

Ruby入門1

~Rubyの基礎を学んで使ってみよう~

課外プロジェクト 2014 4/27

d8161105 渡部未来



◇ 目的

- ▶ *Ruby*の特徴と基本的な書き方を学ぶ
- ▶ プログラミング言語の様々な特徴を学び、メリット・デメリットを学ぶ
- ▶ プログラミング言語の得手不得手を考える

※本資料は、Ruby 1.9.3について解説する。

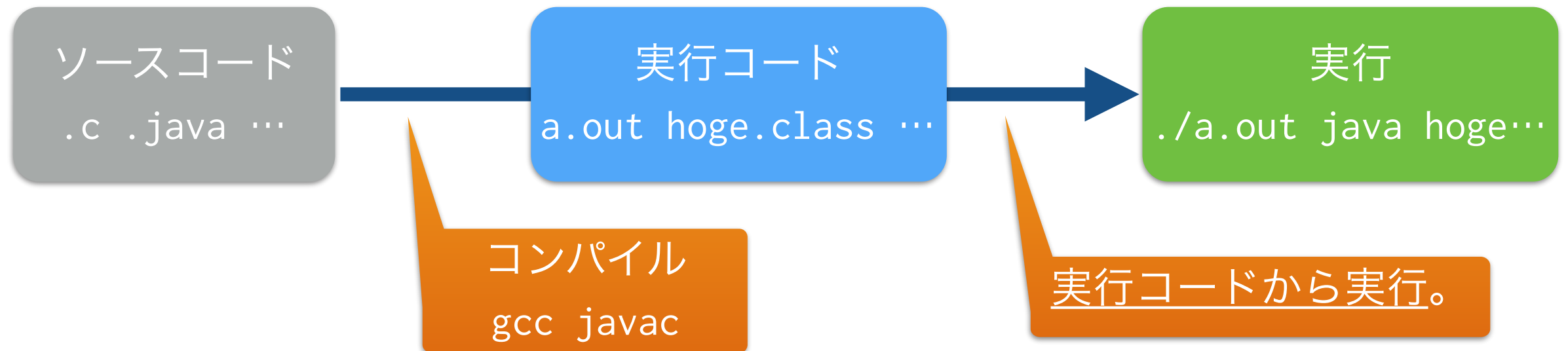
リファレンスマニュアル: <http://docs.ruby-lang.org/ja/1.9.3/doc/index.html>

◇ プログラミング言語 *Ruby*

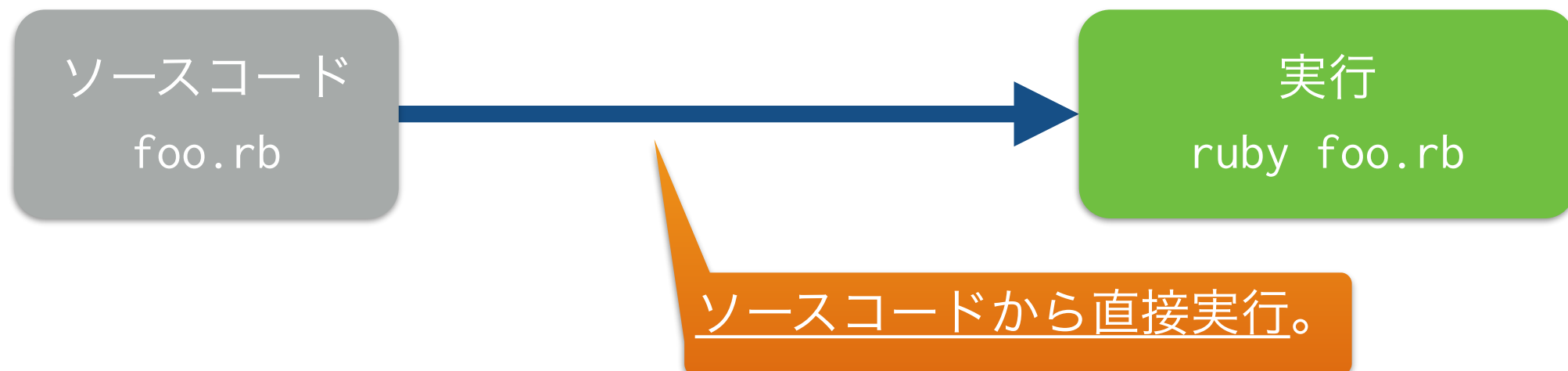
- ▶ まつもとゆきひろさん(*Matz*)によって開発された[汎用プログラミング言語](#)
- ▶ 国産のプログラミング言語なので日本語資料が多い
- ▶ [インタプリタ言語](#)
- ▶ [動的型付け言語](#)
- ▶ 文法・ライブラリが豊富
- ▶ [純粋なオブジェクト指向言語](#)
- ▶ Webアプリケーション・フレームワーク [Ruby on Rails](#)によりヒット

◇ *Ruby* | インタプリタ言語

コンパイラ言語:



インタプリタ言語(*Ruby*):



◇ *Ruby* | 実行の仕方

対話的実行環境 (*irb*) :

簡単なオブジェクトの確認や
簡単な文法の確認などに
用いると便利。


```
$ irb
irb(main):001:0> "string".upcase
=> "String"
```

インタプリタ (*ruby*) :

```
array = [1, 2, 3, 4]

array.each do |elm|
  puts elm
end
```

まとまった処理の確認や
実際の開発時に用いる。



```
$ ruby array.rb
1
2
3
4
```

◇ *Ruby* | 動的型付け言語

静的型付け言語：

型宣言をする。(例外あり)
コンパイル時に型が決まる。

```
int x;    // 整数型  
char c;   // 文字型
```

例: *Fortran, C, C++, Java, Objective-C*

動的型付け言語 (*Ruby*)：

変数に代入された値で型が決まる。
実行時に型が決まる。

```
x = 1      // 整数型  
x = 'aa'   // 文字型
```

例: *JavaScript, Perl, Python, PHP, Ruby*

◇ Ruby | オブジェクト指向言語

- ▶ Rubyは純粋なオブジェクト指向言語
- ▶ データは、すべてオブジェクトである
- ▶ オブジェクトはアイデンティティを持つ
- ▶ オブジェクトは状態 (Field)と振る舞い (Method)を持つ
- ▶ 状態は、いわゆる変数のこと
- ▶ 振る舞いは、いわゆる関数のこと

◇ *Ruby* | オブジェクト指向

非オブジェクト指向言語:

データの計算処理は、
関数・演算子に任せる。

```
strlen( "string" )    // 6  
atoi( "123" )        // 123  
toupper( "abc" )      // ABC
```

オブジェクト指向言語 (*Ruby*):

データはオブジェクト。
オブジェクトの計算処理は、
フィールド・メソッドを使用する。

```
"string".length      // 6  
"123".to_i           // 123  
"abc".upcase         // ABC
```


◇ オブジェクト指向サンプル

- ▶ 簡単な集計プログラムを考える
- ▶ csv (カンマ区切りの値) ファイルを読み込み、二次元配列へ落とし込む
- ▶ 二次元配列から集計をする

◇ オブジェクト指向サンプル | 非オブジェクト指向 その1

- ▶ グローバル変数は、いつ・どこで書き換えられているか分からないため危険
- ▶ もし、扱うcsvが複数になったら…？ (row1, row2などのグローバル変数が増加、管理が難しくなる)

```
#define N 100
// グローバル変数
int row;
int column;

void input(int[][]);
int calc(int[][]);

int main(){
    int res;
    int col[N][N];
    input(col); // row, columnが確定
    res = calc(col);
}
```

◇ オブジェクト指向サンプル | 非オブジェクト指向 その2

- ▶ 構造体で扱いたいメンバが増加していくと管理が困難に
- ▶ 作成した構造体 (型) を扱う専用の関数を用意しなければならない

```
#define N 100
typedef struct {
    int col[N][N];
    int row, column;
} Grade;

void input(Grade*);
int calc(Grade*);

int main(){
    int res;
    Grade *grade;
    input(grade);
    res = calc(grade);
}
```

◇ オブジェクト指向サンプル | オブジェクト指向(*Ruby*)

- ▶ フィールドは、グローバル変数と異なり自身のクラスにスコープが限定される
- ▶ 振る舞いを定義することで、オブジェクト専用の関数を作ることができる
- ▶ 新しい型を作ったので、インスタンス化することで量産が可能

```
class Grade
  def initialize(csv)          // コンストラクタ
    @data = csv.getDate       // csvデータから二次元配列を取得
                              // 本来は自分で実装
    @row = csv.getRow         // 行数を取得、本来は自分で実装
    @column = csv.getColumn   // 列数を取得、本来は自分で実装
  end
  def calc
    @row                      // フィールドはクラス内なら自由に取得できる
    @column
  end
end

grade = new Grade(csv)       // インスタンス化すれば、複数csvも扱える
res = grade.calc
```

◇ オブジェクト指向 | まとめ

- ▶ オブジェクト指向言語では、簡単に**状態と振る舞い**を持った新しい型(オブジェクト)を作ることができる
- ▶ グローバル変数を用いなくとも、状態 (フィールド)を用いることで、オブジェクト内なら自由に参照できる
- ▶ 新しい型を扱う専用の関数を作らなくとも、振る舞い (メソッド)を定義することで、共通処理が記述できる
- ▶ 新しく定義したオブジェクトは、インスタンス化することで、処理を再利用することができる

◇ Ruby | まとめ

- ▶ コンパイルという実行確認のステップがないため、手軽に学習・開発がおこなえる
- ▶ 型を明記する必要がない(型に気をつけなくていいわけではない)のでコード量が減り、開発速度が上がる
- ▶ 手軽だが、豊富なライブラリ・文法、オブジェクト指向が備わっているので、しっかりした開発もおこなえる

◇ Ruby | 学習材料

- ▶ *Ruby*入門: <http://www.rubylife.jp/ini/>
- ▶ *Ruby*リファレンスマニュアル: <http://docs.ruby-lang.org/ja/1.9.3/doc/index.html>
- ▶ 初めての*Ruby*: <http://www.amazon.co.jp/dp/4873113679>