キミならどう書く(Ruby版)

株式会社ネットワーク応用通信研究所
前田 修吾

主要なクラス

- ls-lR.rbの主要クラスは以下の通り。
  - Node
  - Parser
  - Command
  - Shell

Node

- ls -IRを解析して作成されるツリー構造のノードを表現する(Compositeパターン)。
- Nodeは抽象クラスで、実際にはインスタンス化されない。
- サブクラスとして、FileNode, DirectoryNodeという具象クラスが定義されている。

Parser

- 以下の手順でls -IRの出力を解析する。
  1)最初の行を読み、ディレクトリ名を取り出す。
  2)ルートの場合は、DirectoryNodeを作成。そうでない場合は、@directoriesから取り出す。
  3)ディレクトリ内の各ファイル・ディレクトリに対応するNodeを作成し、2のディレクトリの子供として追加。
  4)ディレクトリの場合、@directoriesに保存。
  5)1に戻って入力がなくなるまで繰り返す。

Command

- シェルで実行される各コマンドを表現する(Commandパターン)。
- Commandは抽象クラスで、実際にはインスタンス化されない。
- すべてのサブクラスはexecメソッドを再定義しなければならない。
- execは配列でコマンドに対する引数を受け取り、それに応じた処理を行う。

Shell

- メインのクラス。
- runメソッドで以下の処理を行う。
  1)Parserによって入力を解析する。
  2)標準入力から一行ずつコマンドを受け取り、
  Shellwordsで引数を分解する。
  3)コマンドに応じたCommandオブジェクトの
  execメソッドを呼び出す。

各コマンド毎の処理

- quit(QuitCommand)
  - 終了する
- pwd(PwdCommand)
  - カレントディレクトリのパスを表示する
- cd(CdCommand)
  - カレントディレクトリを変更する
- ls(LsCommand)
  - ファイル･ディレクトリの一覧を表示する
- dfs/bfs(SearchCommand)
  - 深さ優先探索/幅優先探索を行う

quit(QuitCommand)

- exitするだけ。

pwd(PwdCommand)

- カレントディレクトリを表すNodeオブジェクトのpathメソッドを呼び出してパスを取得し、表示する。
- pathメソッドは、再帰的に親ノードのpathメソッドを呼び出し、絶対パスを生成する。

cd(CdCommand)

- 引数で指定されたパスに対応するNodeオブジェクトを取得し、カレントディレクトリに設定する。
- パスに対応するNodeオブジェクトの取得には、Node#get_descendantメソッドを使用する。
- get_descendantは再帰的に子ノードのget_descendantメソッドを呼び出し、パスに対応するNodeオブジェクトを取得する。

ls

- カレントディレクトリの子ノードすべてを順に出力する。
- 各ノードの文字列化にはNode#to_sを用いる(明示的な呼び出しはないが、printメソッドにより間接的に呼び出される)。
- NodeはLs -lRの出力を保持しているので、to_sは単にそれを返すだけ。

dfs/bfs(1)

- dfsは深さ優先探索を、bfsは幅優先探索を行う。
- 探索はNodeオブジェクトのacceptメソッドを呼ぶことにより行う。
- acceptメソッドは各サブクラスで再定義されており、FileNodeではvisit_fileメソッドを、DirectoryNodeではvisit_directoryメソッドを呼び出す(Visitorパターン)。

dfs/bfs(2)

- dfs/bfsはともにSearchCommandクラスが処理するが、探索の順序のみが異なる。
- 探索の順序は、DfsScheduler/BfsSchedulerによって制御する。両者を切替えることによって、SearchCommandをdfs/bfsの両方に対応させる(Strategyパターン)。
- DfsSchedulerは、スタックを使用し、最後に追加されたノードを最初に処理する(深さ優先)。
- BfsSchedulerは、キューを使用し、最初に追加されたノードを最初に処理する(幅優先)。

dfs/bfs(3)

- dfs/bfsは、find(1)に類似したオプションで、検索条件を指定することができる。
- オプションの解析は、FindExpressionParserによって行い、Expressionのサブクラスによって構文木を生成する(Compositeパターン)。
- Expressionのサブクラスはevaluateメソッドを持ち、構文木に対して再帰的に呼び出すことにより、式全体の評価を行う(Interpreterパターン)。
- オプションが指定されなかった場合は、NullExpressionを使用(NullObjectパターン)。

Rubyとデザインパターン

- プログラムの動作には、Javaのinterfaceや abstract methodのようなものはいらない。
- 言語要素により実現可能なものが多い。
  - Iterator → ブロック付きメソッド
  - Command → Procオブジェクト(クロージャ)
  - Singleton → 特異メソッド