

# セキュリティ&プログラミング キャンプ2008

ソースコードの読み方

2008年8月15日

ミラクル・リナックス株式会社

よしおかひろたか

# 本日のアジェンダ

- ソースコードの読み方

# トラブルシューティングと ソースコードの読み方

- トラブルシューティングと問題の理解
- 実践的なコードの理解

# ソースコードを読むチカラ

- プログラマの基礎体力
  - ソフトウェア開発コストの大部分は保守
  - 不具合修正、改良、機能追加にはコードの理解が必須
- 技術者の付加価値
  - OSSは深追いできる
  - 陳腐化しにくい
- プロフェッショナルとしての研鑽
  - すぐれた技術者はソースコードを上手に読む

# コードの読み方

- なぜ、コードを読むのか
- どのように、読むのか

# なぜコードを読むのか？

- 仕事だから
  - トラブルシューティング(不具合修正)
  - 機能修正、機能開発
  - 自己研鑽、勉強
- 趣味だから
  - 楽しいから
  - 自己啓発(知的好奇心)
- 不純な動機
  - 形から入る

# なぜコードを読むのか

- 人それぞれ、人生いろいろ
- 無目的でもいいじゃないか
- 熟読、濫読、積読、黙読、音読、再読、誤読、精読、速読、耽読、通読、復読、輪読、朗読、輪読、…

# コードの理解について

- モットー：  
コードは読むな、  
理解しろ～

# どのようにコードを理解するのか

- 個人的な方法を紹介する
  - 唯一あるいはベストな方法というわけでもない
  - 適材適所、もっと良い方法があると思う
  - 公開することによって進化したい(もっと良い方法への模索)

# ソースコードを読む視点

	微視的理解	巨視的理解
静的理解		
動的理解		

# 理解の仕方、読み方

- 静的、動的理解
- 微視的、巨視的
- 規模の把握
- ツールの利用
- 事例

# 静的理解、動的理解

- 静的理解
  - 字面での理解
- 動的理解
  - 動作による理解

# 静的、動的構造

- 静的構造
  - 規模
  - ディレクトリ構造
  - 名前つけ規約
- 動的構造
  - 呼び出し経路
  - プロファイリング
  - 実行結果

# 微視的、巨視的

- 微視的: 細部からの理解
  - 最終的にはコードの一行
- 巨視的: 全体からの理解
  - 規模、構造、機能など。実行結果(性能?)
  - 俯瞰図、鳥瞰図。

# 規模の把握（巨視的理解）

- 規模重要

- 規模（相手）を知らずして作戦を立てられない
- 大局的な地図、俯瞰図、鳥瞰図
- 大規模になればなるほど、システマティックな方法論が必要になってくる
- 巨視的な理解

# 規模の把握

- 小規模: 100K行未満程度、  
ファイル数100未満  
10人未満
- 中規模: 100K行～1M行程度  
ファイル数100～1000未満  
100人未満
- 大規模: 1M行以上  
ファイル数1000以上  
100人以上  
ざっくりの規模感

# 規模の把握(例)

- `find -type f -name "*.ch"|wc`
- `find -type f -name "*.ch"|xargs wc|grep total`

Ruby	1.8.5	257	197767
Linux	2.6.18	16522	680万
MySQL	5.0.24ε	1795	988463

# ディレクトリ構造

- トップディレクトリは、ソフトウェアの論理的構造を表している
  - doc ドキュメント
  - lib ライブラリ関係
  - test テスト
- ソースツリーの把握

# ドキュメント

- README, INSTALL, COPYING, ...
- 内部ドキュメント(Docs)
- リリースノート
- ChangeLog

# 変更の履歴

- ChangeLog/Release Notes
- コード管理システム
  - 例: Linux git/Subversion/CVS
- Mailing List
- Wiki
- blog
- 変更(時間軸)の微視的理解

# ドキュメント、情報収集

- Mailing List
- Bug database
- 開発者との会話
- Googleに聞く

# ソースコードを読む視点

	微視的理解	巨視的理解
静的理解	ソースコード、ChangeLog、リリースノート	ディレクトリ構造、名前付け規約、規模の把握(行数、ファイル数など)
動的理解	デバッガによる実行	実行性能、リグレッションテスト

# いよいよコードを読む？

- ツール
  - エディタ:(x)emacs
  - デバッガ:gdb
  - クロスレファレンス:cscope
  - カーネルの場合、クラッシュダンプ(crash)

# デモ、実習

- 例題としてGNU coreutilsを読むことにする。
  - <http://ftp.gnu.org/gnu/coreutils/>
  - <http://savannah.gnu.org/projects/coreutils/>
  - <http://git.savannah.gnu.org/gitweb/?p=coreutils.git>
- GNU Hello
  - <http://savannah.gnu.org/projects/hello>

# 実習

- ソースコードの規模を理解する
- ディレクトリ構造を理解する
- ビルドする
- ソースコードのナビゲート
- 機能変更をする

# 実習

- ディレクトリ構造を理解する
- ファイル名を眺める
- ファイル数を調べる
- プログラムの行数を調べる

# 実習

- ソースコードの入手

\$ git clone git://git.savannah.gnu.org/coreutils.git

- ビルドの準備

README-hackingをよく読む

- 必要なツールを揃える

- Automake <<http://www.gnu.org/software/automake/>>
- Autoconf <<http://www.gnu.org/software/autoconf/>>
- Bison <<http://www.gnu.org/software/bison/>>
- Gettext <<http://www.gnu.org/software/gettext/>>
- Git <<http://git.or.cz/>>
- Gperf <<http://www.gnu.org/software/gperf/>>
- Gzip <<http://www.gnu.org/software/gzip/>>
- Perl <<http://www.cpan.org/>>
- Rsync <<http://samba.anu.edu.au/rsync/>>
- Tar <<http://www.gnu.org/software/tar/>>

# 実習

- ビルド

```
$ time git clone git://git.savannah.gnu.org/coreutils.git
```

```
$ cd coreutils
```

```
$ time sudo ./bootstrap
```

```
$ time sudo ./configure
```

```
$ time sudo make
```

```
$ time sudo make check
```

# 動的理解

- ビルド

# 動的理解

- ともかく動かす
  - ~~make; make install~~
  - strace
  - ltrace
  - gdb
  - oprofile
  - リグレッションテスト
  - ベンチマークテスト

# make

- とりあえず、make; make install
- 実行環境の構築
  - gcc -g (デバッグシンボルを付加する)
  - cscopeのインデックスを作成
  - ビルドは(x)emacsのshellなどから行い、ビルドのログを取得しておく

# strace

- システムコールのトレース

```
$ strace ruby -v
execve("/usr/local/bin/ruby", ["ruby", "-v"], [/* 39 vars */]) = 0
uname({sys="Linux", node="asianux2.miraclelinux.com", ...}) = 0
brk(0) = 0x976a000
access("/etc/ld.so.preload", R_OK) = -1 ENOENT (No such file or directory)
open("/etc/ld.so.cache", O_RDONLY) = 3
fstat64(3, {st_mode=S_IFREG|0644, st_size=89946, ...}) = 0
old_mmap(NULL, 89946, PROT_READ, MAP_PRIVATE, 3, 0) = 0xb7fe...
close(3) = 0
open("/lib/libdl.so.2", O_RDONLY) = 3
```

# **gdb**

- デバッガはコードを理解するためにある  
∴コードを読むために使う
- <http://savannah.gnu.org/projects/gdb>

# gdbで読むための準備

- gcc -g でコンパイル
- コンパイルしたコードサイズが大きくなる以外、特に副作用はない
- printfデバッグは有害無益

# gdb

- ブレークポイントを設定
- ウォッチポイント(変数の変更)
- run
- 止まった時点で
  - bt スタックフレームの表示
  - p 変数の表示
  - c 実行再開、s ステップ実行(関数に潜る)、n ステップ実行(関数に潜らない)
- 繰り返す

# oprofile

- プロファイラー
  - 実行時のボトルネックを発見
  - # opcontrol ---start
  - テストの実行
  - # opcontrol ---stop
  - # oprofile -l
- <http://oprofile.sourceforge.net/news/>

# oprofile

CPU: Core Solo / Duo, speed 2666.77 MHz (estimated)

Counted DCACHE\_PEND\_MISS events (Weighted cycles of L1 miss outstanding) with a unit mask of 0x00 (Weighted cycles) count 100000

vma	samples	%	linenr	info	app name	symbol name
000000000042be50	244787	33.2383	gc.c:1661		ruby	os_each_obj
000000000042bfac	1	4.1e-04	gc.c:1599			
000000000042bfb9	5	0.0020	gc.c:1599			
000000000042bfbe	6	0.0025	gc.c:1599			
000000000042bfd0	4862	1.9862	gc.c:1601			
000000000042bfd3	228573	93.3763	gc.c:1601			
000000000042bfd6	2698	1.1022	gc.c:1601			
000000000042bfd8	250	0.1021	ruby.h:672			

# oprofile

- 高速道路で、ズバリ最もコストのかかっているところに連れて行ってくれる
- コードを読まないで、理解する極意

# 微視的理解

- ひたすらコードを読む
- 王道はない
- ↑身もふたもない
- データ構造、変数などに注目し、どこで定義され、参照され、代入(変更)されているかという観点でみる
- デバッガとエディタ、クロスリファレンスツールを駆使

# 微視的理解

- `$ time find -type f |egrep ¥ '¥.([chp](xx|pp)*|cc|hh)$'¥ |xargs egrep -l hogehoge`
- hogehogeを含むファイルを検索

# 微視的理解

- クロスリファレンスツール
  - 変数の定義(型情報)、変更(代入)、参照
- 変数(関数)が、どのように定義されていて、どのように参照、変更されているかを追う
- cscope  
<http://cscope.sourceforge.net/>
- lxr  
<http://lxr.linux.no/>
- GNU GLOBAL  
<http://www.gnu.org/software/global/>

# 実習

- cscopeを利用する  
\$ time cscope-indexer -r
- emacsを利用する
- gdbを利用する
- oprofileを利用する

# ソースコードを読む視点

	微視的理解	巨視的理解
静的理解	ソースコード、ChangeLog、リリースノート、cscope	ディレクトリ構造、名前付け規約、規模の把握(行数、ファイル数など)
動的理解	デバッガ(gdb)による実行、strace	oprofile、strace、実行性能、リグレッションテスト

# 参考書

- Linux
  - 詳解LINUXカーネル第三版  
ISBN:487311313X
  - Linuxカーネル2.6解読室  
ISBN:4797338261
- コードリーディング
  - ISBN:4839912653
- Rubyソースコード完全解説
  - ISBN:4844317210(品切れ中)  
<http://i.loveruby.net/ja/rhg/>

# ユメのチカラ(ブログ)

- <http://blog.miraclelinux.com/yume/>
- ブックマークで見た人気エントリー
- ソースコードの読み方(524個)
  - [http://blog.miraclelinux.com/yume/2007/08/post\\_d6bd.html](http://blog.miraclelinux.com/yume/2007/08/post_d6bd.html)
- デバッグ方法論(99個)
  - [http://blog.miraclelinux.com/yume/2007/08/post\\_d3eb.html](http://blog.miraclelinux.com/yume/2007/08/post_d3eb.html)
- 多くの人に興味がある話題

- ブログ:ユメのチカラ
- <http://blog.miraclelinux.com/yume/>
- 未来のいつか/hyoshiokの日記
- <http://d.hatena.ne.jp/hyoshiok/>