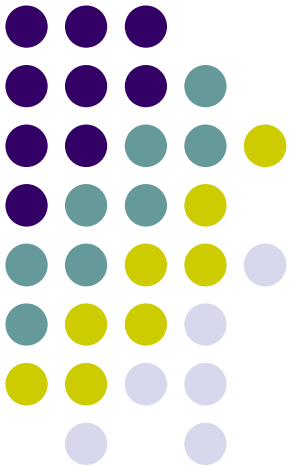
Computação Paralela

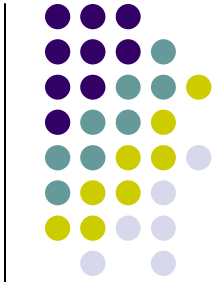
Uma *framework* para aplicações concorrentes

João Luís Ferreira Sobral
Departamento do Informática
Universidade do Minho

Dezembro 2005



ParAspJ – Aplicações Paralelas em Java

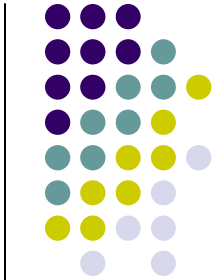


Fases de desenvolvimento de aplicações paralelas

1. Desenvolvimento da aplicação sequencial (cliente – servidor)
2. Acrescentar código de partição do trabalho por vários servidores (cliente - vários servidores)
3. Acrescentar concorrência através da invocação assíncrona de métodos (cliente – Threads – servidor)
4. Distribuir os servidores por várias máquinas

- O ambiente ParAspJ gera automaticamente todo o código necessário para a distribuição dos objectos e distribui os objectos da aplicação pelos nodos de processamento disponíveis
- O código correspondente a cada fase de desenvolvimento é encapsulado num único módulo através da utilização de AOP (programação com aspectos)

ParAspJ – Aplicações Paralelas em Java



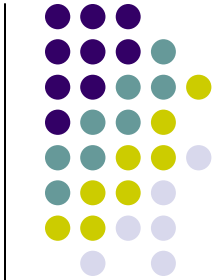
Fases de desenvolvimento para o cálculo de PI

2. Acrescentar código de partição do trabalho por vários servidores (cliente - vários servidores)

```
double n=10000000;
System.out.println("Calcular pi com " + n
double h = 1.0 / n;
CalcPi pc[] = new CalcPi[4];
for (int p=0; p<4; p++)
    pc[p] = new CalcPi();

double aprox[] = new double[4];
for (int p=0; p<4; p++)
    aprox[p] = pc[p].calc(p/4,(p+1)/4,h);

double taprox=0;
for (int p=0; p<4; p++) {
            taprox += aprox[p];
}
```

```
Vector pc = new Vector();

CalcPi around() : call (CalcPi.new()) && within(Client*) {
  for(int p=0; p<4; p++)
        pc.add( new CalcPi() );
  return( (CalcPi) pc.get(0) );
}

double around(…,double h) : call(* *.calc(int,int,double))  {
  Vector apr = new Vector();
  for(int p=0; p<4; p++) {
      CalcPi pcp = (CalcPi) pc.elementAt(p);
      apr.add( pcp.PCalc(p/4,(p+1)4,h) );
  }
  double taprox = 0;
  for(int p=0; p<4; p++)
    taprox += ((MyDouble) apr.elementAt(p)).doubleValue();
  return(taprox);
}
```

ParAspJ – Aplicações Paralelas em Java



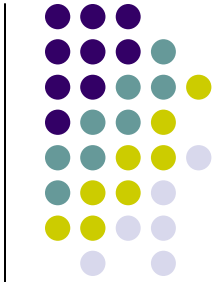
Fases de desenvolvimento para o cálculo de PI

3. Acrescentar concorrência através da invoca

```
…
CalcPi pc[] = new CalcPi[4];
double aprox[][] = new double[4][1];
PiThread pit[] = new PiThread[4];
for (int p=0; p<4; p++) {
        pc[p] = new CalcPi();
        pit[p] = new  PiThread(pc[p],p/4,(
        pit[p].start();
 }
double taprox=0;
for (int p=0; p<4; p++) {
        try {
          pit[p].join();
        } catch(Exception ex) {}
        taprox = aprox[p][0];
}
…
```

```
HashMap results = new HashMap();
HashMap threads = new HashMap();

MyDouble around() : call(* *.PCalc(..)) && within(Partition)
{
   final MyDouble r = new MyDouble(0);
   Thread t = new Thread() {
      public void run() {       results.put(r,proceed());       }
   };
   t.start();
   threads.put(r,t);
   return(r);
}
Object around() : call(* Vector.elementAt(..) ) {
   Object dt = proceed();
   if (threads.get(dt)!= null) {  // resultado pendente
       try {
          ((Thread) threads.get(dt)).join();
        } catch (Exception e) { }
       dt = results.get(dt);
    }
   return(dt);
}
```

ParAspJ – Aplicações Paralelas em Java



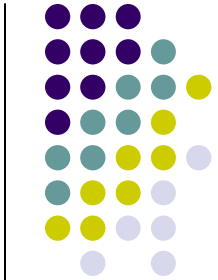
Fases de desenvolvimento para o cálculo de PI

4. Distribuir os servidores por várias máquinas

```
public interface ICalcPi extends Remote {
        public double calc(int min, int max, double passo) throws RemoteException;
}

public class PiThread extends Thread {
        ICalcPi tpic;
        …
        public PiThread(ICalcPi pic, i
                tpic = pic; tmin=
        }
        public void run() {
            try {
                tres[0] = tpic.calc
             } catch(Exception ex) {}
        }
}
```

```
MyDouble around(CalcPi obj,
        double min, double max, double passo) :
        call(* *.PCalc(..)) && args(min,max,passo) && … {
    MyDouble res = null;
    try{
       res = ((ICalcPi) remotes.get(obj)).PCalc(min,max,passo);
    }catch (Exception e){e.printStackTrace();}
    return(res);
}
```

ParAspJ – Aplicações Paralelas em Java



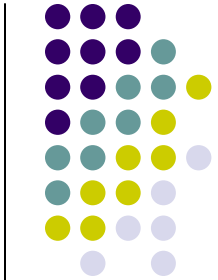
Fases de desenvolvimento para o cálculo de PI

4. Distribuir os servidores por várias máquinas

```
...
ICalcPi pc[] = new ICalcPi[4];
double aprox[][] = new double[4][1];
PiThread pit[] = new PiThread[4];
for (int p=0; p<4; p++) {
    try {   // pc[p] = new CalcPi();
        Context ctx = new InitialContext();
        String nome=new String("PS"+(p+1))
        System.out.println("Localizar " + nor
        pc[p] = (ICalcPi)
            PortableRemoteObject.narrow(ct
            ICalcPi.class);
     } catch(Exception ex) {}
     pit[p] = new  PiThread(pc[p],p/4,(p+1)/4
     pit[p].start();
 }
...
```

```
int conta=0;
HashMap remotes = new HashMap();

Object around() : call (CalcPi.new(..)) && …) {
    ICalcPi remote=null;
    try {
        Context ctx = new InitialContext();
        String name=new String("PS"+(++conta));
        System.out.println("Localizar " + name);
        remote = (ICalcPi)
            PortableRemoteObject.narrow(ctx.lookup(name),
            ICalcPi.class);
    } catch(Exception ex) {  }

    CalcPi local = new CalcPi();
    remotes.put(local,remote);
    return(local);
}
```

ParAspJ – Aplicações Paralelas em Java



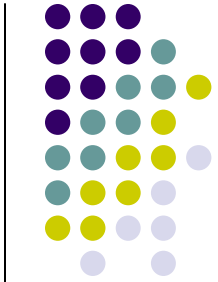
Fases de desenvolvimento para o cálculo de PI

4. Distribuir os servidores por várias máquinas

```
public class CalcPi  extends RemoteObject implements ICalcPi  {
                …
    public double calc(int min, int max, double pass) {
        ….
    }
    public static void main(String args[]) {  // regista
        try {
            CalcPi pic = new CalcPi();
            PortableRemoteObject.exportObject(pic);
            Context ctx = new InitialContext();
            ctx.rebind(args[0],pic);
            …
        } catch(Exception e) { e.printStackTrace(); }
    }
}
```

```
declare parents: CalcPi implements ICalcPi;
declare parents: CalcPi extends RemoteObject;

public static void CalcPi.main(String args[]) {
    try {
        CalcPi pf = new CalcPi();
        PortableRemoteObject.exportObject(pf);
        Context ctx = new InitialContext();
        ctx.rebind(args[0],pf);
        …
    } catch(Exception e) { e.printStackTrace(); }
}
```

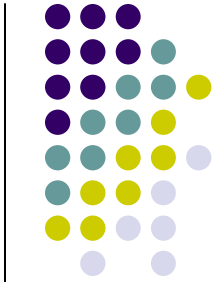
ParAspJ – Aplicações Paralelas em Java



Algumas Combinações possíveis

	Sequencial	Partição	Concorrência	Distribuição
Aplicação sequencial com vários servidores	**Sim**	**Sim**	**Não**	**Não**
Aplicação concorrente com vários servidores	**Sim**	**Sim**	**Sim**	**Não**
Aplicação paralela com vários servidores	**Sim**	**Sim**	**Sim**	**Sim**
Aplicação sequencial distribuída com vários servidores	**Sim**	**Sim**	**Não**	**Sim**
Aplicação sequencial distribuída	**Sim**	**Não**	**Não**	**Sim**

Framework para programação concorrente

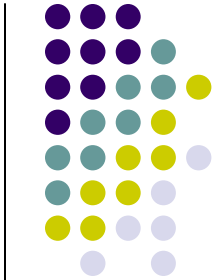


Anotações em Java 5

- Em Java 5 é possível anotar classes, métodos e campos.

```
@ClassAnnotation
public class <class_name> {
        @FieldAnnotation int x;

        //...
        @MethodAnnotation
        public void execute(){}
}
```

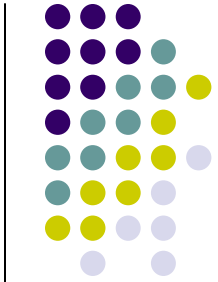
Framework para programação concorrente



Anotações suportadas

- @Oneway(threadGroup=[thread group id]), @Join
- @Future, @FutureClient
- @ActiveObject
- @BarrierAfterExecution (nThreads = 5, threadGroup = "calculus")
- @Synchronized(id = "lockName")
- @Reader, @Writer
- @Scheduled(order=Order.[FIFO | LIFO | ordered| master | single], threadGroup=[thread-group name])
- @ThreadLocal(deepCopy=[yes|no], reduce=[operationId])

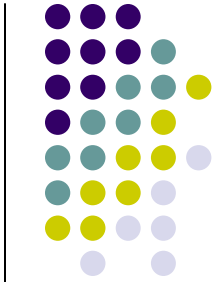
Framework para programação concorrente



Implementação (OneWay)

```
public abstract aspect OnewayProtocol {
   protected pointcut onewayAnnotation(): execution(@Oneway * *.*(..));

   void around(): onewayAnnotation(){
         Thread t = new Thread( new Runnable(){
                  public void run(){
                        proceed();
                  }
            }
         );
         registerThread(t); // save to join later
         t.start();
   }
   ...// other advices and auxiliary methods
}
```

Framework para programação concorrente

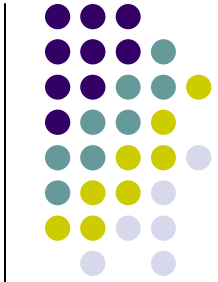


Exemplos 1 (RayTracer)

Annotations	Explicit pointcut

`public class RayTracer {`
`  @OneWay(threadGroup="T1")`
`  public void renderLine(/* … */) {`
`  ...`
`  }`
`  @JoinAfterExecution`
`  public void render(Interval in) {`
`  ...`
`  for(int y=0; y<in.height; y++)`
`  renderLine(/* … */);`
`  ...`
`  }`
`}`
	`public class RayTracer {`
	`  ... // class without annotations`
	`}`
	`aspect OneWay`
	`  extends OnewayProtocol {`
	`  pointcut onewayMethodExecution():`
	`  call(void *.renderLine(..));`
	`  pointcut join():`
	`  call(void * render(..));`
	`  String getThreadGroupName() {`
	`  return("T1"); }`
	`}`

Framework para programação concorrente



Exemplos 2 (Fibonacci)

Annotations	Explicit pointcut

`public class Fib {`
`  long value;`
`@Future`
`@FutureClient`
`public long compute() {`
`  if (value <=1) return(value);`
`  else{`
`  Fib f1 = new Fib(value-1);`
`  Fib f2 = new Fib(value-2);`
`  Long r1 = f1.compute();`
`  Long r2 = f2.compute();`
`  return(r1+r2); // longValue`
`  }`
`}`
`}`
	`public class Fib {`
	`  ... // class without annotations`
	`}`
	`aspect Future`
	`  extends FutureProtocol {`
	`pointcut futureMethodExecution():`
	`  call(Long *.compute()); &&`
	`  if (((Fib) servant).value)>8 &&`
	`  target(servant);`
	`pointcut useOfFuture():`
	`  call(* Long.longValue() );`
	`Long getFakeObject() {`
	`  return(new Long()); }`
	`}`