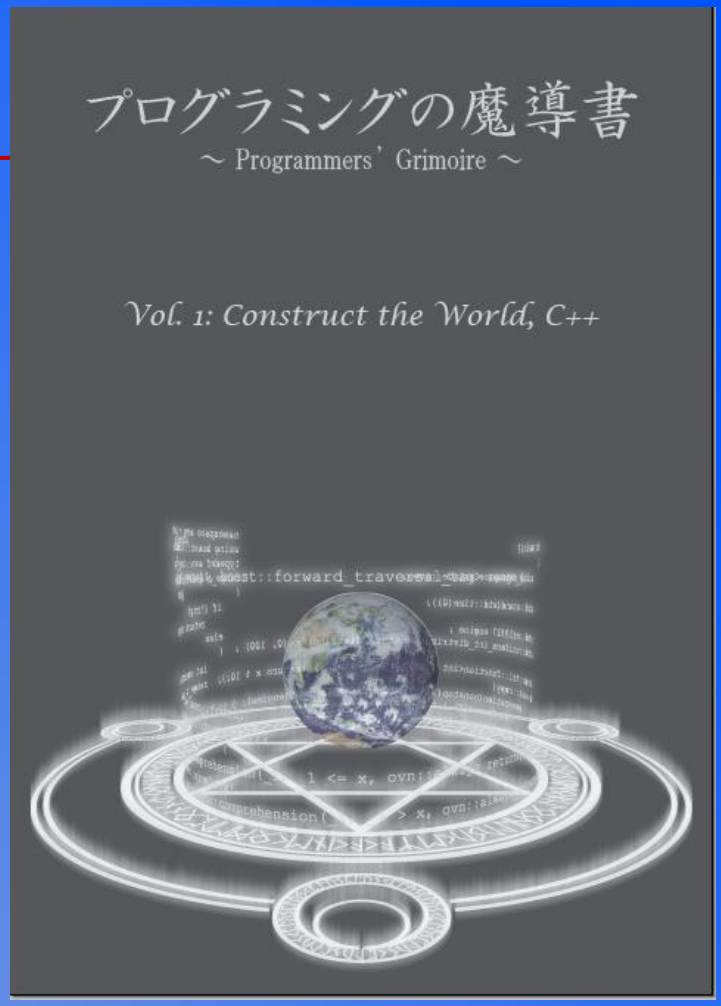
Scalaがもたらす言語の進化

高橋晶 (Akira Takahashi)

はてな：id:faith_and_brave
Twitter：@cpp_akira

自己紹介 1/2

- C++標準化委員会のひと
- 言語好き
  (言語を学び、
  おもしろい機能をC++で実現する)
- 魔導書作ってます
  (そろそろvol.2はじめます)



株式会社ロングゲート – 製品案内
http://longgate.co.jp/products.html

自己紹介 2/2

- ないと生きていけないライブラリ
  - Boost.Fusion – スーパーウルトラハイパータプル
  - Boost.Spirit.Qi – Fusionを基礎とする構文解析器
  - Boost.MultiIndex – 複数のインデックスを持てるコレクション

  Scalaにもほしいですね！

- P-Stade C++ LibrariesのOven Range Libraryのメンテもしてます。
  http://p-stade.sourceforge.net/
  http://sourceforge.net/projects/p-stade/

やりたかった内容

- 言語の設計思想
- どの言語からどの機能を取り入れてきたのか
- 言語拡張の際の互換性のポリシー

やりたかった内容

- 言語の設計思想
- どの言語からどの機能を取り入れてきたのか
- 言語拡張の際の互換性のポリシー

→ Oderskyさんがこのへんについてほとんど何も書いてない。
「JavaやC#と対話しやすいようにした」くらい。

推測で書けるけどあまりおもしろくないのでやめました。

今日やる内容

- Scalaの設計と特徴的な機能
- それによって今後の言語にどのような影響を与えるか

Scalaとはどんな言語か

- オブジェクト指向と関数型のハイブリット言語
  → マルチパラダイム言語
- Java VM上に実装されていて、Javaのライブラリをそのまま使える
- .NET版は(おそらく人手不足で)開発が停滞中

Scalaは
言語の進化を促進させるための言語

- 私にとって、Scala自体は理想的な言語ではない

- Scalaは、理想の言語を生み出すため、
  オブジェクト指向から関数型へのパラダイムシフトを促し、
  マルチパラダイム + DSLという今後のプログラミング言語の
  在り方、方向性を広めるための言語だと考えている

マルチパラダイムの重要性 1/3

- Javaは、1.5でGenericsが入る前までは
  純粋なオブジェクト指向言語だった。
  これは、プログラミング初心者に(もしくはどんなバカにでも)
  オブジェクト指向プログラミングを強制させるのに役立った。

- Haskellも純粋な関数型言語という意味ではJavaと同じ。
  その言語らしいプログラミングを強制させることができた。

- しかし、これらが十分に浸透した今、プログラミングスタイル
  を縛るだけではなく、より強力な表現力が求められる。

マルチパラダイムの重要性 2/3

オブジェクト指向と関数型、
他のパラダイムは互いに相容れないものではない。

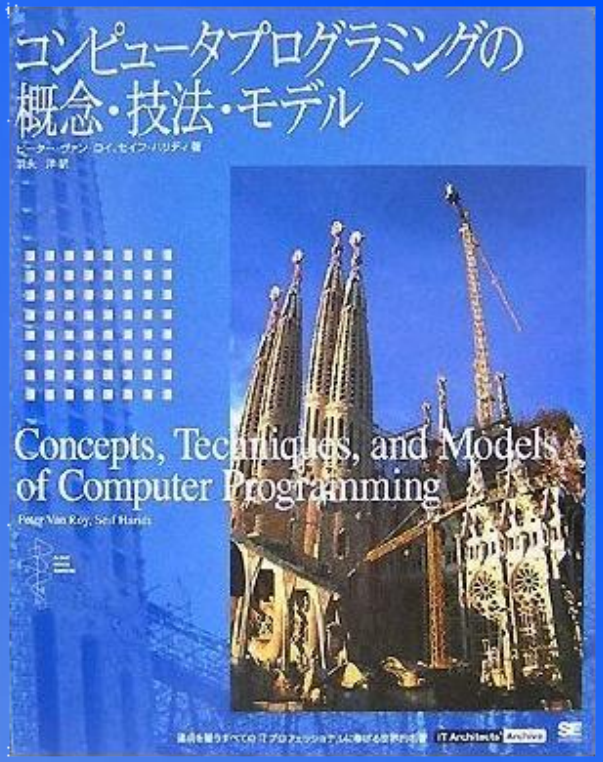
互いを組み合わせることにより、互いが苦手とする
分野を補完し合い、さらにより良く強力な
プログラミングが可能となる。

マルチパラダイムは、複数の考えの組み合わせによって
プログラミングに無限の可能性を与えうるものである。
これはC++が言語拡張なしにライブラリだけで今日でも
成長し続けていることが証明していると言える。

*「アイデアというものは、無数の小さなアイデアの
組み合わせによってのみ生み出される」*

マルチパラダイムの重要性 3/3

マルチパラダイムは、プログラミングを体系的に学ぶのにも役立つ。
教育の場において、オブジェクト指向プログラミングを教えるのにはJavaを使い、関数型プログラミングを教えるのにはHaskellを使う、というような方法では学生を混乱させるだけで、本質的なことに集中して教育することができない。



「科学的基盤を導入しようと色々努力しても、プログラミングはまず間違いなく、小手先仕事として教えられることになる。普通、1つ(か、ほんの少し)のプログラミング言語(例えば、Javaとそれを補うHaskell、Scheme、またはProlog)の枠の中で教えられる。…
ツールと概念がごちゃごちゃになっている。それに輪をかけて、オブジェクト指向、論理型、関数型等の「パラダイム(paradigm)」と呼ばれる、プログラミングの異なる見方に基づく、考え方の異なる流派がはびこっている。流派それぞれが独自の科学を持つ。1つの学問としてのプログラミングの統一性は失われた。…」

言語の進化としてのScalaの功績 1/5

1. オブジェクト指向と関数型をうまく融合させた

・immutable

不変の式を表現できるようになったことで、ユーザーが副作用について意識するようになった。
また、Concurrent/Parallelなプログラミングが必須となる時代にうまく適合した。
言語(Odersky)がvarよりもval, mutableコレクションよりimmutableコレクションを推奨していることも重要。
immutableは、Concurrentプログラムの最適化の可能性、プログラムをTestableにしやすい等々利点がいっぱい。

・パターンマッチ

手続き的言語ベースの構文の中でパターンマッチをうまく表現した。
case classのアイデアは、さらに改良されて他の言語にも今後取り入れられるだろう。
パターンマッチがなかったころは、形式チェックと値取り出し、変数定義と分解を分けて行わなければならなかったため、間にバグが入り込みやすかった。

言語の進化としてのScalaの功績 2/5

2. Concurrentプログラムの抽象化

・アクターモデルの採用
生のスレッドを使う時代はFree Lunch(*)とともに終わった。
共有メモリにアクセスするためにlock/unlockを書くような危険な行為は、ユーザーにさせてはいけない。

Scalaはアクターモデルを採用することで、Concurrentプログラムにおける
「**安全地帯**」を生み出した。

* 昔々、プログラムもコンピュータも遅かったころ、プログラムの改良をせずに
ムーアの法則でコンピュータが速くなるのを待ってるだけでよかった時代があった。
シングルコアの改良に限界がきてマルチコアの時代が訪れたとき
「Free Lunch Is Over(タダ飯の時間は終わりだ)」という宣言がなされ、
並列プログラミングの時代がはじまった。

言語の進化としてのScalaの功績 3/5

3. Genericsの進化

・Structual Subtypingの採用

Scalaでは、「Tという型はfooというメソッドを持っていれば継承関係になくていい」というのを表現できるようになったため、インタフェースの定義/実装にとらわれずによくなった。

```
// ※継承関係にない
class Rectangle { def draw = println( “Rectangle” )  }
class Circle    { def draw = println( “Circle” ) }

// drawというメソッドを持っていればいい
type Drawable = { def draw: Unit }

// 同じリストに入れる
val ls = List[Drawable](new Rectangle, new Circle)
ls foreach(_.draw)
```

言語の進化としてのScalaの功績 4/5

4. 例外を必要としないエラーハンドリング

・さよならtry/catch
try/catchはプログラムを冗長にした。
Scalaでは、
　　・パターンマッチ
　　・Option(無効/エラー値と正常値を表現)
　　・Either(エラー情報付きOption。なぜかあまり使われていない)
を標準で採用することで、しばしば冗長になるtry/catchを事実上不要にした。

ここで重要なのは、例外が必須(デフォルト)ではなくなったことにより、例外が真に必要なケースを議論できるようになったこと。

言語の進化としてのScalaの功績 5/5

5. 言語内DSLを作りやすい設計

・復活の演算子オーバーロード、ようこそユーザー定義演算子
ユーザー定義型は組み込み型と同じように振る舞えるようにできるという思想の元にC++で採用された演算子オーバーロード。
デバッグしにくいという理由でJavaは不採用とし、「演算子オーバーロードは悪」という文化を産み出してしまった。

Scalaでは、C++(演算子オーバーロード)やHaskell(ユーザー定義演算子)の思想を正しく引き継ぎ[**要出典(*)**]、これらを採用した。
これにより、ベクトルや行列といったクラスを設計する際に演算子が使用できるようになり、言語内DSLのためにユーザーが好きに意味を持たせることのできる演算子を定義できるようになった。

＊ Odersky先生がこのへん何も書いてない・・・

言語の普及のさせ方

- Scalaは、Java/C#と対話しやすい言語を目指し、これらの言語からユーザーを獲得しようと目論んでいる。
- これはC++がCに対して行った戦略と酷似している。
- C++はこの方法によりたしかに普及したが、多くのレガシーコードを産み出してしまった。
- Scalaコミュニティはこの過去から学ばないといけない。

Scalaへの不満／今後に向けて 1/3

1. 可変長型引数がない。
   それを表現するためのコードジェネレート(マクロ)機能が言語に備わっていない。
   メタプログラミング周りのサポートが弱い。
   これを実現している言語は現状、私が知る限りC++とDしかない。Scalaでは可変長型引数がないため、コピペコードが標準ライブラリ内でも量産されている。

```
trait Function0 [+R] extends AnyRef
      Function with 0 parameters.
trait Function1 [-T1, +R] extends AnyRef
      Function with 1 parameter.
trait Function10 [-T1, -T2, -T3, -T4, -T5, -T6, -T7, -T8, -T9, -T10, +R] extends AnyRef
      Function with 10 parameters.
trait Function11 [-T1, -T2, -T3, -T4, -T5, -T6, -T7, -T8, -T9, -T10, -T11, +R] extends AnyRef
      Function with 11 parameters.
trait Function12 [-T1, -T2, -T3, -T4, -T5, -T6, -T7, -T8, -T9, -T10, -T11, -T12, +R] extends AnyRef
      Function with 12 parameters.
trait Function13 [-T1, -T2, -T3, -T4, -T5, -T6, -T7, -T8, -T9, -T10, -T11, -T12, -T13, +R] extends AnyRef
      Function with 13 parameters.
trait Function14 [-T1, -T2, -T3, -T4, -T5, -T6, -T7, -T8, -T9, -T10, -T11, -T12, -T13, -T14, +R] extends AnyRef
      Function with 14 parameters.
trait Function15 [-T1, -T2, -T3, -T4, -T5, -T6, -T7, -T8, -T9, -T10, -T11, -T12, -T13, -T14, -T15, +R] extends AnyRef
      Function with 15 parameters.
trait Function16 [-T1, -T2, -T3, -T4, -T5, -T6, -T7, -T8, -T9, -T10, -T11, -T12, -T13, -T14, -T15, -T16, +R] extends Any
      Function with 16 parameters.
trait Function17 [-T1, -T2, -T3, -T4, -T5, -T6, -T7, -T8, -T9, -T10, -T11, -T12, -T13, -T14, -T15, -T16, -T17, +R] exten
      Function with 17 parameters.
trait Function18 [-T1, -T2, -T3, -T4, -T5, -T6, -T7, -T8, -T9, -T10, -T11, -T12, -T13, -T14, -T15, -T16, -T17, -T18, +R]
      Function with 18 parameters.
trait Function19 [-T1, -T2, -T3, -T4, -T5, -T6, -T7, -T8, -T9, -T10, -T11, -T12, -T13, -T14, -T15, -T16, -T17, -T18, -T1
      Function with 19 parameters.
```

Scalaへの不満／今後に向けて 2/3

2. ScalaのGenericsはそれでもテンプレートに勝てない(まだ発展途上)。
   C++で言うところの部分特殊化、パラメータ化継承を表現できない等、型引数に対する操作が制限されている。
   パラメータ化継承は、いまのErasureベースのGenericsではムリらしい(みずしまさん談)

テンプレートは、不明瞭なコンパイラメッセージの欠陥があるにも関わらず、いまだgenericity hillのkingである。
- Herb Sutter

Scalaへの不満／今後に向けて 3/3

3. 定数式が弱い。
   静的型付け言語の特徴として、コンパイル時にできる限りのエラーをとる、というものがあるが、コンパイル時に可能な限りの計算を行えるのもまた静的型付け言語ならでは。Scalaはそれを生かしきれていない。

   Javaは携帯ゲームの開発といったパフォーマンス、サイズ制限の厳しい環境でも使われているため、Scalaが定数式の強力なサポートを導入すればJavaを置き換える理由になりえる。

   コンパイル時計算はD言語が強力で、文字列操作、数値計算、関数呼び出し、if、foreach等、かなりのことをコンパイル時に解決する。コンパイルタイムデバッガまである。

まとめ

これは私の個人的な見解をまとめたに過ぎません。
マルチパラダイムや言語の進化について考えるには
より多くの意見が必要です。

ぜひみなさんの意見も聞かせてください。