



ejs project

eJSCの起動方法

eJSCを利用する前に、環境設定を行う必要がある。
ejsディレクトリの下のjsディレクトリで、npmによる初期化を行う。

```
$ cd ejsc/js  
$ npm install
```

eJSCは次のように起動する。(カレントディレクトリはビルドしたディレクトリとする。)

```
$ node ../ejsc/js/ejsc.js [options] source1.js ...
```

これにより、source1.js, source2.js, ... をコンパイルし、コンパイル結果を単一のSBCファイル(拡張子は.sbc)あるいはOBCファイル(拡張子は.obc)に出力する。デフォルトはSBCファイルである。生成されたSBC/OBCファイルは、source1.js, source2.js, ... の内容を順に実行するようなものである。もしも-logオプションがあるソースファイルの直前に指定されていれば、それ以降のソースファイルはログ付き命令を用いるようにコンパイルされる。

現在は陽にejsc.jar(あるいはnewejsc.jar)を指定することはできない。
カレントディレクトリやPATH(CLASSPATH?)の通ったディレクトリにあるejsc.jar等が利用される。

eJSCを以下の通り直接起動することもできる。

```
$ java -jar compiler.jar --spec specfile [options] source1.js source2.js ...
```

この場合、ejsc.jarに内蔵のパーサが利用される。しかし残念ながら内蔵パーサにはバグが多く、ある程度の規模のjsプログラムではしばしばパースに失敗/間違える。

node ejsc/js/ejsc.js を利用した場合、Babelによりjsプログラムがパースされjson形式で出力され、これがejsc.jarへ渡される。

オプションの説明

optionsには、以下が指定できる。

出力

- --out-obc : OBCファイルを出力する。
- -o filename : オプションの説明 出力先のファイル名をfilename.sbcあるいはfilename.obcにする。デフォルトではsource1が使われる。
- --out-ei : EIファイル(Embedded Instruction,埋め込み命令)を出力する。Mbedなど組み込み機器のFLASH領域に命令列を配置する目的で利用する。

最適化

- -O : おすすめの最適化オプションをつけてコンパイルする。これを与えることが推奨されている。(--bc-opt const:cce:copy:rie:dce:reg:rie:dce:reg -opt-g3と同じである。)
- --bc-opt optimizers : バイトコード上の最適化を行う。optimizersには、一連の最適化器を指定する。指定できる最適化器は次の通りである。
 - const : 定数伝播と合成命令の利用 (Constant propagation and using superinstructions)
 - cce : 重複する定数ロード命令除去 (Common constant loading elimination)
 - copy : コピー伝播 (Copy propagation)
 - rie : 冗長命令除去 (Redundant instruction elimination)
 - dce : 到達しない命令の除去 (Dead code elimination)
 - reg : レジスタの再利用 (Register (re-)assignment)

- `-opt-g3` : 可能であれば、局所変数と実引数にレジスタを割り当てる。

その他

- `--spec specfile`: 仕様ファイル (通常はejsのmake時に作成される`ejsvm.spec`)を指定する。デフォルトは合成命令のない通常の命令セットである。
- `-fn n`: 関数番号の開始をnにする。(現在は使われていないので、このオプションは将来削除されるかもしれない。)
- `-log source.js` : `source.js` (とそれ以降に指定されたファイル) をログ付き命令を利用するコードにコンパイルする。

デバッグ用

- `--estree` : ESTreeを出力する。
- `--iast` : iASTを出力する。
- `--analyzer` : result of some AST-based analysis (???) を出力する。
- `--show-llcode` : low-level internal code を出力する。
- `--show-opt` : 最適化の詳細情報を出力する。

開発者向け情報






- デフォルトのmakeでは、`compiler.jar`ではなく`newejsc.jar`が作られる。
- 以下の ant properties が利用できる。
 - `specfile`: 仕様ファイルのパス。(デフォルトは `src/ejsc/default.spec`).
 - `vmgen`: `vmgen.jar`のパス (デフォルトは `../ejsvm/vmgen/vmgen.jar`).




eJSVMの起動方法

eJSVMは、次のようにして起動する。

```
% ./ejsvm options file1 file2 ...
```

optionsには、以下が指定できる。

- `-l`: 最後の式の評価結果を出力する。
- `-f`: 実行前に関数表を出力する。(DEBUGビルドでのみ有効)
- `-t`: 実行トレース (実行される仮想機械命令) を出力する。(DEBUGビルドでのみ有効)
- `-a`: `-l`, `-f`, `-t` をすべて指定したと同じ。
- `-u`: 総実行時間, GC時間, GC回数を出力する。
- `-R`: Replモードで起動する。通常、このオプションをユーザが陽に使うことはない。
- `--hc-prof`: 隠れクラスの情報を出力する。デバッグ用のオプションである。(ビルド時に`HC_PROF`マクロが有効な場合のみ)
- `--shape-prof`:  added by ugawa (ビルド時に`SHAPE_PROF`マクロが有効な場合のみ)
- `--profile`: プロファイル情報を出力する。出力先は、次の`--poutput`オプションが指定されていない限り、標準出力である。(ビルド時に`PROFILE`マクロが有効な場合のみ)
- `--poutput <poutput name>`: プロファイル出力先をfilenameにする。(ビルド時に`PROFILE`マクロが有効な場合のみ)
- `--coverage`: ログつき命令の網羅率情報を出力する。(ビルド時に`PROFILE`マクロが有効な場合のみ)
- `--icount`: 実行されたログつき命令のカウント情報を出力する。(ビルド時に`PROFILE`マクロが有効な場合のみ)
- `--forcelog`: すべての命令を強制的にログつき命令にする。(ビルド時に`PROFILE`マクロが有効な場合のみ)
- `--pcntuniq`:  added by hirasawa (ビルド時に`PROFILE`マクロが有効な場合のみ)
- `--bpoutput <bpoutput name>`:  added by hirasawa (ビルド時に`PROFILE`マクロが有効な場合のみ)
- `--icccprof <icccprof name>`:  added by hirasawa (ビルド時に`ICC_PROF`マクロが有効な場合のみ)
- `--gc-prof`: データ型ごとのアロケーション量、GCあたりの回収量、GCあたりの生存量と言った、GCに関する詳細な統計情報を出力する。(ビルド時に`GC_PROF`マクロが有効な場合のみ)
- `--ic-prof`:  added by ugawa (ビルド時に`IC_PROF`マクロが有効な場合のみ)

- `--as-prof` :  added by ugawa (ビルド時にAS_PROFマクロが有効な場合のみ)
- `-m` : VMヒープサイズをバイト数で指定する.
- `--threshold` : GC開始スレッシュホールドをバイト数で指定する.
- `-s` : VMスタックサイズをワード数(JSValue数)で指定する.
- `--dump-hcg <dump hcg file name>` :  added by ugawa (ビルド時にDUMP_HCGマクロが有効な場合のみ)
- `--load-hcg <load hcg file name>` :  added by ugawa (ビルド時にLOAD_HCGマクロが有効な場合のみ)
- `--buildinfo` : ビルド時情報を出力 (ビルド時にPRINT_BUILD_INFOマクロが有効な場合のみ)

file1, file2, ... の部分には, eJSC によってコンパイルされたsbcファイルあるいはobcファイルを指定する. これらのファイルは, 並べられた順に実行される.

eJSiの起動方法

eJSiはeJSVMの対話的実行 (REPL) 環境である. ejsiを実行するには, 次のようにする.

```
% ./ejsi options
```

optionsには, 以下が指定できる.

- `-h` : オプション一覧とその説明を表示する. `--help`としても可.
- `-v ejsvm` : ejsvmのパスを指定する. `--ejsvm ejsvm`としても可. デフォルトは./ejsvm.
- `-r runtime` : ejscのバックエンド(パーサ部分)をnodeかjavaから選択する. `--runtime runtime`としても可. デフォルトはnode.
- `-a arguments` : ejscのバックエンド(パーサ部分)に渡す引数を指定する. `--runtime-arg arguments`としても可. デフォルトは, nodeに対しては../ejsc/js/ejsc.js, javaに対しては-jar.
- `-c ejsc_jar` : ejscのjarファイルのパスを指定する. `--ejsc ejsc_jar`としても可. デフォルトは./ejsc.jar.
- `-s ejsvm_spec` : ejscを起動する際に与えるspecファイルのパスを指定する. `--spec ejsvm_spec`としても可. デフォルトは./ejsvm.spec.
- `-t tmpdir` : 一時的に作成するjsファイル, および, OBC ファイルを置くディレクトリを指定する. `--tmpdir tmpdir`としても可. デフォルトは/tmp.
- `-p prompt` : プロンプトの文字列を指定する. `--prompt prompt`としても可. デフォルトはeJSi>.
- `--verbose` : 付加情報も出力する.
- `--legacy` : ejscを古いパーサを利用して(nodeを利用せずに)実行する. `--runtime node --runtime-arg ../ejsc/js/ejsc.js`と指定するのと同じ.

現状の ejsi は, ユーザとの間のインタラクションを行うフロントエンド部と, 実際にプログラムを実行するVM部からなる. 上によって起動されるのはフロントエンド部である. フロントエンド部はパイプを作成した後, 子プロセスをforkする. 子プロセスは ejsvm を `-R` オプションをつけて exec する,

フロントエンド部は, 以下のことを繰り返し行う,

- ユーザからの入力されたJavaScriptの式を読み, js ファイルに出力する.
- ejsc を起動し, jsファイルをコンパイルして OBC ファイルを生成する. 現状では SBC ファイルは使わない.
- OBCファイルの内容をパイプに書き込み, VM部に伝える.
- パイプを通して実行結果をVM部から受け取り, 出力する.

現状の問題点

MacOS上で実行した場合, ejsiをCtrl-Zでいったんsuspendしてからfgで再開すると, フロントエンドとejsvmとの間のパイプが腐って終了してしまう. Linuxなど他のOSでどうなるかは, 未確認.

```
% ./ejsi
eJSi> 1+2
it = number:3.000000e+00
eJSi> ^Z
Suspended
% fg
```

```
./ejsi  
%
```

ejsi はいずれ ejsvm をライブラリ化して libejsvm を作った後は libejsvm を利用するように再実装する予定なので、この問題点については対処しない。

ejs-build-and-run_run.txt · 最終更新: 2023/05/02 06:19 by iwasaki