1B-4

動的拡張可能なクラスを持つプログラム言語の考案

髙橋　平†　　筧　捷彦‡

†早稲田大学基幹理工学部情報理工学科　‡早稲田大学理工学術院

1. 概要

世の中には様々な特徴をもつスクリプト言語が存在する。しかし、そのほとんどが独特の構文をもっていて、一般に普及している C 言語や Java などの慣れ親しんだ構文とはかけ離れている。そのため、習得やコードの理解に苦しむプログラマは少なくない。

本研究では、C 言語や Java の文法をベースにクラスの動的拡張、高階関数などの、コードを短く書くための機能を追加したオブジェクト指向型プログラム言語「Garnet」を開発し、既存言語に比べどの程度簡潔に記述できるかを調べた。

2. 既存のオブジェクト指向言語について

Garnet を設計するにおいて、様々な言語の構文について調べた。その中で、オブジェクト指向の概念を持つスクリプト言語を選び、それらの言語の特徴を比較した。比較した言語は、クラスという概念がある(クラスベース)言語として Ruby、Python、ActionScript(3.0 のみ)、クラスという概念のない(プロトタイプベース)言語として JavaScript である。

Ruby、Python、ActionScript は、クラスの動的な拡張が可能な言語である。ただし、Ruby では構文として動的な拡張がサポートされているわけではなく、クラスを拡張するメソッドを呼ぶことで拡張する。ActionScript、Python はクラスの動的な拡張を構文としてサポートしている。ただし、ActionScript ではメソッドの追加のみ可能で、クラス宣言時に「dynamic」を指定する必要がある。

JavaScript は、クラスの概念を持たないプロトタイプベース言語である。プロトタイプベース言語のメソッドの呼び出しには、プロトタイプチェーンと呼ばれる、親へ親へと順にメソッドを検索していく手法が使用される。

クラスベースとプロトタイプベースにはそれぞれ異なった特徴があり、それらの特徴を組み合わせることでより効率よく開発を行うことができるのではないかと考えた。

Development of the programming language possessing dynamic-extensible class

† Taira TAKAHASHI, Department of Computer Science and Engineering, Fundamental Science of Engineering, Waseda University (notes_spinel@akane.waseda.jp)

‡ Katsuhiko KAKEHI, Faculty of Science and Engineering, Waseda University

3. プログラム言語「Garnet」の特徴

本研究では、上記の言語の特徴を踏まえ、以下のような特徴を持つプログラム言語を開発した。

3.1. JavaScript ライクな構文

Garnet は、JavaScript や C#といった、C 言語の構文をベースとするプログラム言語を参考に設計されている。これは、過去にこれらの言語を用いて書かれたコードの再利用をしやすくするためである。また、C 言語や Java などを使用しているプログラマは多く、扱う際に新しい構文の習得を最低限に抑えることができる。

3.2. 動的拡張可能なクラス

クラスとは本来静的なデータ構造であるが、Garnet では、クラスを動的なデータ構造として扱う。クラスを動的なデータ構造とすることで、メソッドやフィールドを後から動的に追加することが可能となり、既存のクラスを柔軟に拡張することができる。これにより、クラスを継承した後に新しくメンバを追加する、といった記述が不必要になるため、コード記述量を抑えることができる。

また、実装には至っていないが、XML ファイル等から DOM ツリー構造を動的に生成し扱う、といった使用方法も考えられる。

3.3. ガーベジコレクション

本研究の直接の目的ではないが、メモリ管理を行うためにガーベジコレクションを実装した。通常は参照カウントによりメモリを管理し、循環参照などでヒープが足りなくなった場合、マーク＆スイープを実行しメモリを解放する。また、マーク＆スイープにより確保できるヒープが少なくなった場合、すぐに次のマーク＆スイープが実行されてしまうのを防ぐため、一定量までヒープサイズを拡大する。

3.4. 高階関数

コードを簡潔に書くための機能の一つとして、高階関数を実装した。また、それらの高階関数を有用に扱えるライブラリを組み込むことで、既存言語よりもコード記述量を少なくすることは可能である。クロージャ、ラムダ式も実装してある。

3.5. オブジェクト

Garnet では、整数、浮動小数点、文字列などを

全てオブジェクトとして実装している。組み込み命令の一部は、対象となる型のメソッド呼び出しに置き換えられる。

```
charAt(str, 5) → str.charAt(5)
```

3.6. 高階関数を有用に扱うライブラリ

先述の通り、Garnet では高階関数を扱うことができ、この高階関数を有用に扱うためのライブラリを多数実装している。例えば、配列の全ての要素に対し繰り返しを行う each メソッド、全ての要素に対し操作を行う map メソッドなどである。もちろん、これらの機能を扱わなくてもプログラム可能であるが、これらの機能を有用に扱うことで、既存言語よりも簡潔に記述することができる。

例)全ての要素を 2 倍にする

```
r = [1, 2, 3].Map($(v){ v * 2; });
```

3.7. 戻り値を返す制御構文

Garnet では、制御構文は戻り値を返す(最後に評価した値を返す)。また、式の途中に組み込むことが可能である。

```
a = if(x) 100 else 0;
```

上記プログラムは、C 言語などで言うところの三項演算にあたる。

4. コード記述量の比較

既存言語と比較して、どの程度短く記述できているかを比較した。比較対象の言語として Ruby、JavaScript、C#、Java などを選び、これらの言語でツェラーの公式を用いてカレンダーを作るプログラムを実装して比較した。結果は次のとおりである。

言語	行数	文字数
プロデル [1]	62 行(258.3%)	992 文字(176.5%)
C#	56 行(233.3%)	943 文字(167.8%)
Java	52 行(216.7%)	899 文字(160%)
JavaScript	39 行(129.4%)	727 文字(129.4%)
Ruby	42 行(175%)	586 文字(104.3%)
Garnet	24 行(100%)	562 文字(100%)

(括弧内は、Garnet を 100%としたとき値を示す)

記述した処理にもよるが、どの言語よりも記述量は大幅に減った。記述量が大幅に減ったものでは、約 1/3 程度の記述量で同等のプログラムを書くことが出来ていた。

5. 問題点

Garnet を設計するに当たって、いくつかの問題点が見つかった。

5.1. 可読性の問題

Garnet では、コードを短く書くことに重点をおいている。しかし、コードを短く書くということは、同時に可読性の低下を招く。また、関数や演算による副作用の発生なども考えられる。

しかし、この問題はどのプログラム言語にでも言えることであり、大抵の場合、コードを短く書くことは可読性の低下につながる。

Garnet では、簡潔に記述できる構文の他に、C 言語等で記述できるような単純な構文もサポートしている。したがって、一度しか使わないコードや、自分のみが使うようなコードの場合は簡潔に、他人に公開する必要がある、あるいは可読性が重視される場合は既存言語が持つ制御構文等で記述することが可能である。

5.2. 意図しないクラスの拡張

クラスの動的拡張は、コードを効率良く使い、かつ短く書くために有用である。しかし、プログラマの意図しない場所でのクラスの拡張が発生してしまった場合、その修正は非常に困難である。

したがって、Garnet ではクラスの動的拡張を制限する機能をいくつか実装した。

通常、クラスは動的拡張可能な状態で定義される。クラス定義時、fixed を指定することでそのクラスは動的拡張が不可能なクラスとなる。これにより、クラスの意図しない場所での動的拡張を防ぐことが可能である。また、fixed はメンバ毎に設定することが可能であり。特定のメンバのみの動的拡張を防ぐことも可能である。

6. 結論

今回開発した Garnet では、C 言語や Java などの言語が持つ構文を継承しつつ、独自の機能を実装することで、コードを簡潔に記述することが可能となった。また、クラスの動的拡張を導入することによって、既存のコードをより柔軟かつ簡潔に拡張することが可能となった。

使い捨てプログラムや小規模なプログラムを記述する場合、Garnet は非常に効率よくプログラムを記述することのできるプログラム言語であると言える。

参考文献

1. ゆうと. 日本語プログラミング言語「プロデル」. (オンライン) (引用日: 2011 年 1 月 11 日) http://rdr.utopiat.net/.