『ストラウストラップのプログラミング入門』

で語られなかったいくつかのこと

Programming

高橋 晶(Akira Takahashi)
id:faith_and_brave
@cpp_akira

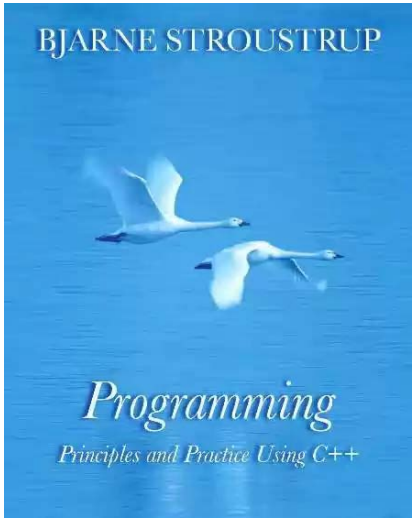
わんくま東京勉強会 2011/08/27(土)

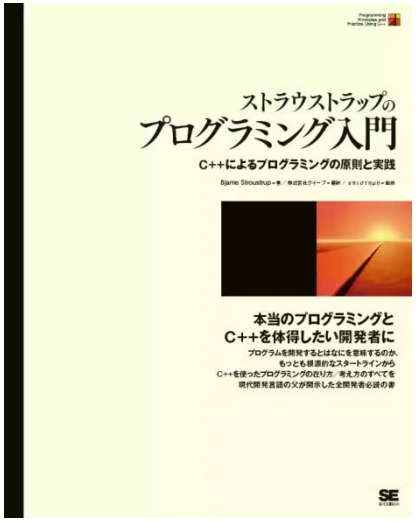
動機

C++の始祖Bjarne Stroustrupによる名著『Programming』。

この本では「プログラミングとは何か」というところから始まり、幅広い話題について広く掘り下げて解説されている。

この発表では、『Programming』で扱われなかったいくつかの重要なプログラミングの分野について解説していきます。





お題

- 並列処理
- ネットワーク
- コンピュータビジョン
- メタプログラミング

Chapter 1

並列処理

Parallel Processing

タダ飯の時間はとっくに終わってる

- 昔々、そんなに昔ではないけれど、
  遅いプログラムを組んでも、ムーアの法則によってコンピュータが速くなるのを待っていればいい時代がありました。

- ムーアの法則に限界が見えて、コンピュータが単純には速くならなくなった時代が到来したとき
  「The Free Lunch is Over(タダ飯の時間は終わりだ)」
  という宣言がなされ、並列プログラミングの時代がやってきました。

並行と並列

シングルスレッドで動かすプログラムは単純にはこれ以上速くならない。そこでとられた2つのアプローチ:

- スレッドやプロセスによる擬似的な複数同時処理
- 複数CPUによる完全な複数同時処理

前者を並行コンピューティング(Concurrent Computing)

後者を並列コンピューティング(Parallel Computing)

と呼び、両方合わせて並列コンピューティングと呼ぶ場合もある。
(並行は、実行環境によって並列になり得る。)

並行と並列 – 並行コンピューティング

スレッド/プロセスによる並行コンピューティングは、1つのCPUで複数の処理を同時に行う。
1つのCPUの性能を分割して複数の操作を行うため、それぞれの処理性能は低下する。

```
// 関数fと関数gを同時に動かす
thread t1(f);
thread t2(g);

...

// 終わるまで待つ
t1.join();
t2.join();
```

並行と並列 - 並列コンピューティング

並列コンピューティングは、複数CPUによって同時に複数処理を行う。これはCPUのコア数 (などのハードウェア並行性)分だけ真に同時に動かすことができる。

```
// CPUの個数分、コンテナvの各要素に対して
// 関数fを同時に適用していく
parallel_for_each(v, f);
```

並列プログラミングの難しさ

- 並列プログラミングでは、複数のタスクから同じ変数にアクセスする場合に問題が起こる。
  これは原因の追いにくいバグを生み出しやすい。
  (ミューテックスと呼ばれる仕組みによって、同時に1人しかアクセスできないようロックすることができるが、デッドロックや管理の複雑化などで問題が起き得る)
- 独立したタスクになるよう強く意識して設計する必要がある。

```
static int x = 0;

void f() { … x = 1; }
void g() { … x = 2; }

thread t1(f);
thread t2(g);

// xの値はどうなるだろう？
```

並列プログラミングの抽象化

並列プログラミングの難しさに対処するため、大きく2つの抽象化モデルが登場した。

- アクターモデル
メッセージパッシングによって並行プログラミングの安全地帯を提供する。

- STM(Software Transactional Memory)
トランザクションによって競合を検知する。

これらは、現代の多くの言語で標準、もしくはサードパーティライブラリによって提供され始めている。

関数型プログラミング

並列プログラミングが難しいのは、「状態が変更されるから」である。

あちこちでデータを書き換えたりせず、状態の変更を最小限にすれば、並列プログラミングの難しさはかなり緩和される。

近年、関数型プログラミング(Functional Programming)というパラダイムがようやく脚光を浴び始めた。

純粋な関数型プログラミングは、「関数は同じ引数を与えたら必ず同じ値を返す」というポリシーの元、データの不変性を基礎としたプログラミング手法である。

C#, C++0x, Scalaのような近年のプログラミング言語では、関数型プログラミングの特徴を多く取り入れ始め、状態の変更を最小限にできるような言語機能やライブラリを提供し始めている。

他の並列プログラミングのアプローチ

- Lock-free
  ミューテックスによる共有変数のロックは重いため、Compare and Swap(CAS)と呼ばれる軽量の競合チェック + 破壊的操作を行うLock-freeという手法が浸透してきている。
  (抽象の中に隠すのが一般的)

- GPGPU(General-purpose computing on GPU)
  近年のGPUは1,000を超える大量のコアを持っているため、同時処理に向いている。GPGPUは、GPUの能力をGPU本来の計算以外の用途に応用する技術である。

- 分散処理
  もはや一つのマシンだけでは処理しきれない巨大データや計算を扱うために、複数のサーバーに処理を分散させる手法である。

Chapter 2

ネットワークプログラミング

Network Programming

ネットワークプログラミング

- PCやモバイル端末がインターネットに接続できることが前提となる現代において、ネットワークプログラミングは欠かすことのできないものとなった。
- もはやスタンドアロンで目的を達成できるアプリケーションの分野はかなり限られる。

ネットワークプログラミングとは何か

- ネットワークプログラミングとは、ネットワークを介して相手とデータのやりとりを行うプログラミングのことである。

- サーバーに要求を行い結果を得るクライアント、
クライアントからの要求を受け結果を渡すサーバー、
というクライアントサーバーモデルが一般的である。

ネットワークプログラミングの領域は拡大している

ネットワークプログラミングが当たり前となってきた現代において、その用途もまた多岐にわたるようになった。

•Peer to Peer(P2P)
お互いがクライアントにもサーバーにもなり得る対等な関係の通信モデル。モバイル端末同士の通信で対戦ゲームなどで使用されている。

•巨大データの分散処理
ひとつのPCでは処理しきれないほど巨大なデータを複数のサーバーを使って処理する手法。ネットワークを介して他のサーバーに処理を依頼する。ログの統計や科学計算などで使用されている。

•ユーザー操作の統計
「ユーザーがどんな行動をとり、どれくらいの時間でプレイを中断し、二度と復帰しなくなったか」といった情報は、ソフトウェア開発者にとって重要な情報となり得る。ソーシャルゲームの分野では、ユーザーのあらゆる操作を集計し、次回作のために役立てている。

C++におけるネットワークプログラミング

- C++では、ネットワークプログラミングの方法として、Boost Asio Libraryを利用する方法がある。これは、非同期処理全般を扱うライブラリで、次期標準*の候補としても挙がっている。

```
asio::io_service io_service;
tcp::socket socket(io_service);

…
// 非同期にデータを送信する
std::string request =  “Hello” ;
async_write(socket, buffer(request), on_send);

…
// 送信終了時にこの関数が呼ばれる
void on_send(const error_code& error)
{
    if (!error)
        std::cout <<  “正常終了”  << std::endl;
}
```

* C++0xの後

通信で使用されるデータ形式

通信で使用される主なデータ形式は以下：

- XML
  HTMLライクな、タグで囲んで値を記述する形式
- JSON
  名前と値の組でデータを表現し、短い表記で型を表現できるデータ形式
- MessagePack
  言語間通信を意図した汎用バイナリ形式。データは現在のところ
  最も小さくなる。

ライブラリを使用して、ユーザー定義型をこれらのデータ形式にシリアライズして通信で送受信するのが一般的である。

Chapter 3

コンピュータビジョン

Computer Vision

コンピュータビジョンとは

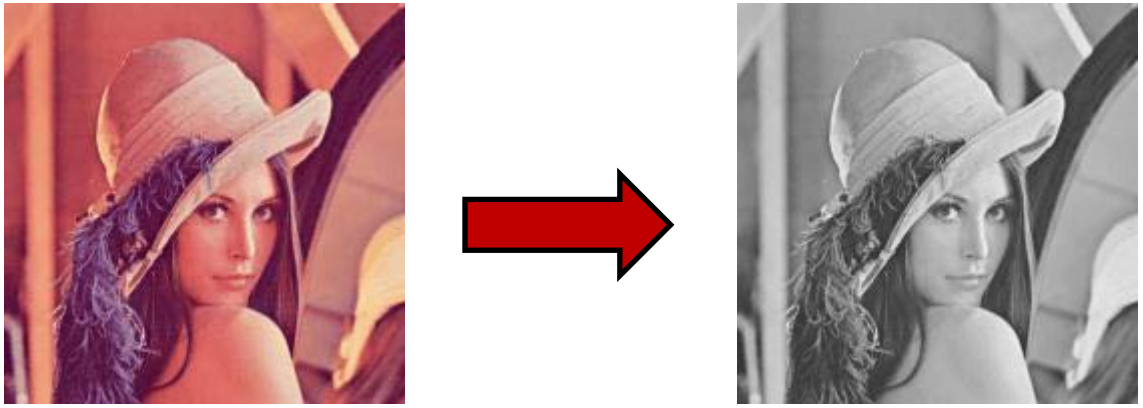
- 画像処理、画像認識といった、画像に対する操作、カメラから得られた情報を解析する操作などの分野がコンピュータビジョンと呼ばれている。

- この分野は古くから研究／利用されてきてはいるが、コンピュータの速度がネックになっていたため日の目を見なかった。処理速度が向上したことで複雑な処理でもようやく現実的に使えるようになってきた。

画像処理

- 画像に対する処理は、様々な分野において利用されている。簡単なところで言うと、画像の各ピクセルのRGB値に対して、ある値を掛けるとグレースケールになる：

```
Y（輝度）= r * 0.229 + g * 0.587 + b * 0.114;
r = Y;
g = Y;
b = Y;
```



画像認識

- 画像認識は、画像データや映像データから特定のパターンを認識して情報を抽出する技術である。
  物体検出や、人の顔認識、指紋認識やOCRなどが画像認識に当たる。

コンピュータビジョンはどこで使われるか

- 画像処理自体は、ゲームや、一般的なソフトウェアでも使われているが、近年は画像認識がより身近になってきた。
- 拡張現実(AR : Augmented Reality)と呼ばれる、現実の映像と仮想的な物体を合成してユーザーに提示する技術が注目されてきている。
- また、Kinectというデバイスによって、高機能なセンサによる画像認識が低価格で可能となったため、一般ユーザーが画像認識という技術に触れる機会が多くなってきた。

一般のソフトウェア開発者にもコンピュータビジョンを扱う技術・知識が求められるようになってきている。

OpenCV

- コンピュータビジョンでは、OpenCVというオープンソースのライブラリが広く使われており、C++にも対応している。

- このライブラリは、コンピュータビジョンの専門的な技術を抽象化し、クラス・関数として提供しているため、専門知識がそれほどなくても使用することができる。

- 最近では、OpenMPによる並列処理や、CUDAによるGPGPUへの対応なども行われている。

Chapter 4

メタプログラミング

Metaprogramming

メタプログラミング

メタプログラミングとは、プログラムに関するプログラムである。

- コードの自動生成を行うもの
  (e.g. Cプリプロセッサ、Template Haskell)
- 構文木を操作するもの
  (e.g. Lispマクロ、Template Haskell)
- 型の操作を行うもの
  (e.g. C++テンプレート、HaskellのfundepsとType families)
- EDSL(Embedded Domain Specific Language)の作成
  (e.g. 演算子オーバーロード、関数の中置記法)

などがある。

コード生成系メタプログラミング

- 言語の通常の構文では抽象化しきれないコードを自動生成するためにメタプログラミングが使われることがある。
  C++03における型安全な可変引数を生成するために使われたりする。

- MicrosoftのT4 Templateも、広義においてはコード生成系メタプログラミングの一種ととることができる。

- コピー&ペーストがどうしても必要になってしまう場面で、コード生成を自動化すれば1箇所直せば全て直る、という利点があり、メンテナンスが容易である。

構文木操作系メタプログラミング

- 構文木操作系メタプログラミングは、言語に対してユーザーが独自の構文を追加したり、任意のコンパイル時エラー、警告を発生させたりするのに使用することができる。

- 言語機能としては、Lispマクロ、GroovyのAST変換、D言語に搭載予定のマクロあたりが代表例である。

- また、近年ではコンパイラプラグインという形で、コンパイラが持っている構文木等のメタ情報を抽出し、操作できる機能を提供しているところも出てきた(例: GCC, Clang, Scala)。
これは、IDEの作成に使えるのはもちろんのこと、絶対コーディング標準なども作り出すことができる。

型操作系メタプログラミング

- 型操作系メタプログラミングは、型や、型の要件を集めたコンセプトごとに最適な実装を選択させる、最適化やアダプトの機構として使われることが多い。

- ユーザー定義型はコンパイラの最適化が届かない領域であるため、こういったことが必要になる。
  「抽象化にはコストがともなう」と言われる問題へのアプローチ。

- コンパイル時に戦略を切り替えるPolicy-based Designは、抽象化された実装をひとつだけ用意するのではなく、あらゆる要求に応えるために用途別の最適な実装を切り替えるのに使用される。

EDSL(Embedded Domain Specific Language)の作成

- データベース操作や構文解析といった特定のドメインのための構文がほしくなることがある。
  そういった場合に、演算子などに新たな意味を持たせて言語内言語を作ることで有用となり得る。

- 例:
  - EBNFライクな構文解析をC++上で行うBoost.Spirit
  - C++03でラムダ式を表現するBoost.Lambda
  - Erlangライクな構文でアクターモデルを表現するScalaのActor
  - Scalaでデータベース操作のためのRogue

まとめ

- プログラミングの分野は広い。
- 自分がどの分野に興味があるのか確かめるためにも、いろんな情報を常に仕入れ、飛び込んでいこう。