



C++0xの概要

19-E-5

高橋 晶

株式会社ロングゲート
取締役

自己紹介

- C++標準化委員会 エキスパートメンバ
- はてなID : faith_and_brave
- Twitter ID : cpp_akira
- 著書 : C++テンプレートテクニック

アジェンダ

- C++0xとは
- スケジュール
- 言語の新機能
 - 初期化子リスト, メンバ初期化子, 範囲for文
 - ラムダ式, 型の別名
- 新ライブラリ
 - スマートポインタ, 正規表現, 乱数, スレット
- まとめ

C++0xとは

- 200x年(つまり2009年まで)に策定予定だったC++の次のバージョン。現在のバージョンはC++03。
- いまは2010年です！
- 残念ながら間に合いませんでした><。

スケジュール 1/2

- 当初の予定では2009年中
- 2008年7月に再スケジュールされ、
2011年を予定していた
- 2008年10月にCD (Committee Draft)が
公開された(仕様のβ版みたいなもの)

スケジュール 2/2

- “コンセプト”と呼ばれるC++0xの目玉になる予定だった機能が2009年7月に急遽却下された。この影響でさらに1年遅れると思われる。
- それに加え、まだ新機能の提案が行われていて、Issueの消化が間に合ってない状況なのでさらに遅れるかも
- 2010年3月にCD2が公開される予定

C++0xの言語機能

- 今回は、プログラミングをより容易にするシンタックスシュガーに焦点を当てて紹介します。

1. 初期化子リスト
2. メンバ初期化子
3. 範囲for文
4. ラムダ式
5. 型の別名

初期化子リスト 1/2

- C++03では、配列の初期化を以下のように書くことはできたが

```
int ar[] = {1, 2, 3};
```

- ユーザー定義型ではできなかった

```
// 1. 初期化のために要素を"追加"する
```

```
std::vector<int> v;  
v.push_back(1); v.push_back(2); v.push_back(3);
```

```
// 2. 配列で初期化して、範囲コンストラクタで初期化
```

```
int ar[] = {1, 2, 3};  
std::vector<int> v(ar, ar + 3);
```

初期化子リスト 2/2

- C++0xでは、ユーザ一定義型でも初期化子リストによる初期化を定義することができるようになる

```
std::vector<int> v = {1, 2, 3};
```

```
// 引数で初期化子リストを渡す
```

```
void foo(const std::vector<int>&);  
foo({1, 2, 3});
```

```
// 戻り値で初期化子リストを返す
```

```
std::vector<int> bar()  
{ return {1, 2, 3}; }
```

メンバ初期化子 1/2

- C++03では、メンバ変数を初期化するためにコンストラクタの初期化子リストを使用していた

```
class Window {  
    int x;  
    int y;  
    std::string title;  
  
public:  
    Window()  
        : x(0), y(0), title("") {}  
};
```

メンバ初期化子 2/2

- C++0xでは、メンバ変数の定義時に初期値を指定することができるようになる

```
class Window {  
    int x = 0;  
    int y = 0;  
    std::string title = "";  
};
```

範囲for文 1/3

- C++03のfor文は長かった…

```
std::vector<int> v = {1, 2, 3};
```

```
for (std::vector<int>::const_iterator it =
v.begin(), last = v.end(); it != last; ++it) {
    std::cout << *it << std::endl;
}
```

範囲for文 2/3

- アルゴリズムを使えばfor文を書かなくて済むけど、アルゴリズムに渡す用の関数オブジェクトを作るのがめんどくさい

```
struct disp {  
    void operator()(int x) const  
    { std::cout << x << std::endl; }  
};
```

```
std::vector<int> v = {1, 2, 3};  
std::for_each(v.begin(), v.end(), disp());
```

範囲for文 3/3

- C++0xならこう書ける！

```
std::vector<int> v = {1, 2, 3};
```

```
for (int x : v) {  
    std::cout << x << std::endl;  
}
```

ラムダ式 1/2

- for文が書きやすくなったのはいいけど、
これだとめんどくさいアルゴリズムは
誰も使わなくなっちゃうよ！

でも大丈夫！

ラムダ式 2/2

- C++0xでは、関数オブジェクトをその場で生成するラムダ式という機能が導入される！

```
std::vector<int> v = {1, 2, 3};
```

```
std::for_each(v.begin(), v.end(), [] (int x) {
    std::cout << x << std::endl;
});
```

型の別名 1/2

- C++03では、型の別名を付けるために `typedef`を使用していたが、わかりにくかった

```
// 1.unsigned intにuintという別名を付ける
typedef unsigned int uint;
```

```
// 2.int(*)(double)関数ポインタ型に別名を付ける
typedef int(*func_ptr)(double);
```

```
// 3.テンプレートを使用した型の別名
template <class ValueType>
struct num_map {
    typedef std::map<int, ValueType> type;
};
```

型の別名 2/2

- C++0xでは、型の別名を付けるために usingと=で書ける！

```
// 1.unsigned intにuintという別名を付ける
using uint = unsigned int;
```

```
// 2.int(*)(double)関数ポインタ型に別名を付ける
using func_ptr = int(*)(double);
```

```
// 3.テンプレートを使用した型の別名
template <class ValueType>
using num_map = std::map<int, ValueType>;
```

新ライブラリ

- C++0xでは、プログラミングをより安全に、より強力にするために標準ライブラリが大幅に強化されます。ここでは、その中でもとくに身近なものをいくつか紹介します。
1. スマートポインタ
 2. 正規表現
 3. 乱数
 4. スレッド

スマートポインタ

- newしたら自動でdeleteしてくれる、
ポインタのように振る舞うクラス

```
std::shared_ptr<Base> factory()
{
    return std::shared_ptr<Base>(new Derived());
}

void foo()
{
    std::vector<std::shared_ptr<Base>> v;
    v.push_back(factory());
    v[0] ->foo();
} // ← ここでコンテナ内のポインタが全てdeleteされる
```

正規表現

- その名の通り正規表現ライブラリ。
ECMAScript, basic, grep等の方言にも対応している

```
std::regex r("<[^>]+>");  
std::string str = "template <class T> hoge";  
std::string after = "<censored>";  
  
std::cout <<  
    std::regex_replace(str, r, after)  
<< std::endl;  
// "template <censored> hoge"
```

乱数

- 何種類もの疑似乱数生成器と分布クラスが提供され、それらを自由に組み合わせる

```
// メルセンヌツイスター法 + 整数一様分布
std::mt19937 gen;
std::uniform_int_distribution<> dst(0, 9);

for (int i = 0; i < 5; ++i) {
    int random_no = dst(gen); // 疑似乱数を生成
    std::cout << random_no << ",";
}
// 5, 0, 6, 1, 9,
```

スレッド

- スレットライブラリが汎用化され、
汎用的な並列プログラミングが可能になる

```
int parallel_sum(int* data, int size)
{
    int sum = 0;
    if (size < 1000)
        for (int i = 0; i < size; ++i)
            sum += data[i];
    else {
        future<int> handle =
            std::async(parallel_sum, data+size/2, size-size/2);
        sum += parallel_sum(data, size/2);
        sum += handle.get();
    }
    return sum;
}
```

まとめ

- C++0xでは、開発効率と安全性を向上させる
数多くの言語機能＆ライブラリが提供されます！
(本日紹介したのはほんの一部です)
- これらの言語拡張は、初学者がC++を学ぶのを
容易にするでしょう！
- すでに多くの主要コンパイラがC++0xへの
対応を始めています！

参考資料

- 2010年2月18日時点での最新仕様N3035
<http://www.open-std.org/jtc1/sc22/wg21/docs/papers/2010/n3035.pdf>

- 私のブログ記事
「C++0xの言語拡張まとめ」

http://d.hatena.ne.jp/faith_and_brave/20071022/1193052163

Enjoying C++0x !

(仮称)