LLDB

Chapter 2

1 概要

2 LLDB とは
LLDB は LLVM のデバッガです。

3 コマンド
LLDB を起動すると先頭に(lldb)の表示が出て対話モードになります。

```
(lldb) <noun> <verb> [<options>]
```

の形式でコマンドを実行し、様々の処理を行うことでデバッグを行います。

コマンド入力時は TAB による補完が可能です。また、曖昧さがない場合にはコマンドは先頭数文字だけの入力でも認識されます。

4 ヘルプ
コマンドのヘルプを help で確認できます。

```
(lldb) help [-ahu] [<command>...]
```

文法:

option name	description
-a, --hide-aliases	エイリアスを非表示
-h, --show-hidden-commands	-から始まる隠しコマンドを表示
-u, --hide-user-commands	ユーザー定義コマンドを非表示
command	ヘルプを表示するコマンドを指定

help -au でコマンドの一覧を閲覧できます。

5 基本操作

5.1 LLDB の起動
LLDB を起動するには以下のコマンドをシェルで叩きます。

```
$ lldb [<options>] [<program>]
```

5.2 プログラムの読み込み
デバッグするプログラムを GDB に読み込むには起動時の引数で指定するか、コマンドで指定することができます。

起動時には例えば

```
$ lldb ./a.out
```

のように指定し、コマンドでは

```
(lldb) target create ./a.out
```

のように指定できます。

5.3 プログラムの起動

プログラムを起動するには `process launch` を使用します。

```
(lldb) process launch [<options>]
```

文法:

option name	description
`-s, --stop-at-entry`	エントリポイントで停止
`-n, --no-stdio`	
`-A, --disable-aslr <bool>`	アドレスランダム化を行うか
`-e, --stderr <file>`	標準エラー出力を指定
`-i, --stdin <file>`	標準入力を指定
`-o, --stdout <file>`	標準出力を指定

全てのオプションは `help process launch` で確認できます。

5.4 プログラムの終了

プログラムを停止せずにトレース終了するには `ctrl+c` を使用するか `process kill` コマンドを使用します。

5.5 LLDB の終了

LLDB を終了するには `ctrl+d` を使用するか `quit` コマンドで終了できます。

6 プログラムの停止

デバッグ中のプログラムを停止させるにはあらかじめ停止場所や条件を設定しておく必要があります。停止箇所としてはブレークポイント、ウォッチポイントが設定できます。

6.1 ブレークポイント

プログラムが到達した際に停止する場所です。

6.1.1 ブレークポイントの設置

ブレークポイントを設置するには `breakpoint set` コマンドを使用します。

```
(lldb) breakpoint set [<options>]
```

文法:

option name	description
`-A, --all-files`	全てのファイルを検索
`-D, --dummy-breakpoints`	ダミーのブレークポイントを設置

-H, --hardware	ハードウェアブレークポイントを使用
-d, --disable	無効化されたブレークポイントを設置
-l, --line <linenum>	行番号 linenum を指定
-a, --address <addr>	アドレス addr を指定
-n, --name <func>	関数名 func を指定
-F, --fullname <name>	関数の完全修飾名を指定
-S, --selector <selector>	Objective-C のセレクタ名を指定
-M, --method <method>	C++のメソッド名を指定
-r, --func-regex <reg>	正規表現 reg にマッチする関数名を持つ関数を指定
-b, --basename <func>	関数の基本名が func の関数を指定(C++の名前空間や引数)を無視
-p, --source-pattern-regex <reg>	指定したファイル内のソースコードで正規表現にマッチする箇所を指定
-E, --language-exceprion <lang>	指定した言語の例外スローを指定
-y, --joint-specifier <linespec>	filename:line[:column]の形式でファイルと行を指定
-k, --structured-data-key <none>	スクリプトによるブレークポイントの実装に渡されるキーと値のペアのキー。 ペアは複数指定できます。
-v, --structured-data-value <none>	スクリプトによるブレークポイントの実装に渡されるキーと値のペアの値。 ペアは複数指定できます。
-G --auto-continue <bool>	コマンド実行後自動で再開
-C, --command <cmd>	停止時に自動実行するコマンド
-c, --condition <cond>	条件式 cond を満たすときだけ停止
-i, --ignore-count <n>	ブレークポイントを無視する回数
-o, --one-shot <bool>	一度停止したら削除
-q, --queue-name <name>	指定したキューに入っているスレッドのみ停止
-t, --thread-id <tid>	指定したスレッドのみ停止
-x, --thread-index <tidx>	指定したインデクスのスレッドのみ停止
-T, --thread-name <name>	指定したスレッドのみ停止
-R, --address-slide <addr>	指定されたオフセットを、ブレークポイントが解決するアドレスに追加します。現在のところ、これは指定されたオフセットをそのまま適用し、命令境界に整列させようとはしません。
-N, --breakpoint-name <name>	ブレークポイントの名前
-u, --column <col>	列を指定
-f, --file <filename>	検索するファイルを指定
-m, --move-to-nearest-code <bool>	一番近いコードへブレークポイントを移動
-s, --shlib <name>	共有ライブラリを指定
-K, --skip-prologue <bool>	プロローグをスキップ

6.1.2 ブレークポイントの削除

ブレークポイントを削除するには `breakpoint delete` コマンドを使用します。

```
(lldb) breakpoint delete [<breakpoint-id>]
```

`breakpoint-id` の値は `breakpoint list` で確認できます。 ブレークポイントを指定せずに実行すればすべてのブレークポイントを対象にできます。

6.1.3 ブレークポイントの無効化

ブレークポイントを無効化するには `breakpoint disable` コマンドを使用します。

```
(lldb) breakpoint disable [<breakpoint-id-list>]
```

ブレークポイントを指定せずに実行すればすべてのブレークポイントを対象にできます。

6.1.4 ブレークポイントの有効化

ブレークポイントを有効化するには `breakpoint enable` コマンドを使用します。

```
(lldb) breakpoint enable [<breakpoint-id-list>]
```

ブレークポイントを指定せずに実行すればすべてのブレークポイントを対象にできます。

6.1.5 ブレークポイント到達時実行コマンド

ブレークポイントに到達した際に実行するコマンドを `breakpoint command` コマンドで管理できます。

```
(lldb) breakpoint command <subcommand> [<options>]
```

`subcommand` には `add`, `delete`, `list` を指定できます。

6.1.5.1 add

コマンドを追加します。

```
(lldb) breakpoint command add [<options>] <breakpoint-id>
```

文法:

option name	description
`-o, --one-liner <command>`	コマンドを指定

6.1.5.2 delete

コマンドを削除します。

```
(lldb) breakpoint command delete <breakpoint-id>
```

6.1.5.3 list

設定されたコマンドの一覧を表示します。

```
(lldb) breakpoint command list <breakpoint-id>
```

6.1.6 ブレークポイントの保存

ブレークポイントをファイルに保存するには `breakpoint write` コマンドを使用します。

```
(lldb) breakpoint write [<options>] [<breakpoint-id-list>]
```

文法:

option name	description
`-a, --append`	保存先ファイルが既存の場合、書き加える
`-f, --file <file>`	保存先のファイルを指定

ブレークポイントを指定しない場合、すべてのブレークポイントを対象とします。

6.1.7 ブレークポイントの読み込み

`write` で保存したブレークポイントを読み込むには `breakpoint read` コマンドを使用します。

```
(lldb) break read [<options>]
```

文法:

option name	description
`-N, --breakpoint-name <name>`	名前を指定してその名前付きのブレークポイントのみ読み込む
`-f, --file <file>`	読み込み元のファイルを指定

ブレークポイントを指定しない場合、すべてのブレークポイントを対象とします。

6.2 ウォッチポイント

式の値を監視して変更があった際にプログラムを停止させます。

6.2.1 ウォッチポイントの設置

ウォッチポイントの設置には `watchpoint set` コマンドを使用します。

```
(lldb) watchpoint set <subcommand> [<options>] -- <varname>|<expr>
```

`subcommand` には `variable` と `expression` が指定でき、それぞれ変数、式の監視を行います。
文法:

option name	description
`-s, --byte-size <size>`	監視する値のメモリサイズを指定
`-w, --watch-type <type>`	ウォッチタイプを指定。`read|write|read_write` が指定可能

6.2.2 ウォッチポイントの削除
ウォッチポイントを削除するには `watchpoint delete` コマンドを使用します。

```
(lldb) watchpoint delete [<watchpoint-id-list>]
```

`watchpoint-id` の値は `watchpoint list` で確認できます。 ウォッチポイントを指定せずに実行すればすべてのウォッチポイントを対象にできます。

6.2.3 ウォッチポイントの無効化
ウォッチポイントを無効化するには `watchpoint disable` コマンドを使用します。

```
(lldb) watchpoint disable [<watchpoint-id-list>]
```

ウォッチポイントを指定せずに実行すればすべてのウォッチポイントを対象にできます。

6.2.4 ウォッチポイントの有効化
ウォッチポイントを有効化するには `watchpoint disable` コマンドを使用します。

```
(lldb) watchpoint enable [<watchpoint-id-list>]
```

ウォッチポイントを指定せずに実行すればすべてのウォッチポイントを対象にできます。

6.2.5 ウォッチポイント到達時実行コマンド
ウォッチポイントに到達した際に実行するコマンドを `watchpoint command` コマンドで管理できます。

```
(lldb) watchpoint command <subcommand> [<options>]
```

`subcommand` には `add`, `delete`, `list` を指定できます。

6.2.5.1 add
コマンドを追加します。

```
(lldb) watchpoint command add [<options>] <watchpoint-id>
```

文法:

option name	description
`-o, --one-liner <command>`	コマンドを指定

6.2.5.2 delete
コマンドを削除します。

```
(lldb) watchpoint command delete <watchpoint-id>
```

6.2.5.3 list
設定されたコマンドの一覧を表示します。

```
(lldb) watchpoint command list <watchpoint-id>
```

7 プログラムの再開

ブレークポイント等で停止したプログラムを様々な方法で再開できます。再開方法は主にステップ実行と継続実行に分かれ、ステップ実行では一定分だけ進んで停止し、継続実行では次のブレークポイントまで進みます。

7.1.1 継続実行

継続実行をするには `thread continue` コマンドを使用します。

```
(lldb) thread continue [<thread-index> ...]
```

`thread-index` を指定するとそのスレッドのみ継続実行できます。指定しない場合全てのスレッドが対象となります。`thread-index` の値は `thread list` コマンドで確認できます。

7.1.2 関数から帰る

関数から帰るには `thread step-out` コマンドを使用します。 現在選択中のフレームが帰るまで実行します。フレームの選択方法は Section 8.2 を参照してください。

```
(lldb) thread step-out [<thread-index>]
```

7.1.3 ソース行単位ステップオーバー

ソース行単位ステップオーバーをするには `thread step-over` コマンドを使用します。

```
(lldb) thread step-over [<thread-index>]
```

7.1.4 命令単位ステップオーバー

命令単位ステップオーバーをするには `thread step-inst-over` コマンドを使用します。

```
(lldb) thread step-inst-over [<thread-index>]
```

7.1.5 ソース行単位ステップイン

ソース行単位ステップインをするには `thread step-in` コマンドを使用します。

```
(lldb) thread step-in [<thread-index>]
```

ソース行単位ステップインをするには `thread step-in` コマンドを使用します。

```
(lldb) thread step-in [<thread-index>]
```

7.1.6 命令単位ステップイン

命令単位ステップインをするには `thread step-inst` コマンドを使用します。

```
(lldb) thread step-inst [<thread-index>]
```

7.1.7 until

特定の箇所まで実行したい場合には `thread until` コマンドを使用します。

```
(lldb) thread until [<options>] <linenum>
```

文法:

option name	description
`-a, --address <addr>`	停止箇所のアドレスを指定
`-f, --frame <frame-index>`	`until` の実行フレーム番号を指定

8 スタックフレーム

8.1 バックトレース

コールスタックを表示するには `thread backtrace` コマンドを使用します。

```
(lldb) thread backtrace
```

8.2 フレームの選択

フレームを選択するには `frame select` コマンドを使用します。

```
(lldb) frame select [<options>] [<frame-index>]
```

文法:

option name	description
`-r, --relative <offset>`	選択中のフレームからのオフセットで選択

8.3 フレーム情報の表示

フレーム情報を表示するには `frame info` コマンドを使用します。

```
(lldb) frame info
```

9 変数の表示

変数を表示するには `frame variable`

```
(lldb) frame variable [<options>]
```

文法:

option name	description
`-A, --show-all-children`	上限を無視して子を表示
`-D, --depth <count>`	表示する最大深さ
`-F, --flat`	フラットフォーマットで表示

-G, --gdb-format	GDB フォーマットで表示
-L, --locationn	位置情報を表示
-O, --object-description	言語ごとの説明 API を使用して表示
-P, --ptr-depth <count>	値をダンプする際のポインタの深さ
-R, --raw-output	フォーマットを行わない
-S, --synthetic-type <bool>	synthetic provider に従うかを表示
-T, --show-types	型を表示
-V, --validate <bool>	型チェックの結果を表示
-Y [<count>], --no-summary-depth=[<count>]	概要情報を省略する深さ
-Z, --element-count <count>	型が配列あるかのように表示
-a, --no-args	関数の引数を省略
-c, --show-declaration	変数の宣言情報を表示
-d, --dynamic-type <none>	オブジェクトの完全な動的型を表示
-f, --format <fmt>	フォーマット
-g, --show-globals	グローバル変数を表示
-l, --no-locals	局所変数を省略
-r, --regex	変数名を正規表現として解釈
-s, --scope	変数のスコープを表示
-t, --no-recognized-args	recognized function arguments を省略
-y, --summary <name>	変数の出力が使用すべきサマリーを指定
-z, --summary-string <name>	フォーマットに使用するサマリー文

9.1 ディスプレイの代用

LLDB には `display` の直接的なコマンドはありません(エイリアスとしては存在します)。 代わりに `target stop-hook` コマンドを使用して、停止時に実行するコマンドを設定できます。 `target stop-hook` で `frame variable` を追加することで `display` と同様の動作を行えます。

```
(lldb) target stop-hook add --one-liner "frame variable argc argv"
```