感想より

- ORMしかしらず素のSQLが書けない人が多いという話があって、自分のことだと思って、ぎくっときた。この手の議論は永遠に続きそうだなぁと思う。… 時代の流れとともに熟知すべきとされる低レイアーはどんどん上がっていくのかもなぁ
- As my professor from college was saying, if you learned C, you can pick up any other language easily, but if you learned java, you only learned java

Shigeru Chiba

15

感想より

- ActiveRecordだとSQLで書けること全部はサポートできてないイメージがあります。複雑なことをやろうとすると結局SQLを理解することになる羽目になります。
- 「静的型の言語ではDBの処理にまともな型がつかない」の件ですが，少なくともSQL文を文字列より高級な形で与えさえすればレコード型などを用いて型がつけられる気がします
- RDBMS側で型のついたインターフェースを提供すれば静的型付き言語からも綺麗に操作できる

Shigeru Chiba

16

感想より

- Gradual Typing 早く流行ってほしいです
- C#だとdynamicを使って静的型チェックを意図的に省略できますよね。最近Gradual Typingの話題をよく見ますし、静的型付けと動的型付けのいいとこどりがトレンドになっている気がします。

Shigeru Chiba

17

Static or dynamic?

- Static
  - Pro. compile-time error detection, types help code reading.
  - Con. writing type names is troublesome.
- Dynamic
  - Pro. programs are often short and simple.
  - Con. runtime errors.

Shigeru Chiba

18

In Web programming

- HTTP, SQL , Java Script
  - Dynamically typing

- The rest
  - Static or dynamic?

Writing a type is troublesome.

- A closure if explicit typing is needed

```
{ Student => boolean } filter
  = {(Student s) => s.dept == “creative informatics” };
```

return

parameter

body

```
Function<Student,Boolean> filter
  = s -> s.dept == “creative informatics”;
```

X Student

Type inference

- A program is statically typed but no need to write type names.
  - A compiler automatically fills them.

```
// in Scala

def add(x: Int, y: Int): Int = x + y

def sloppyAdd(x: Int, y: Int) = x + y
```

{ return x+y; }

Shigeru Chiba

21

LINQ (Language INtegrated Query)

- A SQL part is embedded and so typed.

```
UnivDataContext dc = new UnivDataContext();

IEnumerable<Student> rs =
   from s in dc.Students
   where s.dept == "creative informatics"
   select s;

foreach (Student r in rs) {          // now executed
   print(r.id, r.name)
}
```

for each s in

Shigeru Chiba

22

LINQ

- An implicit type var.

```
UnivDataContext dc = new UnivDataContext();

var rs =
   from s in dc.students
   where s.dept == “creative informatics”
   select s;

foreach (var  r  in  rs) {
   print(r.id, r.name)              // r is typed.
}
```

Shigeru Chiba

LINQ

Student{+id, + name}

- An anonymous type.

alias

```
UnivDataContext dc = new UnivDataContext();

var rs =
   from s in dc.students
   where s.dept == “creative informatics”
   select new { num = s.id, who = s.name };

foreach (var r  in  rs) {     // now executed
   print(r.num, r.who)
}
```

Shigeru Chiba

LINQ

- Method-based query syntax.
  - A lambda-expression represents an action.
    - but only a limited kind of lambda-exprs.

```
UnivDataContext dc = new UnivDataContext();

var rs = dc.students.
   Where(s => s.dept == "creative informatics").
   Select(s => new { num = r.id, who = r.name });

foreach (var r  in  rs) {     // now executed
   print(r.num, r.who)
}
```

Shigeru Chiba

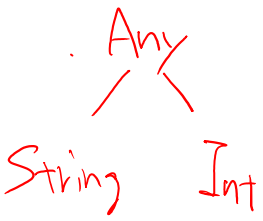
25

Dynamic == Any type?

- What is the type of x? Any type?

In Java

- x = "three"
- print x.length()
- x = 3
- print x * 2



Shigeru Chiba

26

Static vs. Dynamic

- Statically typed
  - All expressions are statically typed.
    - explicitly or implicitly
    - (usually) at compile time
- Dynamically typed
  - anything but statically typed ones?
  - “Write it, run it, and pray” languages?

Shigeru Chiba

Types

- The type of an expression
  - the range of expected values that the expression will result in.

Shigeru Chiba

Types

- The type of a function/method
  - the range of the arguments that a function that runs without type errors.
  - The range of the return value.

Type checking

- Apply typing rules

$\Gamma \vdash e1 : int$

$\Gamma \vdash e2 : int$

$\Gamma \vdash e1 > e2 : bool$

```
int test() { return fact(5); }

int fact(int n) {
   if (n > 1)
      return fact(n – 1) * n;
   else
      return 1;
}
```

$$\frac{\Gamma \vdash e1 : int \qquad \Gamma \vdash e2 : int}{\Gamma \vdash e1 > e2 : bool}$$

Parametric polymorphism

- An identity function

  int a = identity(3)

  - ?? identity(?? x) { return x; }
  - This is a perfectly correct function but not typable? Not reusable!

Shigeru Chiba

Java Generics

- An identity function
  - as a generic method
  - <T> T identity(T x) { return x }
  - T is a type parameter.
    - String s = identity("java");
    - int i = identity(7);

Shigeru Chiba

A subtype

A type is a range of values. So, a subtype is a subset of some type.

- Behavioral subtype
  - by Liskov's substitution principle
  - A value of type T can be replaced with a value of a subtype of T.

A subclass

- A subclass is often a subtype of its super class.

A Point3D object can substitute for a Point object.

```
class Point {
   int x, y;
}

class Point3D extends Point {
   int z;
}
```

```
Point3D p3 = new ....
Point p = p3;
int value = p.x;
```

Subtyping is difficult

Int[] <: Num[]

- A subtype of array type
  - Integer is a subtype of Number
  - Integer[] nat = new Integer[] { 1, 2, 3 };
    Number[] num = nat; // OK?

Float

3.1

num[2] = new Float()

Integer i = nat[2]

Shigeru Chiba