Metaprogramování

Multiparadigmatický přístup v softwarovém inženýrství

Kyjev, 2015 — 2022

Anotace

Všechny programy jsou data. Některá data jsou interpretována jako hodnoty, jiná jako typy těchto hodnot a další jako instrukce pro zpracování prvních dvou. Jakákoli paradigmata a programovací techniky jsou jen způsobem, jak vytvořit metadata, která dávají pravidla a posloupnost toku zpracování jiných dat. Multiparadigmatické programování bere to nejlepší ze všech paradigmat a sestavuje z něj syntaktické konstrukce, které umožňují popsat předmětnou oblast jasněji a pohodlněji. Vysokoúrovňové DSL (doménové jazyky) promítáme do nízkoúrovňových strojových instrukcí prostřednictvím mnoha vrstev abstrakcí. Zde je důležité reprezentovat úkol co nejúčinnějším způsobem pro zpracování na úrovni stroje, nikoli fanaticky následovat jedno paradigma. Nejúčinnější je ten, který s menším počtem vrstev a závislostí, nejlépe čitelný, udržovatelný a upravitelný pro člověka, který zajišťuje spolehlivost kódu a testovatelnost, rozšiřitelnost, opětovnou použitelnost, jasnost a flexibilitu konstrukcí metadat na každé úrovni. Věříme, že takový přístup nám umožní získat jak rychlé první výsledky ve vývoji, tak i neztrácet výkon při velkém toku změn ve fázích, kdy projekt již dosáhl vysoké zralosti a složitosti. Pokusíme se zvážit techniky a principy různých programovacích paradigmat prizmatem metaprogramování a změnit jestli ne samotné softwarové inženýrství tím, tak aspoň rozšířit pochopení disciplíny novými generacemi profesionálů.

Obsah

1. Úvod

Neustálé přehodnocování své činnosti, i té nejjednodušší, by mělo provázet inženýra celý život. Zvyk zapisovat si své myšlenky slovy a zdokonalovat své formulace v tom hodně pomáhá. Tento text se objevil jako moje fragmentární poznámky, psané v různých letech, které jsem shromažďoval a mnohokrát kriticky korigoval. Já jsem často nesouhlasil sám se sebou, když jsem si znovu četl pasáž poté, co chvíli ležela. Proto jsem na textu pracoval tak dlouho, dokud jsem sám nesouhlasil s tím, co bylo napsáno po dlouhé době držení materiálu. Bylo mým úkolem psát co nejstručněji a velké fragmenty jsem opakovaně přepisoval, když jsem zjistil, že by se daly vyjádřit stručněji. Struktura textu a obsah se začaly objevovat po prvním roce výuky, ale v desátém ročníku jsem se rozhodl šířit materiály nejen ve formě otevřených videopřednášek, jak jsem to dělal už asi pět let, ale i ve formě textu. To umožnilo všem ze společenství Metarhia podílet se na tvorbě knihy, díky čtenářům rychle objevovat překlepy a nepřesnosti, a také pro mnohé je formát knihy prostě pohodlnější. Aktuální verzi naleznete na https:// github.com/ HowProgrammingWorks/ Book, bude neustále aktualizována. Žádosti o opravy a doplnění směřujte v angličtině na [issues] (https:// github.com/ HowProgrammingWorks/ Book/ issues), nové nápady zasílejte v libovolném jazyce na [discussions] (https:// github.com/ HowProgrammingWorks/ Book/ discussions), vaše vlastní doplňky a opravy by měly být provedeny formou pull-request do repositáře knihy.

1.1. Přístup k výuce programování

No translation

1.2. Příklady v jazycích JavaScript, Python a C

No translation

```
let first_num = 2;
let secord_num = 3;
let sum = firstNum + secondNum;
console.log({ sum });
```

```
#include <stdio.h>
```

```
int main() {
  int first_num = 2;
  int secord_num = 3;
  int sum = firstNum + secondNum;
  printf("%d\n", sum);
}
```

```
first_num = 2;
secord_num = 3;
sum = firstNum + secondNum;
print({ 'sum': sum  });
```

1.3. Modelování: abstrakce a opětovné použití

No translation

1.4. Algoritmus, program, syntaxe, jazyk

No translation

1.5. Dekompozice a separace odpovědnosti

No translation

1.6. Přehled specializace softwarového inženýra

No translation

1.7. Přehled programovacích paradigmat

No translation

2. Základní pojmy

No translation

```
// Single-line comment
```

```
/*
  Multi-line
  comments
*/
```

```
# Single-line comment
```

```
"""
  Multi-line
  comments
"""
```

```
select name from PERSON -- comments in sql
```

```
<!-- commented block in xml and html -->
```

```
; Single-line comment in Assembler and LISP
```

2.1. Hodnota, identifikátor, proměnná a konstanta, literál, přiřazení

No translation

2.2. Datové typy, skalární, referenční a strukturované typy

No translation

2.3. Kontexty a lexikální rozsah

No translation

2.4. Operátor a výraz, blok kódu, funkce, smyčka, podmínka

No translation

2.5. Procedurální paradigma, volání, zásobník a halda

No translation

2.6. Funkce vyššího řádu, čistá funkce, vedlejší účinky

No translation

2.7. Uzávěry, funkce zpětného volání, zabalení a události

No translation

2.8. Výjimky a řešení chyb

No translation

2.9. Monomorfní kód v dynamických jazycích

No translation

3. Stav aplikace, datové struktury a kolekce

No translation

3.1. Stavové a bezstavové přístupy (stateful and stateless)

No translation

3.2. Struktury a záznamy

```
#include <stdio.h>

struct date {
  int day;
  int month;
  int year;
};

struct person {
  char *name;
  char *city;
  struct date born;
};

int main() {
  struct person p1;
  p1.name = "Marcus";
  p1.city = "Roma";
  p1.born.day = 26;
  p1.born.month = 4;
  p1.born.year = 121;

  printf(
    "Name: %s\nCity: %s\nBorn: %d-%d-%d\n",
    p1.name, p1.city,
    p1.born.year, p1.born.month, p1.born.day
  );
```

```
  return 0;
}
```

Pascal

```
program Example;

type TDate = record
  Year: integer;
  Month: 1..12;
  Day: 1..31;
end;

type TPerson = record
  Name: string[10];
  City: string[10];
  Born: TDate;
end;

var
  P1: TPerson;
  FPerson: File of TPerson;

begin
  P1.Name := 'Marcus';
  P1.City := 'Roma';
  P1.Born.Day := 26;
  P1.Born.Month := 4;
  P1.Born.Year := 121;
  WriteLn('Name: ', P1.Name);
  WriteLn('City: ', P1.City);
  WriteLn(
    'Born: ',
    P1.Born.Year, '-',
    P1.Born.Month, '-',
    P1.Born.Day
  );
  Assign(FPerson, './record.dat');
```

```
  Rewrite(FPerson);
  Write(FPerson, P1);
  Close(FPerson);
end.
```

Rust

```
struct Date {
    year: u32,
    month: u32,
    day: u32,
}

struct Person {
    name: String,
    city: String,
    born: Date,
}

fn main() {
    let p1 = Person {
        name: String::from("Marcus"),
        city: String::from("Roma"),
        born: Date {
            day: 26,
            month: 4,
            year: 121,
        },
    };

    println!(
        "Name: {}\nCity: {}\nBorn: {}-{}-{}\n",
        p1.name, p1.city,
        p1.born.year, p1.born.month, p1.born.day
    );
}
```

TypeScript: Interfaces

```
interface IDate {
  day: number;
  month: number;
  year: number;
}

interface IPerson {
  name: string;
  city: string;
  born: IDate;
}
```

```
const personToString = (person: IPerson): string => {
  const { name, city, born } = person;
  const { year, month, day } = born;
  const fields = [
    `Name: ${name}`,
    `City: ${city}`,
    `Born: ${year}-${month}-${day}`,
  ];
  return fields.join('\n');
};
```

```
const person: IPerson = {
  name: 'Marcus',
  city: 'Roma',
  born: {
    day: 26,
    month: 4,
    year: 121,
  },
};

console.log(personToString(person));
```

TypeScript: Classes

```
class DateStruct {
  day: number;
  month: number;
  year: number;
}

class Person {
  name: string;
  city: string;
  born: DateStruct;
}
```

JavaScript: Classes

```
class DateStruct {
  constructor(year, month, day) {
    this.day = day;
    this.month = month;
    this.year = year;
  }
}

class Person {
  constructor(name, city, born) {
    this.name = name;
    this.city = city;
    this.born = born;
  }
}
```

```
const personToString = (person) => {
  const { name, city, born } = person;
  const { year, month, day } = born;
  const fields = [
    `Name: ${name}`,
    `City: ${city}`,
    `Born: ${year}-${month}-${day}`,
  ];
```

```
  return fields.join('\n');
};
```

```
const date = new DateStruct(121, 4, 26);
const person = new Person('Marcus', 'Roma', date);
console.log(personToString(person));
```

JavaScrip: Objects

```
const person = {
  name: 'Marcus',
  city: 'Roma',
  born: {
    day: 26,
    month: 4,
    year: 121,
  },
};

console.log(personToString(person));
```

JavaScrit: strucute serialization

```
const v8 = require('v8');
const fs = require('fs');
```

Take from previous example:

- class DateStruct - class Person

```
const date = new DateStruct(121, 4, 26);
const person = new Person('Marcus', 'Roma', date);

const v8Data = v8.serialize(person);
const v8File = './file.dat';
fs.writeFile(v8File, v8Data, () => {
```

```
  console.log('Saved ' + v8File);
});
```

File: file.dat

```
FF 0D 6F 22 04 6E 61 6D 65 22 06 4D 61 72 63 75
73 22 04 63 69 74 79 22 04 52 6F 6D 61 22 04 62
6F 72 6E 6F 22 03 64 61 79 49 34 22 05 6D 6F 6E
74 68 49 08 22 04 79 65 61 72 49 F2 01 7B 03 7B
03
```

Nested structures

```
#include <stdio.h>
#include <map>
#include <string>
#include <vector>

struct Product {
  std::string name;
  int price;
};

void printProduct(Product item) {
  printf("%s: %d\n", item.name.c_str(), item.price);
}

void printProducts(std::vector<Product> items) {
  for (int i = 0; i < items.size(); i++) {
    printProduct(items[i]);
  }
}

int main() {
  std::map<std::string, std::vector<Product>> purchase {
    { "Electronics", {
      { "Laptop", 1500 },
```

```
      { "Keyboard", 100 },
      { "HDMI cable", 10 },
    } },
    { "Textile", {
      { "Bag", 50 },
    } },
  };

  std::vector electronics = purchase["Electronics"];
  printf("Electronics:\n");
  printProducts(electronics);

  std::vector textile = purchase["Textile"];
  printf("\nTextile:\n");
  printProducts(textile);

  Product bag = textile[0];
  printf("\nSingle element:\n");
  printProduct(bag);

  int price = purchase["Electronics"][2].price;
  printf("\nHDMI cable price is %d\n", price);
}
```

Python

```
purchase = {
  'Electronics': [
    { 'name': 'Laptop', 'price': 1500 },
    { 'name': 'Keyboard', 'price': 100 },
    { 'name': 'HDMI cable', 'price': 10 },
  ],
  'Textile': [
    { 'name': 'Bag', 'price': 50 },
  ],
}

electronics = purchase['Electronics']
print({ 'electronics': electronics })
```

```
textile = purchase['Textile']
print({ 'textile': textile })

bag = textile[0]
print({ 'bag': bag })

price = purchase['Electronics'][2]['price']
print({ 'price': price })
```

JavaScript

```
const purchase = {
  Electronics: [
    { name: 'Laptop', price: 1500 },
    { name: 'Keyboard', price: 100 },
    { name: 'HDMI cable', price: 10 },
  ],
  Textile: [{ name: 'Bag', price: 50 }],
};

const electronics = purchase.Electronics;
console.log(electronics);

const textile = purchase['Textile'];
console.log(textile);

const bag = textile[0];
console.log(bag);

const price = purchase['Electronics'][2].price;
console.log(price);

const json = JSON.stringify(purchase);
console.log(json);
const obj = JSON.parse(json);
console.log(obj);
```

3.3. Pole, seznam, sada, n-tice

No translation

3.4. Slovník, hashovací tabulka a asociativní pole

No translation

3.5. Zásobník, fronta, deque

No translation

3.6. Stromy a grafy

No translation

3.7. Projekce a zobrazení datových

No translation

3.8. Odhad výpočetní složitosti

No translation

4. Rozšířené koncepty

No translation

4.1. Co je technologický stack

No translation

4.2. Vývojové prostředí a ladění

No translation

4.3. Iterace: rekurze, iterátory a generátory

No translation

4.4. Stavební bloky aplikací: soubory, moduly, komponenty

No translation

4.5. Objekt, prototyp a třída

No translation

4.6. Částečná aplikace a curryfikace, skládání funkcí

No translation

4.7. Řetězení pro metody a funkce (chaining)

No translation

4.8. Mixiny (mixins)

No translation

4.9. Závislosti a knihovny

No translation