Seminář Java
IX
2005/2006

Radek Kočí

Obsah

- Vlákna
- Multithreading
- Sdílení prostředků
- Synchronizace

Pojmy

Proces

- spuštěný program s vlastním adresovým prostorem

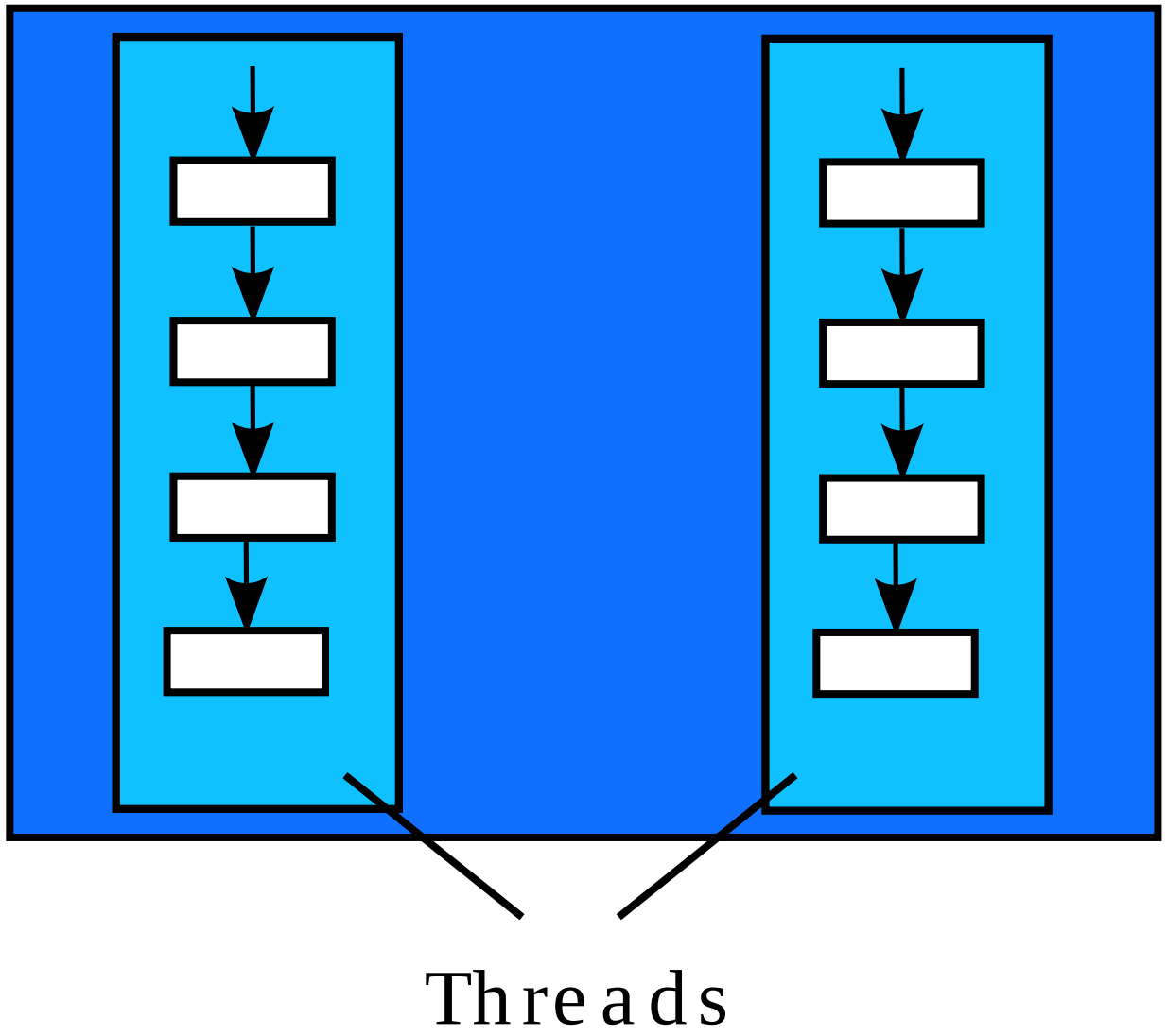
Vlákno (podproces)

- nezávislá vedlejší úloha
- spuštěná v kontextu procesu
- sdílí adresový prostor procesu

Multithreading

- technika rozdělění běhu programu na více podprocesů
- typicky pro oddělení částí programu, které jsou vázané na prostředky

Proces a vlákna



Práce s vlákny

Balík `java.lang`

- `Thread`
- `Runnable`

Vytvoření objektu, který reprezentuje vlákno

- rozšířením (dědičnost) třídy `Thread`
- implementací rozhraní `Runnable`

Třída `Thread`

Metody

- `start()`
  - inicializace vlákna
  - volá metodu `run()`
  - odvozená třída *nepřekrývá* tuto metodu!
- `run()`
  - kód vlákna
  - odvozená třída *překrývá* tuto metodu
- `sleep(long millis)`
  - uspání vlákna na daný počet milisekund
  - výjimka *InterruptedException*

Ukázka SimpleThread

```
public class SimpleThread extends Thread {
    public SimpleThread(String str) {
        super(str);
    }
    public void run() {
        for (int i = 0; i < 10; i++) {
            System.out.println(i + " " + getName());
            try {
                sleep((long)(Math.random() * 1000));
            } catch (InterruptedException e) {}
        }
        System.out.println("DONE! " + getName());
    }
}
```

Ukázka `SimpleThread`

```
public class TwoThreadsDemo {
    public static void main (String[] args) {
        new SimpleThread("Jamaica").start();
        new SimpleThread("Fiji").start();
    }
}
```

Skupina vláken

`java.lang.ThreadGroup`

- vlákno patří vždy k nějaké skupině
- existuje implicitní systémová skupina
- skupiny tvoří stromovou hierarchii
- příslušnost ke skupině je neměnná
- vlákno implicitně "dědí" skupinu vytvářejícího vlákna
- `Thread.currentThread().getThreadGroup()`

Rozhraní `Runnable`

Metody

- `run()`
  - kód vlákna
  - implementující třída musí implementovat tuto metodu

Implementující třída

- není vlákno
- informace, že instance třídy definuje chování vlákna
- pro spuštění vlákna potřebuje třídu `Thread`

Konstruktory třídy `Thread`

`Thread(ThreadGroup group, Runnable target, String name)`

- alokuje nový `Thread` object
- pojmenuje vlákno podle `name`
- `if target != null` – objekt, jehož metoda `run()` je volána
- `group` – skupina vláken, do kterého je vlákno zařazeno

`Thread()`

- jako `Thread(null, null, gname)`
- `gname` – automaticky generované jméno vlákna

`Thread(String name)`

- jako `Thread(null, null, name)`

`Thread(Runnable target)`

- jako `Thread(null, target, name)`

Ukázka `Clock`

```
import java.awt.Graphics;
import java.util.*;
import java.text.DateFormat;
import java.applet.Applet;

public class Clock extends Applet implements Runnable {
    private Thread clockThread = null;
    public void start() {
        if (clockThread == null) {
            clockThread = new Thread(this, "Clock");
            clockThread.start();
        }
    }
    public void paint(Graphics g) {
        //get the time and convert it to a date
        ...
        g.drawString(dateFormatter.format(date), 5, 10);
    }
    ...
```

Ukázka `Clock`

```
    ...
    public void run() {
        Thread myThread = Thread.currentThread();
        while (clockThread == myThread) {
            repaint();
            try {
                Thread.sleep(1000);
            } catch (InterruptedException e) {
                //the VM doesn't want us to sleep anymore,
                //so get back to work
            }
        }
    }
    //overrides Applet's stop method, not Thread's
    public void stop() {
        clockThread = null;
    }
}
```

Třída `Thread` *nebo rozhraní* `Runnable`*?*

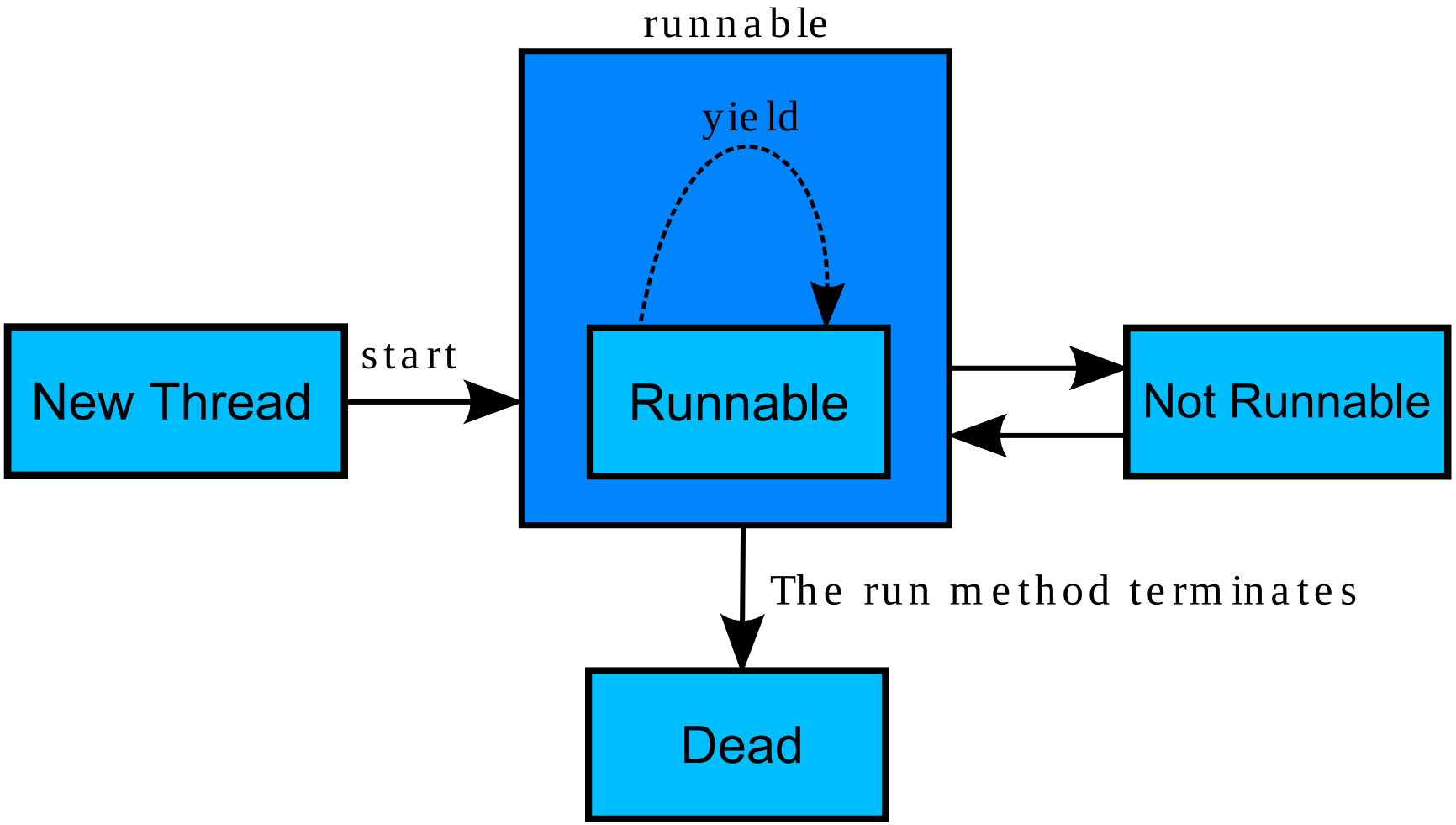
- hodně tříd *potřebuje* dědit (rozšiřovat) jinou třídu

```
public class Clock extends Applet implements Runnable {
    ...
}
```

- Java neumožňuje vícenásobnou dědičnost
- $\Rightarrow$ rozhraní `Runnable`

```
clockThread = new Thread(this, "Clock");
clockThread.start();
```

Životní cyklus vlákna



Životní cyklus vlákna

Vytvoření

- `new Thread(this, "Clock")`

Spuštění

- `start()`
- kód vlákna v metodě `run()` (`Thread` nebo `Runnable`)

Ukončení

- přirozeným doběhnutím metody `run()`
- *existují i jiné metody, ty se ale nedoporučují (deprecated) ...*
  - *`stop()`*
  - *`destroy()`*
  - *`suspend()`, `resume()`*

Životní cyklus vlákna

Test stavu vlákna

- `isAlive()`
- $\Rightarrow$ `true` pokud bylo vlákno spuštěno a nebylo ukončeno (není dead)
- $\Rightarrow$ `false` pokud vlákno nebylo spuštěno, nebo bylo ukončeno (je dead)

Životní cyklus vlákna

Pozastavení (stav *not runnable*)

- `sleep(...)`
- `wait()`
- při i/o operaci

Uvolnění (stav *runnable*)

- uplynutí doby čekání (viz `sleep(...)`)
- `notify()`, `notifyAll()`
- dokončení při i/o operace

Plánování vláken

Jedna CPU

- provádění vláken se musí plánovat
- plánovač v Javě je *fixed-priority scheduling*
- plánuje vlákna na základě jejich priority relativně k ostatním vláknům

Priorita

- vlákno *dědí* prioritu vlákna, ve kterém bylo vytvořeno
- čtení/změna priority: `getPriority()`, `setPriority()`
- rozsah: `Thread.MIN_PRIORITY` – `Thread.MAX_PRIORITY`

Plánování vláken

Plánovač

- vybere vlákno (*runnable*) s nejvyšší prioritou
- pokud je jich více se stejnou prioritou, vybere náhodně

Vlákno běží dokud se nestane:

- na systému s *time-slicing* uběhne přidělené časové kvantum
- jiné vlákno s vyšší prioritou přejde do stavu *runnable* (it preemts the current thread)
- skončí metoda `run()`
- vlákno se vzdá procesoru
- vlákno se dobrovolně vzdá procesoru – zpráva `yield()` $\Rightarrow$ šance pro ostatní vlákna na *stejné* prioritě

Synchronizace vláken

Problém producent–konzument

- jedno vlákno (producent) zapisuje na sdílené místo data
- druhé vlákno (konzument) tato data čte
- operace zápis/čtení se musí střídat!

Ukázkový příklad

- třída `Producer` – producent
- třída `Consumer` – konzument
- třída `CubbyHole` – sdílený prostor (metody get a put)

Producent–konzument: Producent

```
public class Producer extends Thread {
    private CubbyHole cubbyhole;
    private int number;

    public Producer(CubbyHole c, int number) {
        cubbyhole = c;
        this.number = number;
    }

    public void run() {
        for (int i = 0; i < 10; i++) {
            cubbyhole.put(number, i);
            try {
                sleep((int)(Math.random() * 100));
            } catch (InterruptedException e) { }
        }
    }
}
```

Producent–konzument: Konzument

```
public class Consumer extends Thread {
    private CubbyHole cubbyhole;
    private int number;

    public Consumer(CubbyHole c, int number) {
        cubbyhole = c;
        this.number = number;
    }

    public void run() {
        int value = 0;
        for (int i = 0; i < 10; i++) {
            value = cubbyhole.get(number);
        }
    }
}
```

Producent–konzument: možné problémy

Producent je rychlejší

- konzument může "propásnout" čísla

Konzumer je rychlejší

- konzument čte stejné číslo vícekrát

$\Rightarrow$ race condition

- dvě (příp. více) vlákna čtou/zapisují sdílená data; výsledek zavisí na časování jak jsou vlákna plánována

Nutná synchronizace:

- vlákna nesmí současně přistoupit ke sdílenému objektu
- producent musí indikovat, že hodnota je připravena; nezapisuje dokud si hodnotu nepřečte konzument
- konzument musí indikovat, že přečetl hodnotu; nečte dokud producent nezapíše novou hodnotu

Monitor

Monitor

- kritické sekce
- uzamčení objektu při přístupu ke kritické sekci
- pokud je objekt uzamčen, nikdo jiný nemůže přistupovat je kritickým sekcím objektu
- odemknutí objektu při výstupu z kritické sekce

Monitor v Javě

- součástí každého objektu (třída `Object`)
- klíčové slovo `synchronized`

Kritická sekce v Javě

- metoda
- blok

Producent–konzument: CubbyHole

```
public class CubbyHole {
    private int contents;
    private boolean available = false;

    public synchronized int get() {
        //CubbyHole locked by the Producer
        ...
        // CubbyHole unlocked by the Producer
    }

    public synchronized int put(int value) {
        // CubbyHole locked by the Consumer
        ...
        // CubbyHole unlocked by the Consumer
    }
}
```

Zámek: reentrantní

Zámek monitoru je reentrantní

- umožňuje vnořené volání synchronizovaných metod
- opakované vstupování stejného vlákna do kritických sekcí

```
public class Reentrant {
    public synchronized void a() {
        b();
        System.out.println("here I am, in a()");
    }

    public synchronized void b() {
        System.out.println("here I am, in b()");
    }
}
```

Producent–konzument: CubbyHole

(chybná implementace – chybí synchronizace vláken)

```
public class CubbyHole {
    private int contents;
    private boolean available = false;

    public synchronized int get() {
        if (available == true) {
            available = false;
            return contents;
        }
    }
    public synchronized int put(int value) {
        if (available == false) {
            available = true;
            contents = value;
        }
    }
}
```

Producent–konzument: CubbyHole

```
public synchronized int get() {
    while (available == false) {
        try {
            //wait for Producer to put value
            wait();
        } catch (InterruptedException e) { }
    }
    available = false;

    //notify Producer that value has been retrieved
    notifyAll();

    return contents;
}
```

Producent–konzument: CubbyHole

```
public synchronized void put(int value) {
    while (available == true) {
        try {
            //wait for Consumer to get value
            wait();
        } catch (InterruptedException e) { }
    }
    contents = value;
    available = true;

    //notify Consumer that value has been set
    notifyAll();
}
```

Synchronizace vláken (třída `Object`)

`wait()`

- aktuální vlákno bude čekat, dokud se nezavolá `notify()` (`notifyAll()`) nad objektem

`wait(long timeout)`

- `...`nebo neuplyne timeout

`notify()`

- vzbudí jedno vlákno čekající na monitoru objektu

`notifyAll()`

- vzbudí všechna vlákna čekající na monitoru objektu

Synchronizace vláken (třída `Object`)

`wait(), ...`

- před suspendováním vlákna se odemkne monitor
- pokud vlákno vlastní více monitorů, odemkne se pouze monitor daného objektu

`notify(), ...`

- řízení není okamžitě předáno vzbuzenému vláknu

Tyto metody může volat pouze to vlákno, které je vlastníkem monitoru

- vstoupení do kritické sekce (`synchronize`) – metoda, blok

Synchronizace vláken – efektivita

Při uzamčení objektu se zvýší náklady na režii

- uvážit změnu návrhu
- použít zámek (synchronized) na konkrétní objekt
- neodbýt souběžný přístup synchronizací všech metod

Synchronizace vláken

Blok jako kritická sekce

- stejný princip synchronizace
- metody se nemusí deklarovat jako synchronizované
- deklaruje se objekt, jehož monitor se použije při obsluze kritické sekce

```
synchronized(cubbyhole) {
    cubbyhole.put(i);
}
```

```
synchronized(cubbyhole) {
    value = cubbyhole.get();
}
```

Proč nepoužívat `stop` *a* `suspend`*?*

`stop()`

- uvolní všechny monitory blokované vláknem
- nebezpečí přístupu k objektům v nekonzistentním stavu
- $\Rightarrow$ přirozené ukončení metody `run()`

`suspend()`, *`resume()`*

- suspenduje/uvolní vlákno
- suspendované vlákno drží monitor (viz kritická sekce); vlákno, které ho má uvolnit (viz resume), musí vstoupit do této sekce $\Rightarrow$ dead-lock
- $\Rightarrow$ `wait()`, `notify()`

---

více na
`http://java.sun.com/j2se/1.5.0/docs/guide/misc/threadPrimitiveDeprecation.html`

Explicitní zámky

`java.util.concurrent.locks`

- rozhraní `Lock`
- implementace `ReentrantLock`
- rozhraní `Condition`
- flexibilnější synchronizační prostředek než monitory objektů

```
private Lock aLock = new ReentrantLock();
private Condition condVar = aLock.newCondition();

aLock.lock();
condVar.await();
condVar.signalAll();
condVar.awaitInterruptibly() // nelze přerušit

...
```

Synchronizované struktury

`java.util.concurrent`

- `BlockingQueue`
- `SynchronousQueue`
- `Semaphore`
- `...`

```
private BlockingQueue cubbyhole;
cubbyhole.put(i);

...
```

Zdroje informací

http://java.sun.com/docs/books/tutorial/essential/

http://www.programming-x.com/programming/brian-goetz.html

http://www.javaworld.com