Dartを軽く触ってみた

Dart Meetup Tokyo #4

(2018/03/29)

水島宏太

自己紹介

- Twitter ID: @kmizu
- GitHub: kmizu
- Love: Scala/Rust/Nemerle/...
- 構文解析アルゴリズムマニア
- Dart歴：1週間くらい（合計時間）

今日のお話

- Dartを触ってみた雑感
- プログラミング言語マニア的視点で

きっかけ

- 2018年2月にGoogleがFlutterのβを公開した
- DartがFlutterの主要な言語であることを知る
- Dart触ってなかったし、入門してみるか

1.x or 2.x

- 2.x系はまだ開発版らしい
  - 型チェック周りが色々改善されたと聞いた

→ 2.0の開発版をインストール

私のプログラミング言語の始め方

1. 公式ページのドキュメントをざっと眺める
2. 処理系をインストール
3. Hello, Worldを書く
4. パーザコンビネータを書く ←重要

パーザコンビネータとは

- パーザを組み立てるための内部DSL（EDSL）
- パーザを関数（かオブジェクト）とみなす
- 小さいパーザを合成して、大きなパーザを作る

パーザコンビネータの例（Scala）

```
object Calculator extends SCombinator[Int] {
  def E: Parser[Int] = rule(A)
  def A: Parser[Int] = rule(chainl(M) {
    $("+").map { op => (lhs: Int, rhs: Int) => lhs + rhs } |
    $("-").map { op => (lhs: Int, rhs: Int) => lhs - rhs }
  })
  def M: Parser[Int] = rule(chainl(P) {
    $("*").map { op => (lhs: Int, rhs: Int) => lhs * rhs } |
    $("/").map { op => (lhs: Int, rhs: Int) => lhs / rhs }
  })
  def P: Parser[Int] = rule {
    (for { _ <- string("("); e <- E; _ <- string(")")} yield e) |
    number
  }
  def number: P[Int] = rule(set('0'to'9').+.map{_.mkString.toInt}
}
```

何故パーザコンビネータか

言語の概要を把握するのに便利

- 種々の機能を利用する
  - クラス/オブジェクト
  - 関数/無名関数
  - ジェネリクスなどの型関係の機能
- 文字列処理
- 遅延評価
- 演算子多重定義

完成図（利用側）

- スライドに収めるために若干改変

```
Parser<int> expression() => rule(() => additive());
Parser<int> additive() => rule( ... );
Parser<int> multitive() => rule(
  () {
    var Q = s('*').map((op) => (int lhs, int rhs) => lhs * rhs);
    var R = s('/').map((op) => (int lhs, int rhs) => lhs ~/ rhs);
    return primary().chain(Q | R);
  }
);
Parser<int> primary() => rule(
  () => number()
| (s('(').then(expression()).then(s(')'))).map((t) =>
    t.item1.item2
  )
);
```

完成図（利用側）

- スライドに収めるために若干改変

```
void main() {
  var e = expression();
  print(e('1+2*(3/4)')); // 1
  print(e('(1+2)*3/4')); // 2
  print(e('10+20*30')); // 610
}
```

パーザコンビネータ（1）

- 部品の準備

```
abstract class ParseResult<T> {
  String next;
  ParseResult(this.next) {}
  T get value;
  bool get successful;
}
class Pair<T1, T2> {
  T1 item1;
  T2 item2;
  Pair(this.item1, this.item2);
  @override String toString() {
    return '(${item1}, ${item2})';
  }
}
```

パーザコンビネータ（1）

- 部品の準備

```
class ParseSuccess<T> extends ParseResult<T> {
  T value;
  ParseSuccess(this.value, String next) : super(next);
  @override bool get successful => true;
  @override String toString() =>
    'ParseSuccess(${value}, ${next})';
}
class ParseFailure<T> extends ParseResult<T> {
  T value = null;
  ParseFailure(String next) : super(next);
  @override bool get successful => false;
  @override String toString() =>
    'ParseFailure(${next})';
}
```

パーザコンビネータ（1）

- 部品の準備

```
typedef ParseResult<T> ParserFunction<T>(String input);
typedef U BiFunction<T1, T2, U>(T1 t1, T2 t2);
Parser<T> rule<T>(Parser<T> body()) => parserOf((input) =>
  body()(input)
);
Parser<T> parserOf<T>(ParserFunction<T> fun) =>
  new Parser<T>(fun);
```

感想（1）

- 文法はかなりJava風味
- 関数の型の宣言がイケてない
  - C言語の関数ポインタの文法の悪い部分まんま
- 関数の本体が式のときに => で楽できるの良い
  - C#にもあったような気がする
- コンストラクタ定義の糖衣構文が微妙

パーザコンビネータ（2）

- プリミティブな関数の定義

```
Parser<String> s(String literal) => parserOf((input) =>
  input.startsWith(literal) ?
    new ParseSuccess<String>(literal,
      input.substring(literal.length)
    ) :
    new ParseFailure<String>(input)
);
```

パーザコンビネータ（3）

- 連接コンビネータ

```
Parser<Pair<T, U>> then<U>(Parser<U> that) => parserOf(
  (input) {
    var r1 = this.fun(input);
    if(r1.successful) {
      var r2 = that.fun(r1.next);
      return r2.successful ?  new ParseSuccess ... :
                              new ParseFailure ... ;
    } else {
      return new ParseFailure<Pair<T, U>>(input);
    }
  }
);
```

パーザコンビネータ（4）

- 選択コンビネータ

```
Parser<T> operator |(Parser<T> that) => parserOf(
  (input) {
    var r1 = this.fun(input);
    return r1.successful ? r1 : that.fun(input);
  }
);
```

パーザコンビネータ（5）

- 結果を加工するコンビネータ

```
Parser<U> map<U>(U f(T result)) => parserOf(
  (input) {
    var r = this.fun(input);
    return r.successful ?
      new ParseSuccess<U>(f(r.value), r.next) :
      new ParseFailure<U>(r.next);
  }
);
```

感想（2）

- varで楽できるのはGood
- だいたいJavaの知識＋αで書けるのもGood
- ジェネリックな関数の文法は良い
- 演算子がジェネリックになれないのが不便
  - then なんて名前を付ける羽目に
- 関数の型に引数名書かないといけないの面倒
  - dartanalyzerが怒る

パーザコンビネータ（6）

- 繰り返しコンビネータ

```
Parser<List<T>> many() => parserOf(
  (input) {
    var rest = input;
    List<T> values = [];
    while(true) {
      var r = this.fun(rest);
      if(!r.successful) return new ParseSuccess(values, rest);
      values.add(r.value);
      rest = r.next;
    }
  }
);
```

パーザコンビネータ（7）

```
Parser<List<T>> many1() => this.then(this.many()).map((t) {
  List<T> result = [];
  result.add(t.item1);
  result.addAll(t.item2);
  return result; });
```

パーザコンビネータ（8）

- これでほぼ全部

```
Parser<T> chain(Parser<BiFunction<T, T, T> > q) =>
  this.then(q.then(this).many()).map((t) {
    var init = t.item1; var list = t.item2;
    return list.fold(init, (a, fb) {
      var f = fb.item1;
      var b = fb.item2;
      return f(a, b);
    });
  });
```

感想（3）

- 型推論周りでちょっとハマった
  - var result = []; としちゃった
- dartanalyzerに割と怒られた
  - 先にdartコマンドで実行結果確認してたせい
- 言語組み込みタプルが欲しい
  - パターンマッチも

とある型推論のIssue

- はじまり
- 嘆きのツイート
- Dartの中の人の返事

これが未だに解決されてないのが解せぬ...

感想（全体）

- 尖った機能・先進的な機能は全くない
  - 「面白く」はない
- かなりJava/C#フレンドリー
  - (Java + C#) / 2 な感じの文法
- シンタックスシュガーは割と色々ある
  - 比較的書きやすい
- Java/C#プログラマに勧めやすい言語