Metaprogramovanie

AOP — pokračovanie

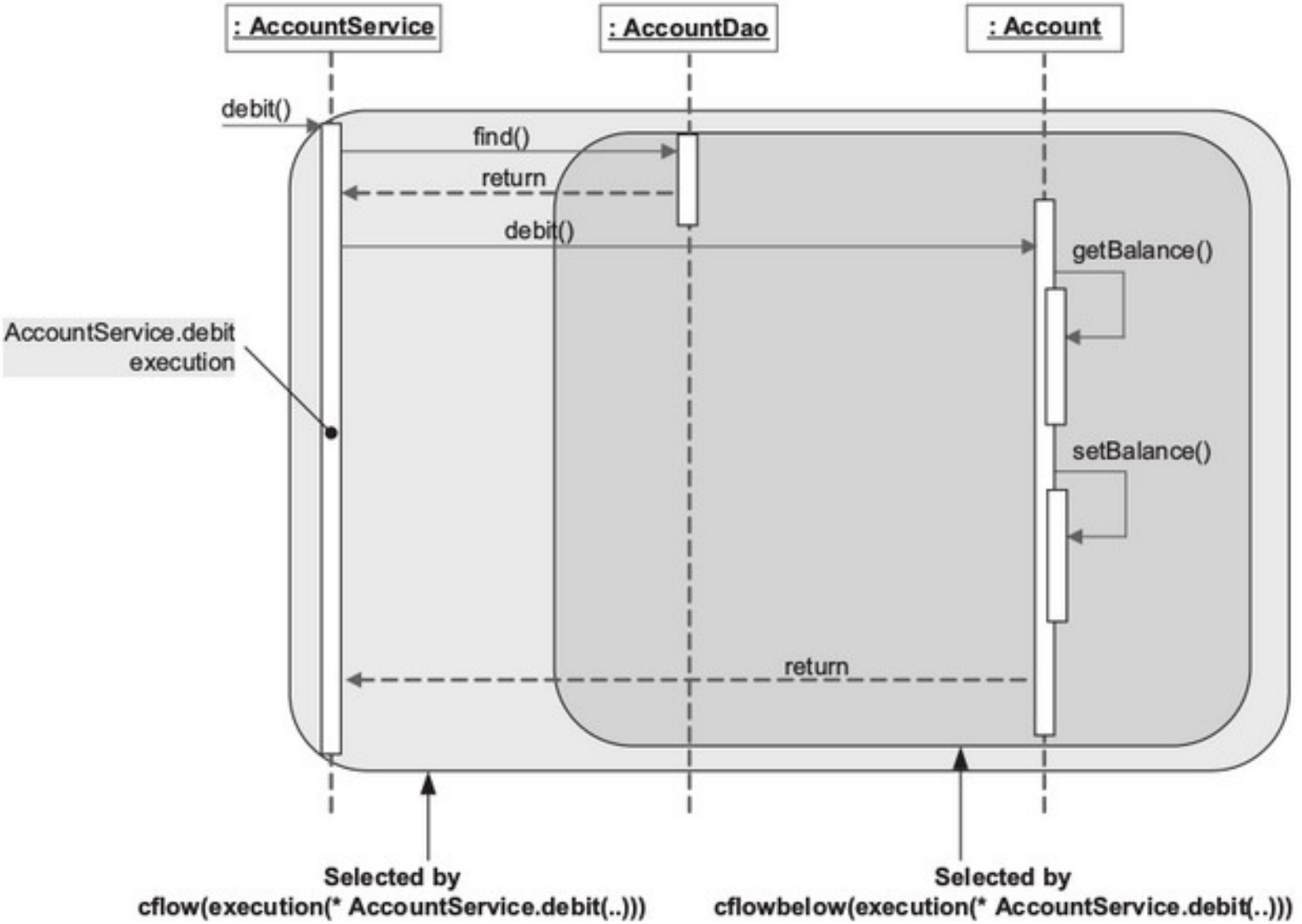
Sergej Chodarev

12

Ďalšie bodové prierezy

Prierez podľa toku riadenia

- **cflow** («bodový prierez»)
- **cflowbelow** («bodový prierez»)



Ramnivas Laddad. AspectJ in Action, Second Edition (Manning 2009)

Iba prvé volanie v rekurzií

```
public class Factorial {
    public long factorial(int n) {
        if (n == 0) {
            return 1;
        }
        return n * factorial(n – 1);
    }
}
```

Iba prvé volanie v rekurzií

```
pointcut factorial(int n)
    : call(long Factorial.factorial(int)) && args(n);

before(int n)
    : factorial(n) && !cflowbelow(factorial(int)) {
    System.out.println("Calculating factorial for " + n);
}
```

Zrušenie vstupu a výstupu

```
public aspect NoIOAspect {
    Object around() :
      call(* java.io..*(..)) && cflow(execution(@NoIO * *(..))) {
        return null;
   }
}
```

Prierez podľa lexikálnej štruktúry

- **within** («vzor typu»)
- **withincode** («vzor metódy / konštruktora»)
- aj vnorené triedy

Iba podtriedy

```
this(Typ) && !within(Typ)
```

Vynechanie aspektu

```
public aspect PrintRecAspect {
    pointcut printTest() : call(* println(..));

    before() : printTest() {
        System.out.println("In Advice ...");
    }
}
```

Riešenie

```
public aspect PrintRecAspect {

    pointcut printTest() : call(* println(..))
         && !within(PrintRecAspect);


    before() : printTest() {
        System.out.println("In Advice ...");
    }

}
```

Alternatívne riešenie

```
public aspect PrintRecAspect {

    pointcut printTest() : call(* println(..))
         && !cflow(adviceexecution());


    before() : printTest() {
        System.out.println("In Advice ...");
    }

}
```

Prierez podľa anotácie

- @this
- @target
- @args
- @within
- @withincode
- **@annotation**

Prierez podľa podmienky

- **if** («výraz»)

Príklad: Cache

@AspectJ

@AspectJ

- Alternatívna syntax — Java triedy s anotáciami
- Kompilované pomocou kompilátora Java
- Pretkávanie ako samostatný krok
- Bez potreby špeciálnej podpory od IDE

Príklad

```
@Aspect
public class LoggingAspect {

    private Logger logger;
    public LoggingAspect() { logger = new Logger(); }

    @Pointcut("execution(* Product.setPrice(..)) && this(product)"
                                                + "&& args(price)")
    public void priceChange(Product product, BigDecimal price) {}

    @Before("priceChange(product, price)")
    public void log(Product product, BigDecimal price) {
        logger.writeLog("Changed " + product.getPrice() + " to " + price);
    }
}
```

After

```
@Aspect
public class ConnectionMonitor {

    @Pointcut("call(* java.sql.Connection.*(..)) && target(connection)")
    public void connectionOperation(Connection connection) {}

    @AfterReturning("connectionOperation(connection)")
    public void monitorSuccessfulUse(Connection connection) {
        System.out.println("Just used " + connection
                           + " successfully");
    }

    @AfterThrowing("connectionOperation(connection)")
    public void monitorFailedUse(Connection connection) {
        System.out.println("Just used " + connection
                           + " but met with a failure");
    }

}
```

Around

```
@Aspect
public abstract class Monitoring {

    @Pointcut
    public abstract void monitored();

    @Around("monitored()")
    public Object measureTime(ProceedingJoinPoint pjp) throws Throwable {
        long startTime = System.nanoTime();
        Object ret = pjp.proceed();
        long endTime = System.nanoTime();
        System.out.println("Method " + pjp.getSignature().toShortString()
            + " took " + (endTime-startTime));
        return ret;
    }
}
```

Statické pretínanie

Medzitypové deklarácie

- *Intertype declarations (ITD)*
- Definovanie členskej premennej alebo metódy pre ľubovoľný typ
- `public String Customer.name;`
- Iba 1 typ
  - alebo rozhranie

```
public interface Nameable {
    public void setName(String name);
    public String getName();
    static aspect Impl {
        private String Nameable.name;
        public void Nameable.setName(String name) {
            this.name = name;
        }
        public String Nameable.getName() {
            return this.name;
        }
    }
}
```

Deklarácie dedenia a implementácie

```
public aspect EntityBeanParticipationAspect {
    declare parents: @Entity * implements BeanSupport;
}
```

Deklarácie anotácií

```
declare @type: @PrivacyControlled * : @Tracked;
```

Varovania a chyby

- **declare error** : «bodový prierez» : «správa»;
- **declare warning** : «bodový prierez» : «správa»;
- Nepovolené bodové prierezy: *cflow*, *cflowbelow*, *this*, *target*, *args*, *if*

Zmäkčenie výnimiek

```
declare soft : RemoteException :
                        call(void TestSoftening.perform());
```

Aspektové knižnice

Aspektové knižnice

- Parametrizované bodové prierezy
  - abstraktné bodové prierezy → definované pri dedení
- Parametrizované spávanie
  - abstraktné metódy
- Bodové prierezy na základe anotácií

Abstraktný aspekt

```
public abstract aspect AbstractTracing {

    public abstract pointcut traced();

    public abstract Logger getLogger();

    before() : traced() {
        getLogger().log(Level.INFO, "Before: " + thisJoinPoint);
    }

}
```

Dediaci aspekt

```
public aspect BankingTracing extends AbstractTracing {

    public pointcut traced() : execution(* banking..*.*(..));

    public Logger getLogger() {
        return Logger.getLogger("banking");
    }

}
```

Inštancie aspektov

Inštancie aspektov

- Vytvárajú sa automaticky systémom
  - bezparametrický konštruktor
- Štandardne je aspekt singletonom
  - `aspect «NázovAspektu» issingleton() { … }`

Asociácie aspoktov

- s objektom
  - **perthis**(«bodový prierez»)
  - **pertarget**(«bodový prierez»)
- s tokom riadenia
  - **percflow**(«bodový prierez»)
  - **percflowbelow**(«bodový prierez»)
- s typom
  - **pertypewithin**(«vzor typov»)

Príklad

```
public abstract aspect TracingAspect
                                    pertypewithin(ajia.services.*) {
    private Logger logger;

    abstract pointcut traced();

    after() : staticinitialization(*) {
        logger = Logger.getLogger(getWithinTypeName());
    }

    before() : traced() {
        logger.log(...);
    }
}
```

Získanie inštancie

- Statické metódy aspektu
  - `aspectOf()`
  - `hasAspect()`
  - argument podľa typu asociácie
- Metódy triedy *Aspects*

Privilegované aspekty

- **`privileged`** `public aspect PrivilegeTestAspect { … }`
- Prístup k privátnym metódam a premenným

Spring AOP

Spring AOP

- Alternatíva AspectJ
- @AspectJ syntax
  - alebo XML konfigurácia
- Implementácia pomocou dynamických proxy
  - alebo *cglib* proxy
- Aspect Oriented Programming with Spring

Obmedzenia Spring AOP

- Z bodov spájania iba **execution**
- Nie všetky typy bodových prierezov
- Iba pre beany manažované Springom
- Nerieši volania metód v rámci triedy

Konfigurácia

- `@EnableAspectJAutoProxy`
- `<aop:aspectj-autoproxy/>`

Spring + AspectJ

- Dependency injection pre objekty vytvárane mimo Springu
  - Označiť triedy `@Configurable`
  - `@EnableSpringConfigured`
  - `<context:spring-configured/>`
- Pretkávanie pri načítaní tried

Článok

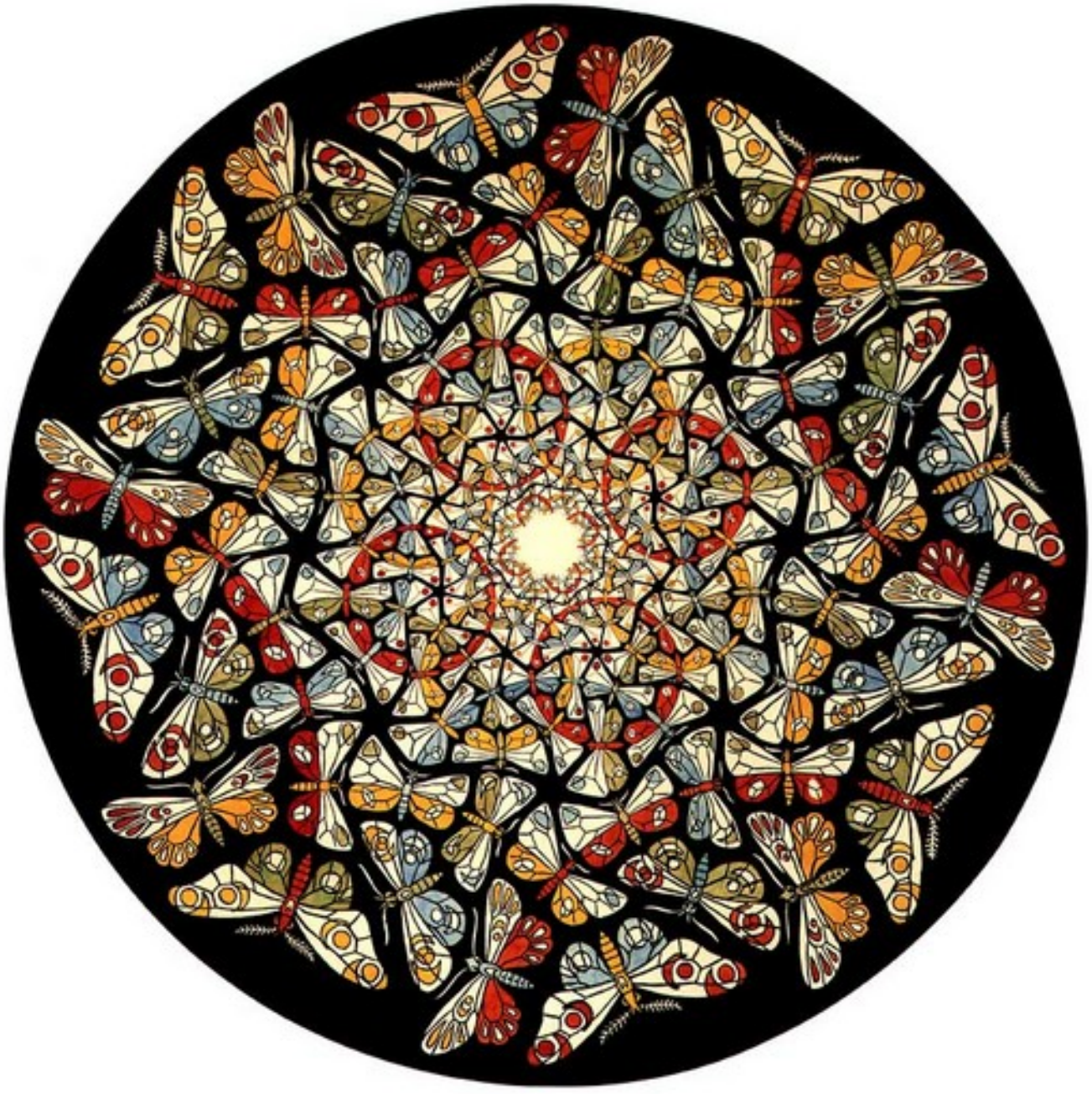
K. Ostermann, P. Giarrusso, C. Kästner, and T. Rendel, “**Revisiting Information Hiding: Reflections on Classical and Nonclassical Modularity**,” in ECOOP 2011, doi: 10.1007/978-3-642-22655-7_8

Ďalší zaujímavý článok

F. Steimann, “**The Paradoxical Success of Aspect-oriented Programming**,” OOPSLA '06, doi: 10.1145/1167473.1167514

Záver

- Metaprogramovanie — silný nástroj a nebezpečná zbraň
- Paradoxálny labyrint odporujúci intuícii
- Princíp najmenšieho prekvapenia
- Napríklad, anotácie označujúce miesta, kde sa používa metaprogramovanie



M.C. Escher, Circle Limit with Butterflies (1950)