Juliaの型システムの研究

寺崎 敏志

Julia型のシステム

● 型を明示する目的

ここまでは関数の引き数や変数に型を書きませんでした．Juliaでは型を明示しなくても，入力として受け取った値の型を調べ，その型に応じてコンパイルを行います．では，どのような場合に型を明示する必要があるのでしょうか？ JuliaのドキュメントのTypes（型）の部分から一部を引用します．

「型を明示するのは3つの目的がある．多重ディスパッチと可読性の向上および，プログラムのミス抑制である．」

パフォーマンスに言及していないのは面白いですね．

● 引き数の型に応じて実行するメソッドを選ぶ仕組み「多重ディスパッチ」

多重ディスパッチ（multiple-dispatch）は関数に与える引き数の数，型の組み合わせによって，どのメソッドを実行するのかを決定する仕組みです．Juliaでは見た目が同じ識別子のコードでも，入力する型に応じて違う振る舞いを見せる関数がたくさんあります．

関数とメソッドという似たような言葉を使ったことに気をつけてください．ここでは関数は「入力を受け取りそれに対応する値を返す抽象的な仕組み」として用いています．メソッドは具体的な振る舞いや実装を指しています．

乱数生成用のrand関数を例にとります．

```
julia> length(methods(rand))
```

を実行すると，rand関数の情報が表示され80個のメソッドを持っていることが分かります．

ソースコードを眺めると，引き数と返り値の型の組み合わせによってさまざまなメソッドが実装されています．その一部を**リスト1**に示します．

リスト1　引き数の型や組み合わせで動作が異なる同じ名前の関数が複数用意されている

```
julia> rand()                  # Float64 の型の乱数を1つ得る
julia> rand(Float32, 3)
          # 要素数が3の配列に乱数が各々入る．型は Float32.
julia> rand(Float64, (3,3))        # 3×3の乱数行列を生成
julia>       #内部で上と同じメソットが呼び出されるように帰着される
julia> rand(Float64, 3, 3)         # 3×3の乱数行列を生成
julia> rand([1,2,3,4,5],2)    # 1～5の中から2つ要素を選ぶ
```

● Pythonと比べる

一方，Pythonは多重ディスパッチとは反対に，シングル・ディスパッチであると言えます．クラスを作り，その中にメソッドを記述します．オブジェクトとメソッドが密接につながっています．

例えば，NumPy，SymPy，PyTorchにおいては，numpy.exp，sympy.exp，torch.expのように，おのおののオブジェクトに対して指数関数の実装が提供されています．

● 多重ディスパッチのメリット

多重ディスパッチのメリットは，汎用性のある自作の演算子，関数を定義するときに発揮されます．

数学の拡張概念であるDual Numberを用いた微分の実装を考えます．関数fに対してxにおける値f(x)と導関数f'(x)の値を保持したクラスに対して四則演算，べきの演算を実装することで，それらを組み合わせた関数の微分の値を計算できます．Pythonで実装したものが**リスト2**(a)です．

一方，Juliaで書くと**リスト2**(b)のようにできます．

コードの分量も全然違いますし，2つのDual Numberに対しての演算子の定義を関数として実装し，数学的な定義と同じような見た目にできます．

一方でPythonではどうでしょうか．クラスDに紐づいている四則演算を与える特殊メソッドが，第1引き数にくるf（通常はselfとよく書かれる）と第2引き数のg（otherのようなもの）を対等に扱うことができず，通常は変数の順序を入れ替え可能と期待される演算子についても，

```
__radd__
```

という冗長な定義を書かなければいけません（他の演算子についても同様）．

ここでgが通常の数である場合，定数関数D（スカ