Closures: Modernizando a linguagem Java

Michael Nascimento Santos

Rodrigo V. M. Kumpera

JustJava 2007

Michael Nascimento Santos

- 8 anos de experiência com Java
- Co-líder da JSR-310 (Date and Time API)
- Expert nas JSRs 207, 250, 270 (Java 6), 296 (Swing Application Framework), 303 (Bean Validation)
- Co-fundador do SouJava
- Fundador do genesis (https://genesis.dev.java.net) e do ThinNB (https://thinnb.dev.java.net)
- Palestrante no JavaOne, JustJava, Abaporu, FISL, COMDEX, BrasilOne e Conexão Java

Rodrigo V. M. Kumpera

- Engenheiro do Mono (http://www.mono-project.com/)
- Gosta de estudar sobre linguagens de programação
- E implementar o runtime delas das 8 às 17h

Agenda

- Definição
- Exemplos em outras linguagens
- Soluções existentes em Java
- Propostas de closures
- Comparação das propostas
- Possibilidades de inclusão na linguagem
- Q & A

O que são closures?

- Uma **closure** é uma **função** que se refere a variáveis livres no seu contexto léxico
- Uma **função** é um bloco de código com um ou mais parâmetros que pode ou não retornar um valor
- Uma **variável livre** é um identificador usado mas não definido pela closure
- ... ????

Exemplos: JavaScript

```
function filtra(numeros, referencia) {
   return numeros.filter(
      function (numero) {
         return numero >= referencia;
      }
   );
}
```

Exemplos: Clipper

```
LOCAL ARQUIVOS : = DIRECTORY (“*.*”)
LOCAL NOMES : = {}
CLEAR
AEVAL (ARQUIVOS, {  | INFO | AADD
   (NOMES, INFO[1]  ) } )
ESCOLHA :=  ACHOICE ( 10, 10, 20, 35,
   NOMES)
```

Problemas nas soluções atuais

- Implementação de for each

```
interface OneArg<A> {
  void invoke(A arg);
}

<T> void forEach(Iterable<T> i, OneArg<T> block) {
  for (Iterator<T> it = i.iterator(); c.hasNext(); ) {
  block.invoke(it.next());
}
```

Problemas nas soluções atuais

- Ideal

```
for (String s : strings) doThing(s);
```

- Real

```
forEach(strings, new OneArg<String>() {
  public void invoke(String s) {
    doThing(s);
  }
});
```

Problemas nas soluções atuais

- Ideal

```
String toString(Thing t) { ... }
for (Thing t : things) println(toString(t));
```

- Real

```
forEach(things, new OneArg<Thing>() {
  public void invoke(Thing t) {
    println(toString(t)); // erro!
  }
});
```

Problemas nas soluções atuais

- Ideal

```
for (String s : strings)
  if (... ) throw new MyException();
```

- Real

```
forEach(strings, new OneArg<String>() {
  public void invoke(String s) {
    if (... ) throw new MyException(); // erro!
  }
});
```

Problemas nas soluções atuais

- Ideal

```
for (String s : strings)
    if (... ) return computeResult(s);
```

- Real

```
forEach(strings, new OneArg<String>() {
   public void invoke(String s) {
      if (... ) return computeResult(s); // erro!
   }
});
```

Problemas nas soluções atuais

- Ideal

```
String found = null;
for (String s : strings) if (...) found = s;
```

- Real

```
String found = null;
forEach(strings, new OneArg<String>() {
  public void invoke(String s) {
    if (...) found = s; // erro!
  }
});
```

Problemas nas soluções atuais

- Verbosas, confusas, feias, redundantes etc.
- Captura incompleta do contexto léxico
  - ✓ ***this, overloading***
  - ✓ ***return, break, continue***
  - ✓ ***variáveis locais não final***
- Não podem ser usadas para substituir um bloco arbitrário de código

Propostas de closures

- **Real**

```
public static <T> T withLock (Lock lock, Callable<T> block)
    throws Exception {
    lock.lock();
    try {
        return block.call();
    } finally {
        lock.unlock();
    }
}
withLock(lock, new Callable<String>() {
    public String call() { return “hello”};
});
```

Propostas de closures

- **BGGA (Bracha, Gafter, Gosling, Ahé)**

```
public static <T>T withLock(Lock lock, {=>T} block) {
    lock.lock();
    try {
        return block.invoke();
    } finally {
        lock.unlock();
    }
}
withLock(lock, {=> return "hello"});
withLock(lock) {
   return "hello"
}
```

Propostas de closures

- **FCM (First-Class Methods) + JCA**

```
public static <T>T withLock(Lock lock, #(T()) block) {
    lock.lock();
    try {
        return block.invoke();
    } finally {
        lock.unlock();
    }
}
withLock(lock, #{return "hello";});
withLock(lock) {
   return "hello";
}
```

Propostas de closures

- **CICE (Concise Instance Creation Expressions)**

```
public static <T>T withLock(Lock lock, Callable<T> block) {
    lock.lock();
    try {
        return block.invoke();
    } finally {
        lock.unlock();
    }
}
withLock(lock, Callable<String>() {return "hello";});
protected(lock) { // construcao especial da extensao ARM
   return "hello";
}
```

Comparação das propostas

- **Todas permitem:**
  - ✓ ***Leitura de variáveis locais***
  - ✓ ***Escrita de variáveis locais (CICE requer public)***
  - ✓ ***Conversão de closures para classes anônimas***

Comparação das propostas

- **CICE - Concise Instance Creation Expressions (Bob Lee, Doug Lea, Joshua Bloch) ARM – Automatic Resource Management**
  - ✓ ***Simplifica criação de classes anônimas e acesso a variáveis locais e ignora os outros problemas (this, overloading, return, break)***
- **BGGA (Bracha, Gafter, Gosling, Ahé)**
  - ✓ ***Permite que o significado dos bloco de código não seja alterado, permite criar abstrações de controle, mas cria um return-na-última-linha-sem-ponto-e-vírgula***

Comparação das propostas

- **FCM - First Class Methods (Stephen Colebourne,Stefan Schulz) – e JCA – Java Control Abstraction (+ Ricky Clarkson)**
  - ✓ ***Foca em permitir closures sem romper com a sintaxe da linguagem, mas permite mudanças de semântica em casos sutis entre um bloco dentro de um método e dentro de uma closure***

Possibilidades de inclusão na linguagem

- **Tentou-se propor uma JSR; falhou:**
  - ✓ ***Divergências no Google (BGGA x CICE)***
  - ✓ ***Exclusão dos líderes da FCM do EG***
- **Novo caminho: protótipo**
  - ✓ ***Apenas Neal Gafter (BGGA) está trabalhando num protótipo no momento***
  - ✓ ***Protótipo desenvolvido sob a JRL***
  - ✓ ***Se o OpenTCK (GPL) for aplicado sobre o protótipo, não poderá ser integrado ao JDK por problemas de licença***

Status do protótipo por Neal Gafter

- **Plan to work with Doug Lea on**
  - ✓ ***Writing sample clients for jsr166y to exploit closures***
  - ✓ ***Writing a new API for jsr166y to exploit function types***
- **Things that work now:**
  - ✓ ***Closures capturing variables that aren't reassigned after declaration.***
  - ✓ ***Function types, auto-generated.***
  - ✓ ***The 'closure conversion'.***

Status do protótipo por Neal Gafter

- **Things that work now:**
  - ✓ ***The 'null type' in some cases.***
  - ✓ ***The "Unreachable" type (not for primitives)***
- **In development:**
  - ✓ ***Type inference for the concrete type of a closure expression (depends on context)***
  - ✓ ***Exception transparency***
- **Not done yet:**
  - ✓ ***Capturing mutated variables, non-local control flow and control invocation statement***

Status do protótipo por Neal Gafter

- Disponível na época do Javapolis (Dezembro / 2007)
- Liberação do código fonte dependente de resolução da questão legal criada pela Sun
- Se a JSR para o Java 7 for criada depois disso, há chances de inclusão

Perguntas?

Obrigado!

http://blog.michaelnascimento.com.br/
http://www.kumpera.net/

Michael Nascimento - michael@summa-tech.com

Rodrigo V. M. Kumpera – kumpera@gmail.com

JustJava 2007