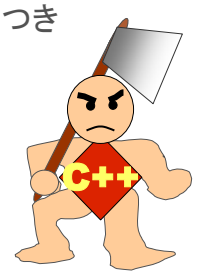
- 想定する開発現場
  - 開発人数： 10人以上
  - 開発規模： 十万行以上
  - 開発期間： 数ヶ月以上
  - プラットフォーム： Windows, UNIX系混在
  - 開発言語： C++言語




CODE GEAR FROM Borland
Where Developers Matter
Integrated Development Environments for Windows, Java, and Web Developers
DEVELOPER CAMP

# 暴れん坊C++に蹂躙される現場

- 多人数開発で起きる問題
  - プログラマの技量によるコード・品質のばらつき
  - コンパイル・リンク時間の増大
  - 名前の衝突
  - ライブラリ分割の問題
  - 同じクラスが異なるライブラリに重複
  - ソースコードのディレクトリ構造




本文書の一部または全部の転載を禁止します。本文書の著作権は、著作者に帰属します。

# 暴れん坊C++に蹂躙される現場

- プログラマの技量によるコード・品質のばらつき
  - プログラマのC++知識の理解・使用レベルは揃っていない

| | |
|---|---|
| C言語知識 | better Cとして使用するレベル<br>配列・インデックス、C標準関数を使う |
| 古典C++言語知識 | クラスを中心に使用するレベル<br>クラス・継承・集約、C標準+ iostreamを使う |
| 標準C++言語知識 | クラスに加えSTL・テンプレートを使用するレベル<br>イテレータ、アルゴリズムを使う |
| 前衛C++言語知識 | テンプレートを駆使するレベル<br>他の知識レベルでは理解不能なコード |




CODE GEAR FROM Borland
Where Developers Matter
Integrated Development Environments for Windows, Java, and Web Developers
DEVELOPER CAMP

# 暴れん坊C++に蹂躙される現場

- プログラマの技量によるコード・品質のばらつき

  例）Sensor型オブジェクトを管理するデータ構造と処理の書きっぷりの違い

「古典C++言語知識」

```
Sensor* sensors[MAX_NUM];
    :
for (int i=0; i<MAX_NUM; i++) {
    sensors[i]->update();
}
```

「標準C++言語知識」

```
vector<Sensor*> sensors;
    :
for (vector<Sensor*>::iterator it = sensors.begin(); it!=sensors.end(); ++it) {
    (*it)->update();
}
```

「前衛C++言語知識」

```
std::vector<Sensor*> sensors;
    :
std::for_each(sensors.begin(), sensors.end(), std::mem_fun(&Sensor::update));
```

# 暴れん坊C++に蹂躙される現場

- プログラマの技量によるコード・品質のばらつき

  例）数値型から文字列への変換処理の書きっぷりの違い

```
int value = 12345;
char buffer[16];
snprintf(buffer, 16, "%d", value);
```

「C言語知識」

```
int value = 12345;
ostringstream os;
os << value;
string str = os.str();
```

「古典C++言語知識」

```
int value = 12345;
string str = boost::lexical_cast<string>(value);
```

「標準C++言語知識」(+Boost)




CODE GEAR FROM Borland
Where Developers Matter
Integrated Development Environments for Windows, Java, and Web Developers
DEVELOPER CAMP

# 暴れん坊C++に蹂躙される現場

- コンパイル・リンク時間の増大
  - コード規模が大きくなると指数的にビルド時間が増加する

例） オープンソースのCORBA実装　ACE+TAO（Ver.1.6.1）の場合

| | ACEのみ | ACE+TAO 合計 |
|---|---|---|
| ビルド時間 | 0:10:40 | 2:35:30 |
| 総行数 | 307,819 | 1,066,708 |
| 命令行数 | 79,930 | 247,391 |
| ファイル数 | 1,248 | 2,592 |
| クラス数 | 841 | 4,548 |

ビルドマシンのスペック

- CPU: Athlon 64x2 4200+(2.2GHz)
- Mem: 2GB (DDR2 800)
- HDD: SATA 1.5Gbps
- OS: Windows Vista 64bit
- C++: VC++ 9.0 (VS 2008)

本文書の一部または全部の転載を禁止します。本文書の著作権は、著作者に帰属します。

# 暴れん坊C++に蹂躙される現場

- コンパイル・リンク時間の増大
  - 差分コンパイルは現実的ではない
    - C++はヘッダーファイルに実装細部を記述する（privateメンバー）
      - publicな部分に修正がなくても再コンパイルが必要
    - ヘッダーファイルが芋づる的に#includeされるため、広範囲に変更波及

```
project/
   +--- include
   |       +--- Alfa.h
   |       +--- Bravo.h
   |       +--- Charlie.h
   +--- alfa
   |       +--- Alfa.cpp
   +--- bravo
   |       +--- Bravo.cpp
   +--- charlie
           +--- Charlie.cpp
```

Alfa.h → Bravo.h → Charlie.h のinclude関係があれば、Charlie.hを修正すると、Charlie.cppだけでなく、Bravo.cpp、Alfa.cppの再コンパイルも必要となる



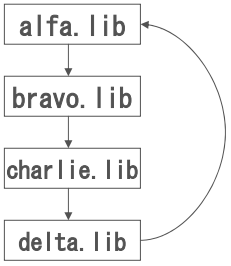

CODE GEAR FROM Borland
Where Developers Matter
Integrated Development Environments for Windows, Java, and Web Developers
DEVELOPER CAMP

# 暴れん坊C++に蹂躙される現場

- 名前の衝突
  - クラス名、マクロ名、ファイル名が衝突し、一見不可解なコンパイルエラーが発生
    - ヘッダーファイルのinclude順番で挙動が変わる
      - 例）<winsock2.h>と<windows.h>
    - ヘッダーファイルがなぜか読み込まれない
      - 同じファイル名が複数存在した
      - インクルードガードが重なっていた

# 暴れん坊C++に蹂躙される現場

- ライブラリ分割の問題
  - ライブラリ同士に相互依存や循環依存があると、ビルドが一筋縄ではいかない



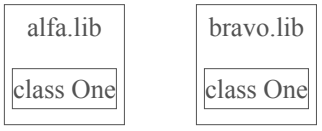

それぞれのライブラリ開発担当者には、相互依存の認識はほとんどない




CODE GEAR FROM Borland
Where Developers Matter
Integrated Development Environments for Windows, Java, and Web Developers
DEVELOPER CAMP

# 暴れん坊C++に蹂躙される現場

- 同じクラスが複数ライブラリに重複
  - 曖昧なパッケージ分割設計により、共通するクラスがあちこちのライブラリに含まれてしまう
    - クラスのstaticなデータがシステムで唯一とならない
    - #ifdefで異なるロジックがコーディングされていると、悲惨な状況に



# 暴れん坊C++に蹂躙される現場

- ソースコードのディレクトリ構造にも問題
  - ライブラリに関する問題の一因に、C/C++の伝統的なソースコードのディレクトリ構造がある

ディレクトリ分割なし

```
project/
  +--include
  |     +--Alfa.h
  |     +--Bravo.h
  |     +--Charlie.h
  +--src
        +--Alfa.cpp
        +--Bravo.cpp
        +--Charlie.cpp
```

中途半端なサブディレクトリ化

```
project/
  +--include
  |     :
  +--src
        +--alfa
        |     +--Alfa.cpp
        +--bravo
        |     +--Bravo.cpp
        +--charlie
              +--Charlie.cpp
```





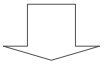

# 暴れん坊C++の押さえどころ

プログラマの技量によるばらつきをおさえる

パッケージ設計により設計段階からライブラリ化を考慮することで、ビルド問題、名前衝突問題、ライブラリ問題をおさえる

# 暴れん坊C++の押さえどころ

- プログラマの技量のばらつきをおさえる
  - プロジェクトで使用するC++知識範囲を定義する
    - C++言語特徴のどの範囲を使用するか
    - ライブラリセットは何を使用するか
    - 型、マクロ、エラー処理、メモリ管理の方法はどうするか



プロジェクト言語の規定




CODE GEAR FROM Borland
Where Developers Matter
Integrated Development Environments for Windows, Java, and Web Developers
DEVELOPER CAMP

# 暴れん坊C++の押さえどころ

- プログラマの技量のばらつきをおさえる
  - 「プロジェクト言語」をコーディング規約で規定する
    - 使用するC++言語範囲を定める
      - 例外機構の使用有無
      - 多重継承の使用有無
      - STL、テンプレートの使用有無　　等
    - 使用するライブラリセットを定める
      - OSのシステムコール使用有無
      - C標準関数の使用有無
      - C++標準クラスの使用有無
      - サードパーティライブラリの使用有無

本文書の一部または全部の転載を禁止します。本文書の著作権は、著作者に帰属します。

# 暴れん坊C++の押さえどころ

- プログラマの技量のばらつきをおさえる
  - プログラマは規約範囲を理解し実践できる技量が必要
    - 理想はプロジェクトに参加する時点で不足知識のトレーニング実施
      - 定型の教育コースでは難しい
      - トレーナーになれるリーダー格プログラマは忙しい
      - 定番書籍もない （Effective C++は難しい）
    - 現実解としてプロジェクト参加のための課題を用意
      - ミニ機能を設計・実装させる
      - 添削・模範解答の提示と指導
      - 開発ルール・開発環境はプロジェクト基準を使用




CODE GEAR FROM Borland
Where Developers Matter
Integrated Development Environments for Windows, Java, and Web Developers
DEVELOPER CAMP

# 暴れん坊C++の押さえどころ

- コーディング規約参考
  - 書籍（日本語）
    - 「C++ Coding Standards – 101のルール、ガイドライン、ベストプラクティス」（H.Sutter、A.Alexandrescu著）
    - 「Effective C++ 第3版－プログラムとデザインを改良するための55項目」（S.Meyers著）
    - 「C++スタイルブック」（T.Misfeldt、G.Bumgardner、A.Gray著）
  - Web（英語）
    - “Applied Informatics C++ Coding Style Guide – Rules and Recommendations” (フレームワークPOCOの規約）
    - “Joint Strike Fighter Air Vechicle C++ Coding Standards”(ロッキード・マーチン社）

# 暴れん坊C++の押さえどころ

- パッケージ設計の導入
  - C++はライブラリ化の機能はサポートしていない
    - OS／コンパイラ系に依存したライブラリの仕組みを使う
      - Windows： DLL
      - UNIX系： shared object (so)
  - 設計ルールでパッケージ設計を導入する
    - C++のnamespaceを使う
    - ヘッダーファイルの可視／非可視でパッケージアクセス制御
      - ディレクトリ構造上で可視／非可視を分ける






# 暴れん坊C++の押さえどころ

- パッケージ設計の導入
  - パッケージはそれぞれ異なるnamespaceに置く
  - namespaceごとにヘッダーファイル・ソースファイルを格納するディレクトリを設ける

パッケージ”alfa”のディレクトリとnamespace

```
project/
  +--alfa/
  |    +--include/
  |    +--src/
  +--bravo/
  |    +--include/
  |    +--src/
  +--charlie
  |    +--include/
  |    +--src/
```

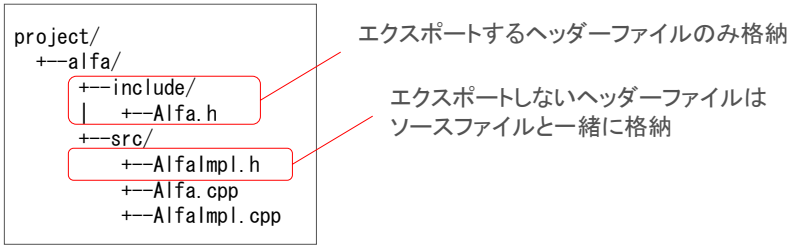
```
namespace alfa {

class Alfa {
};

}
```

# 暴れん坊C++の押さえどころ

- パッケージ設計の導入
  - パッケージ外部から可視なヘッダーファイルと非可視なヘッダーファイルとを分離し、格納するディレクトリを分ける



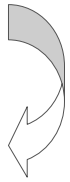




CODE GEAR FROM Borland
Where Developers Matter
Integrated Development Environments for Windows, Java, and Web Developers
DEVELOPER CAMP

# 暴れん坊C++の押さえどころ

- パッケージ設計の導入
  - 可視なヘッダーファイルから非可視なヘッダーファイルをincludeしない
    - → 絶縁設計を検討（書籍「大規模C++ソフトウェアデザイン」）
    - 継承よりコンポジション
    - データメンバーは実体よりポインタ
    - プロトコルクラス
    - 完全絶縁具体クラス（pimplイディオム）
    - コンポーネントラッパー



# 暴れん坊C++の押さえどころ

- パッケージ設計の導入
  - プロトコルクラスの実装例（Javaのinterfaceによく似ている設計）

```
#include <string>
#include <boost/shared_ptr.hpp>
using std::string;
using boost::shared_ptr;

namespace character {

class Character {
public:
   virtual ~Character();
   virtual const string getName() const = 0;
   virtual void setName(const string& aName) = 0;
   static shared_ptr<Character>
       create(const string& aName);
};
}
```

・private/protectedメンバーなし
・純粋仮想関数
・非インライン仮想デストラクタ
で定義する

実装はパッケージ外から非可視のクラスで提供する

インスタンス生成は、別クラス、外部関数、静的関数等で実現





# 暴れん坊C++の押さえどころ

- パッケージ設計の導入
  - パッケージ単位にライブラリをビルドする
  - ライブラリとエクスポートするヘッダーファイルだけを利用者へリリースする

```
project/
  +--alfa/
      +--lib/
      |   +--alfa.dll
      +--include/
      |   +--Alfa.h
      +--src/
          +--Alfaimpl.h
          +--Alfa.cpp
          +--Alfaimpl.cpp
```

alfaパッケージ利用者に提供する範囲
・ライブラリファイル
・可視なヘッダーファイル

利用者は、可視なヘッダーファイルが変更されなければ再コンパイル不要
（ライブラリの差し換えで対応可能）

# 暴れん坊C++の押さえどころ

- ビルドツール
  - 共通設定を一元化でき、一発操作でビルドできる、特定のIDEやプラットフォームに依存しないツールが理想
    - GNUmake : 記述が大変
    - Apache　Ant :　Javaでは定番だが、C++適用ノウハウが少ない
  - 着目しているツールは
    - ACE+TAOのビルドツール　MPC (MakeProjectCreator)
    - CMake




CODE GEAR FROM Borland
Where Developers Matter
Integrated Development Environments for Windows, Java, and Web Developers
DEVELOPER CAMP

## まとめ

多人数開発での問題の多くは、一見設計ではなく管理的な問題に見える。
しかし、管理的な問題を押さえるためには、設計から取り組む必要がある。

本文書の一部または全部の転載を禁止します。本文書の著作権は、著作者に帰属します。