Chương 4

Lập trình đa tuyến đoạn

1. Tổng quan

Khi thực hiện một công việc phức tạp người ta thường chia công việc ra thành nhiều phần và giao công việc cho nhiều người cùng thực hiện, điều này giúp cho công việc được tiến hành nhanh chóng. Các ứng dụng phần mềm sử dụng một chiến lược tương tự được gọi là đa tuyến đoạn để chia nhỏ các tác vụ thành các đơn vị dễ quản lý. Lập trình đa tuyến đoạn là một khái niệm quan trọng trong lập trình mạng bằng Java vì các client và server thường phải thực hiện một số tác vụ đồng thời tại cùng một thời điểm (ví dụ lắng nghe các yêu cầu và đáp ứng các yêu cầu, xử lý dữ liệu và cập nhật giao diện đồ họa người dùng). Trước khi đi vào tìm hiểu lập trình đa tuyến đoạn trong Java, ta cần hiểu rõ sự khác nhau giữa lập trình đơn tuyến đoạn, lập trình đa tiến trình và lập trình đa tuyến đoạn.

1.1. Lập trình đơn tuyến đoạn

Khái niệm đa tuyến đoạn là khái niệm khó đối với những người mới bắt đầu làm quen. Rất nhiều ngôn ngữ lập trình và hệ điều hành trước đây không hỗ trợ đa tuyến đoạn.

Phần mềm truyền thống được viết bằng các ngôn ngữ thủ tục được biên dịch thành một khuôn dạng mà máy có thể hiểu được gọi là mã máy. Bộ xử lý trung tâm đọc mã này và xử lý các lệnh theo cấu trúc tuần tự hết lệnh này đến lệnh tiếp theo. Thời gian thực hiện các lệnh có thể thay đổi tùy thuộc vào bản chất của các lệnh.

Ưu điểm chính của kiểu lập trình này là tính đơn giản của nó. Nếu một lệnh không hoàn thành thì lệnh tiếp theo sẽ không được xử lý. Điều này nghĩa là người lập trình có thể dự đoán trạng thái của máy tại bất kỳ thời điểm nào cho trước.

1.2. Lập trình đa tiến trình

Đa nhiệm là khả năng của một hệ điều hành máy tính chạy nhiều chương trình đồng thời trên một CPU. Điều này được thực hiện bằng cách chuyển hoạt động từ một chương trình này sang chương trình khác tương đối nhanh để tạo cho người sử dụng cảm giác tất cả các chương trình đang được xử lý đồng thời. Có hai kiểu đa nhiệm:

- Đa nhiệm ưu tiên. Trong đa nhiệm ưu tiên, hệ điều hành xác định cách phân bổ các thời gian của CPU cho từng chương trình. Cuối mỗi khoảng thời gian mà CPU phân bổ, chương trình hiện đang hoạt động buộc phải trả quyền điều khiển cho hệ điều hành, dù nó có muốn hay không. Các ví dụ về hệ điều hành hỗ đa nhiệm ưu tiên là Unix, Windows 95/98, Windows NT.
- Đa nhiệm hợp tác. Trong đa nhiệm hợp tác, mỗi chương trình kiểm soát một phần thời gian CPU mà nó cần. Điều này nghĩa là một chương trình phải hợp tác để trao quyền điều khiển cho các chương trình khác, nếu không nó sẽ chiếm dụng CPU. Các hệ điều hành đa nhiệm hợp tác là Windows 3.1 và Mac OS 8.5.

Những ai đã quen lập trình trên hệ thống Unix hẳn là đã quen với khái niệm lập trình đa tiến trình. Để hỗ trợ đa nhiệm, Unix sử dụng khái niệm các tiến trình. Mỗi ứng dụng đang chạy là một tiến trình, với bộ nhớ được phân bổ cho chương trình và dữ liệu. Có nhiều tiến trình chạy trên cùng một máy. Hệ điều hành sẽ phân bổ thời gian cho từng tiến trình, dừng tiến trình khi hết thời gian và cho phép tiến trình khác tiếp tục. Đôi khi, một tiến trình bị phong tỏa hoặc có thể tự chọn để giành thời gian CPU.

Lập trình đa tiến trình có các lợi ích khác. Các chương trình tự chúng có thể tạo ra các tiến trình mới , một phần chương trình thực hiện một tác vụ trong khi một phần khác thực hiện công việc khác. Ví dụ, khi đang kiểm tra email trên một máy ở xa, giao diện người dùng có thể hiển thị diễn tiến của thao tác và cho phép người dùng soạn thảo các thông điệp và đọc các thông điệp đã được tải về trước đó.

Mặc dù lập trình đa tiến trình hoạt động tốt, nhưng nó vẫn có những nhược điểm. Trước hết, khi một tiến trình phân nhánh thành hai tiến trình, sẽ dẫn đến sự chồng chéo giữa

việc lưu trữ dữ liệu của tiến trình này với tiến trình khác. Mỗi tiến trình cần có một bản sao dữ liệu của riêng nó, vì vậy nếu có nhiều tiến trình thì sẽ cần nhiều bộ nhớ. Thứ hai là không có cách nào để một tiến trình truy xuất và sửa đổi dữ liệu của một tiến trình khác.

1.3. Lập trình đa tuyến đoạn

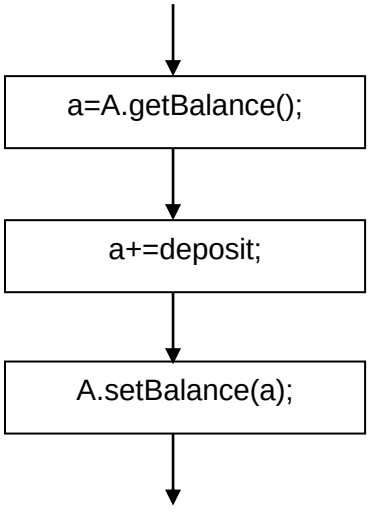
Đa tuyến đoạn mở rộng khái niệm đa nhiệm bằng cách cho phép một chương trình thực hiện một số tác vụ đồng thời. Mỗi tác vụ được xem như là một tuyến đoạn và nó có một luồng điều khiển riêng. Đa tuyến đoạn rất hữu ích trong các ứng dụng thực tế. Ví dụ, nếu quá trình nạp một trang web vào trình duyệt quá lâu, người sử dụng cần phải có khả năng ngắt việc nạp trang web đó bằng cách ấn nút lệnh stop. Giao diện người dùng có thể tiếp tục đáp ứng các yêu cầu của người dùng bằng cách sử dụng một tuyến đoạn riêng cho hoạt động nạp trang web.

Để lập trình đa tuyến đoạn ta cần có một cách nhìn nhận khác về phần mềm. Ngoài việc xử lý tuần tự, các tác vụ còn có thể được xử lý đồng thời. Nghĩa là, nhiều tác vụ được thực hiện cùng một lúc mà không cần đợi tác vụ này hoàn thành mới tiến hành tác vụ khác. Đa tuyến đoạn còn có nghĩa là nhiều tuyến xử lý, cho phép một chương trình có nhiều thể hiện cùng hoạt động, cùng sử dụng chung bộ nhớ. Một ứng dụng có thể thực hiện nhiều tác vụ đồng thời và các tuyến đoạn có thể truy xuất tới các biến dữ liệu dùng chung để cùng làm việc hợp tác với nhau.

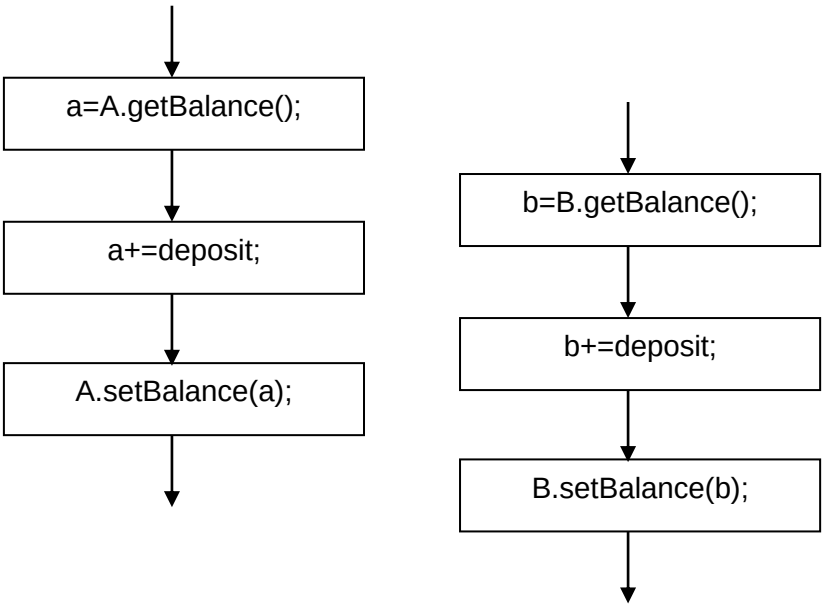
Nếu máy tính chỉ có một CPU, thì chỉ có một tuyến đoạn đang chạy tại một thời điểm cho trước. Hệ điều hành duy trì một hàng đợi các tuyến đoạn và phân bổ thời gian CPU cho chúng. Tuy nhiên, thời gian phân bổ cho một tuyến đoạn là công bằng trong nhiều tình huống, vì nó ngăn các tuyến đoạn khác thực hiện công việc. (Tình trạng này còn được gọi là đói tuyến đoạn). Ta cần phải có một cách nào đó để phân bổ thời gian CPU giành cho mỗi tuyến đoạn là như nhau.

Mục đích của tiến trình và tuyến đoạn đều là cho phép nhiều máy tính thực hiện nhiều tác vụ đồng thời.

Ví dụ: A là một tài khoản tại ngân hàng. Phương thức getBalance(): lấy ra giá trị của tài khoản. Phương thức setBalance(): chép lại giá trị vào bản ghi tài khoản. Xét ví dụ tuyến đoạn dưới đây



Hình 4.1: *Đơn tuyến đoạn*



Hình 4.2: *Đa tuyến đoạn (Multi-thread)*

Một ứng dụng có thể có nhiều tuyến đoạn. Khả năng làm việc với nhiều tuyến đoạn được gọi là đa tuyến đoạn. Đa tuyến đoạn cho phép bạn viết các chương trình hiệu quả, tận dụng tối đa CPU bằng cách duy trì thời gian trễ là tối thiểu.

Tại cùng một thời điểm có hai khách hàng cùng ghi vào cùng một tài khoản. Nếu thực hiện như vậy thì chỉ có tuyến đoạn thứ hai mới thực sự ảnh hưởng tới tài khoản, kết quả của giao dịch thứ nhất sẽ bị mất. Để khắc phục điều này cần có một cơ chế để thông báo một đối tượng đang được sử dụng hay không.

Vòng đời của một tuyến đoạn

Một tuyến đoạn có thể ở một trong bốn trạng thái sau trong suốt vòng đời của nó:

- new: Một tuyến đoạn mới là một tuyến đoạn được tạo ra bằng cách sử dụng toán tử new nhưng vẫn chưa được khởi động.
- runnable: Một tuyến đoạn ở trạng thái runnable mỗi khi phương thức start() của nó được kích hoạt. Điều này nghĩa là mã lệnh trong phương thức run() có thể được xử lý bất kỳ khi nào giành được quyền xử lý từ hệ điều hành.
- blocked (bị phong tỏa): Một tuyến đoạn chuyển vào trạng thái blocked nếu một trong các sự kiện sau xảy ra:
    - Phương thức sleep() của tuyến đoạn được gọi. Trong trường hợp này, tuyến đoạn vẫn ở trạng thái blocked cho tới khi hết một số ms (mili giây) xác định.
    - Tuyến đoạn gọi phương thức wait() của một đối tượng. Trong trường hợp này tuyến đoạn vẫn ở trạng thái blocked cho tới khi phương thức notify() hoặc notifyAll() được gọi từ một tuyến đoạn khác. Các phương thức wait(), notify()và notifyAll() thường được tìm thấy trong các phương thức đồng bộ synchronized của đối tượng.
    - Tuyến đoạn phong tỏa một thao tác vào/ra. Trong trường hợp này, tuyến đoạn bị phong tỏa cho tới khi hoạt động vào ra hoàn thành.
- dead (chết): Một tuyến đoạn thường ở trạng thái dead khi phương thức run() hoàn thành việc xử lý.

2. Tạo các ứng dụng đa tuyến đoạn với lớp Thread

Lớp java.lang.Thread cung cấp các phương thức để khởi động (start()), tạm dừng (suspend()), phục hồi (resume()) và dừng hẳn (stop()) một tuyến đoạn, cũng như kiểm soát các khía cạnh khác như độ ưu tiên của tuyến đoạn hoặc tên của tuyến đoạn gắn với nó. Cách đơn giản nhất để sử dụng lớp Thread là thừa kế lớp này và nạp chồng phương thức run(), phương thức này được gọi khi tuyến đoạn được khởi động lần đầu. Bằng cách nạp chồng phương thức run(), một tuyến đoạn có thể thực hiện một số tác vụ hữu ích ở hậu trường.

Chú ý: Cần nhớ rằng các tuyến đoạn không chạy tự động tại thời điểm chạy. Thay vào đó, ta phải gọi phương thức Thread.start(), nếu không tuyến đoạn sẽ không chạy.

Khuông dạng chung để tạo một ứng dụng đa tuyến đoạn bằng cách sử dụng lớp Thread

```
class C1 extends Thread
{
  public C1(){this.start();}
  public void run(){...}
}
```

Ví dụ: Viết chương trình tạo lập một ứng dụng đa tuyến đoạn. Tạo lập ra hai tuyến đoạn mỗi tuyến đoạn in ra một từ với tốc độ khác nhau bằng cách thừa kế lớp Thread

```
class PingPong extends Thread {
    String word;
        int delay;
        PingPong(String s, int d)
        {
                word =s;
                delay=d;
        }
        public void run()
        {
                try{
                        for(;;)
                        {
                                System.out.print(word+" ");
                                sleep(delay);
                        }
                }
                catch(InterruptedException e)
                {
                        return;
                }
        }
        public static void main(String[] args)
        {
                new PingPong("ping",33).start();
                new PingPong("PONG",100).start();
        }
}
```

3. Tạo ứng dụng đa tuyến đoạn với giao tiếp Runnable

Thừa kế lớp Thread là một cách để tạo ra ứng dụng đa tuyến đoạn nhưng nó không phải là giải pháp tốt nhất. Chúng ta đều biết rằng Java chỉ hỗ trợ đơn thừa kế nên việc cài đặt các ứng dụng đa thừa kế là không thể được nếu tạo ứng dụng đa tuyến đoạn bằng cách thừa kế từ lớp Thread. Một giải pháp có thể khắc phục điều này là thực thi giao tiếp java.lang.Runnable.

Giao tiếp Runnable định nghĩa duy nhất một phương thức run(). Các lớp thực thi giao tiếp này chỉ ra rằng chúng có thể chạy độc lập như một tuyến đoạn riêng. Giao tiếp này không định nghĩa bất kỳ phương thức nào khác hoặc cung cấp bất kỳ chức năng tuyến đoạn cụ thể nào. Mục đích duy nhất của nó là báo hiệu các lớp thực thi giao tiếp này có thể chạy như các tuyến đoạn. Khi một đối tượng thực thi giao tiếp Runnale được truyền cho constructor của một tuyến đoạn, và các phương thức start() của tuyến đoạn được gọi, phương thức run() sẽ tự động được gọi bởi tuyến đoạn vừa được tạo ra. Khi phương thức run() kết thúc xử lý, tuyến đoạn sẽ dừng hoạt động.

Việc sử dụng giao tiếp Runnable có một số ưu điểm so với thừa kế lớp Thread. Trước hết, lớp thực thi giao tiếp Runnable có thể tự do thừa kế từ một lớp khác. Thứ hai, cùng một đối tượng Runnable có thể được truyền cho nhiều tuyến đoạn, vì vậy một số tuyến đoạn tương tranh có thể sử dụng chung mã và thao tác trên cùng dữ liệu.

Khuôn dạng chung để tạo một ứng dụng đa tuyến đoạn bằng cách thực thi giao tiếp Runnable

```
class C2() implements Runnable
{
  public C2(){Thread t = new Thread(this);}
  public void run(){...}
}
```

Để lập trình tuyến đoạn ta phải sử dụng gói java.lang, tuy nhiên do gói ngày được mặc định nên ta không cần phải khai báo.

Ví dụ: Viết chương trình tạo lập một ứng dụng đa tuyến đoạn. Tạo lập ra hai tuyến đoạn mỗi tuyến đoạn in ra một từ với tốc độ khác nhau bằng cách thực thi giao tiếp Runnable.

```
class RunPingPong implements Runnable
{
 String word;
 int delay;
 RunPingPong(String w, int d)
 {
        word =w;
        delay=d;
 }
 public void run(){
        try{
                for(;;){
                        System.out.println(word+" ");
                        Thread.sleep(delay);
                }
        }
```

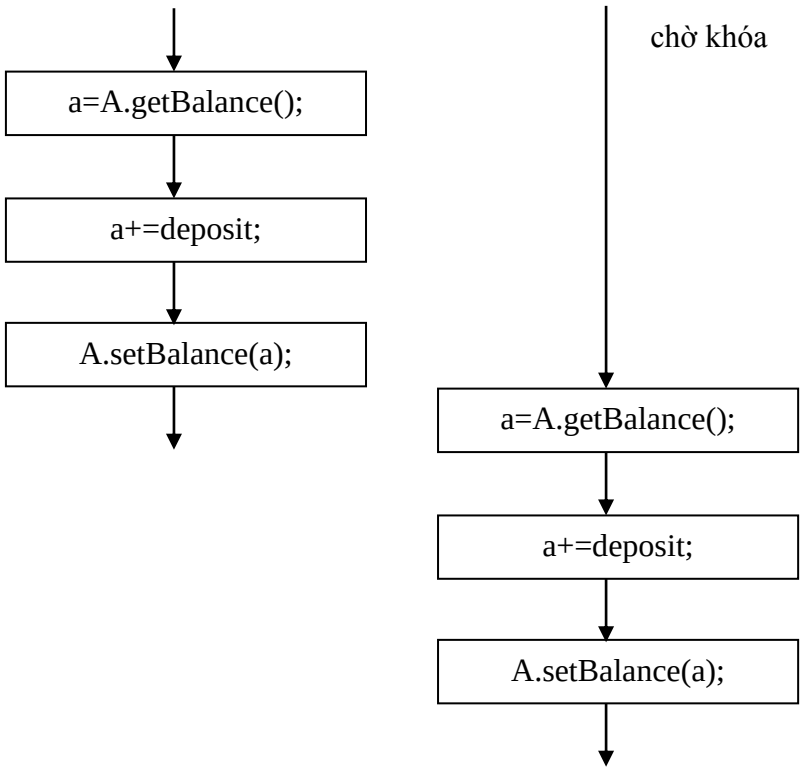
```
            catch(InterruptedException e)
            {
                  return;
            }
      }
      public static void main(String[] args)
      {
            Runnable ping = new RunPingPong("ping",33);
            Runnable pong = new RunPingPong("PONG",100);
            new Thread(ping).start();
            new Thread(pong).start();
      }
    }
```

4. Sự đồng bộ hóa (Synchronization)

Khi hai tuyến đoạn cần sử dụng cùng một đối tượng, có một khả năng có các thao tác đan xen nhau làm phá hỏng dữ liệu. Đa tuyến đoạn có một cơ chế để ngăn ngừa điều đó, bằng cách sử dụng phương thức chiếm dụng đối tượng. Nếu một đối tượng bị phong tỏa bởi một tuyến đoạn nào đó thì chỉ có tuyến đoạn đó mới có thể truy cập tới đối tượng.

4.1. Các phương thức synchronized

- Để làm cho một lớp có thể sử dụng được trong môi trường đa tuyến đoạn, các phương thức tương ứng sẽ được khai báo là synchronized.
- Nếu một tuyến đoạn kích hoạt một phương thức synchronized trên một đối tượng, đối tượng đó sẽ bị chiếm dụng bởi tuyến đoạn đó. Một tuyến đoạn khác kích hoạt phương thức synchronized trên cùng đối tượng đó sẽ bị phong tỏa cho tới khi khóa trên đối tượng được giải phóng.



Hình 4.3

Ví dụ: Cài đặt lớp Account:

```
class Account{
  private double balance;
  public Account(double initialDeposit)
  {
        balance = initialDeposit;
  }
  public synchronized double getBalance()
  {
        return balance;
  }
  public synchronized void setBalance(double amount)
  {
    balance+=amount;
  }
 }
}
```

4.2.Lệnh synchronized

Lệnh synchronized cho phép đồng bộ hóa một đối tượng mà không cần yêu cầu bạn tác động một phương thức synchronized trên đối tượng đó.

- Cú pháp

```
synchronized (expr)
        statement
```

Khi có được khóa, statement được xử lý giống như nó là một phương thức synchronized trên đối tượng đó.

Ví dụ: Chuyển các phần tử trong mảng thành các số không âm

```
public static void abs(int[] v)
{
 synchronized(v)
 {
  for(int i=0;i<v.length;i++)
        {
            if(v[i]<0) v[i]=-v[i];
        }
 }
 }
```

Java có thể chạy trên cả máy đơn và máy đa xử lý, với nhiều tuyến đoạn hay đơn tuyến đoạn.

5. Phương thức wait và notify

Cơ chế chiếm dụng đồng bộ hóa ngăn không cho các tuyến đoạn chồng chéo nhau. Nhưng trong một số trường hợp ta cũng cần phải cung cấp một cách nào đó để các tuyến đoạn truyền tin với nhau. Java cung cấp cho người sử dụng các phương thức cho phép các tuyến đoạn không bị xử lý chồng chéo mà vẫn có thể trao đổi thông tin với nhau bằng cách sử dụng các phương thức wait() và notify(). Để thực hiện điều này phương thức wait() được định nghĩa để cho phép một tuyến đoạn đợi cho tới khi một điều kiện nào đó xảy ra. Phương

thức notify() được định nghĩa để báo cho các tuyến đoạn biết sự kiện nào đó đã xảy ra. Các phương thức này được định nghĩa trong lớp Object và được thừa kế từ các lớp Object.

```
public class WaitNotify extends Thread
{
        public static void main(String args[]) throws Exception
        {
                Thread notificationThread = new WaitNotify();
                notificationThread.start();
                // Chờ tuyến đoạn cảnh báo kích hoạt sự kiện
                synchronized (notificationThread)
                {
                        notificationThread.wait();
                }
                // Báo cho người dùng biết phương thức wait đã hoàn thành
                System.out.println ("The wait is over");
        }
        public void run()
        {
                        System.out.println ("Hit enter to stop waiting thread");
                        try
                        {
                                System.in.read();
                        }
                        catch (java.io.IOException ioe)
                        {
                                // no code req'd
                        }
                        // Notify any threads waiting on this thread
                        synchronized (this)
                        {
                        this.notifyAll();
                        }
        }
}
```

Một số dạng của phương thức wait và notify

Tất cả các phương thức đều có trong lớp Object và hoạt động trên đối tượng hiện thời:

- *public final void wait(long timeout) throws InterruptedException*

  Tuyến đoạn hiện thời chờ cho tới khi được cảnh báo hoặc một khoảng thời gian timeout nhất định. Nếu timeout bằng 0 thì phương thức sẽ chỉ chờ cho tới khi có cảnh báo về sự kiện.

- public final void notify()

  Cảnh báo ít nhất một tuyến đoạn đang chờ một điều kiện nào đó thay đổi. Các tuyến đoạn phải chờ một điều kiện thay đổi trước khi có thể gọi phương thức wait nào đó.

- public final void notifyAll()

  Phương thức này cảnh báo tất cả các tuyến đoạn đang chờ một điều kiện thay đổi. Các tuyến đoạn đang chờ thường chờ một tuyến đoạn khác thay đổi điều kiện nào đó. Trong số các tuyến đoạn đã được cảnh báo, tuyến đoạn nào có độ ưu tiên cao nhất thì sẽ chạy trước tiên.

6. Lập lịch cho tuyến đoạn

Chương trình Java có thể chạy trên cả các máy đơn xử lý và đa xử lý với nhiều tuyến đoạn hoặc đơn tuyến đoạn. Java gán cho mỗi tuyến đoạn một độ ưu tiên để xác định cách tuyến đoạn đó được xử lý như thế nào so với các tuyến đoạn khác. Độ ưu tiên của các tuyến đoạn là các số nguyên. Các độ ưu tiên của tuyến đoạn chỉ có tính chất tương đối so với các tuyến đoạn khác. Các tuyến đoạn có độ ưu tiên cao nhất sẽ được xử lý trước tiên nếu không có gì đặc biệt. Các tuyến đoạn có độ ưu tiên thấp hơn sẽ chỉ chạy khi các tuyến đoạn có độ ưu tiên cao hơn bị phong tỏa.

Với một tuyến đoạn cho trước ta có thể xác định độ ưu tiên lớn nhất và độ ưu tiên nhỏ nhất nhờ các hằng số *Thread.MAX_PRIORITY*, *Thread.MIN_PRIORITY*. Độ ưu tiên chuẩn cho một tuyến đoạn mặc định là *Thread.NORM_THREAD*. Độ ưu tiên của tuyến đoạn hiện thời có thể bị thay đổi bất kỳ khi nào nhờ phương thức setPriority(). Để nhận về độ ưu tiên của một tuyến đoạn ta dùng phương thức getPriority().

Một số phương thức tĩnh của lớp Thread điều khiển lịch trình của tuyến đoạn hiện thời

- public static void sleep(long ms) throws InterruptedException

  Phương thức này đưa tiến đoạn hiện hành vào trạng thái nghỉ tối thiểu là ms (mili giây).

- public static void yeild()

  Phương này giành lấy quyền thực thi của tuyến đoạn hiện hành cho một trong các tuyến đoạn khác.

7. Bế tắc-Deadlock

Một kiểu lỗi đặc biệt mà ta cần phải tránh có liên quan đến đa nhiệm là bế tắc (deadlock), bế tắc xảy ra khi hai tuyến đoạn có một sự phụ thuộc xoay vòng trên một cặp đối tượng đồng bộ. Ví dụ, giả sử một tuyến đoạn chiếm dụng đối tượng X và một tuyến đoạn chiếm dụng đối tượng Y. Nếu tuyến đoạn chiếm dụng X cố gắng gọi bất kỳ phương thức đồng bộ trên Y, thì nó sẽ bị phong tỏa. Nếu tuyến đoạn chiếm dụng Y gọi phương thức đồng bộ trên X, tuyến đoạn sẽ chờ vô hạn.

Ví dụ:

```
class A
{
 synchronized void phuongthuc1(B b)
```

```
 {
        String tenTD=Thread.currentThread().getName();
        System.out.println(tenTD+" dang goi phuong thuc A.phuongthuc1()");
        try{
                Thread.sleep(1000);
        }
        catch(Exception e)
        {
                System.out.println("A bi ngat");
        }
        System.out.println(tenTD+" Dang thu goi B.phuongthuc4()");
        b.phuongthuc4();
 }
 synchronized void phuongthuc2()
 {
        System.out.println("Ben tron phuong thuc A.phuongthuc2()");
 }

}
class B
{
 synchronized void phuongthuc3(A a)
 {
        String tenTD=Thread.currentThread().getName();
        System.out.println(tenTD+" dang goi phuong thuc B.phuongthuc3");
        try{
                Thread.sleep(1000);
        }
        catch(Exception e)
        {
                System.out.println("B bi ngat");
        }
        System.out.println(tenTD+" Dang thu goi phuon thuc A.phuongthuc2()");
        a.phuongthuc2();
 }
 synchronized void phuongthuc4(){
        System.out.println("Ben trong phuong thuc B.phuongthuc4()");
 }
}
```

```
class Deadlock implements Runnable
{

        A a=new A();
        B b=new B();
 Deadlock()
 {
        Thread.currentThread().setName("MainThread");
        Thread t=new Thread(this,"RacingThread");
        t.start();
        a.phuongthuc1(b);
        System.out.println("Trong tuyen doan main");
 }
 public void run()
 {
        b.phuongthuc3(a);
        System.out.println("Trong tuyen doan main");
 }
 public static void main(String[] args)
 {
        new Deadlock();
 }
};
```

Kết quả thực hiện chương trình

```
C:\MyJava>java Deadlock

MainThread dang goi phuong thuc A.phuongthuc1()

RacingThread dang goi phuong thuc B.phuongthuc3

RacingThread Dang thu goi phuon thuc A.phuongthuc2()

MainThread Dang thu goi B.phuongthuc4()
```

Chương trình bị treo hay nói cách khác hai tuyến đoạn A và B rơi vào tình huống bế tắc, tuyến đoạn nó chờ tuyến đoạn kia giải phóng đối tượng mà đối tượng kia đang chiếm dụng và ngược lại, điều này dẫn đến chờ đợi nhau mà ta có thể gọi là tình trạng hoài vọng.

8. Điều khiển tuyến đoạn

8.1. Ngắt một tuyến đoạn Thread

Khi gọi phương thức Thread.sleep(int) thì phương thức này sẽ đặt tuyến đoạn vào trạng thái nghỉ trong một khoảng thời gian xác định nào đó. Tuy nhiên để kích hoạt một tuyến đoạn sớm hơn ta phải sử dụng ngắt tuyến đoạn. Ta có thể ngắt một tuyến đoạn bằng cách gọi phương thức *interrupt().* Tất nhiên, điều này cần một tuyến đoạn khác tham chiếu tới tuyến đoạn hiện thời.

```
public class SleepyHead extends Thread
```

```
{
// Run method is executed when thread first started
        public void run()
        {
                System.out.println ("I feel sleepy. Wake me in eight hours");
            try
            {
                // Sleep for eight hours
                Thread.sleep( 1000*60 );
                System.out.println ("That was a nice nap");
            }
            catch (InterruptedException ie)
            {
                System.err.println ("Just five more minutes....");
            }
 }
// Main method to create and start threads
            public static void main(String args[]) throws java.io.IOException
            {
                // Create a 'sleepy' thread
                Thread sleepy = new SleepyHead();
                // Start thread sleeping
                sleepy.start();
                // Prompt user and wait for input
                System.out.println ("Press enter to interrupt the thread");
                System.in.read();
                //              Interrupt the thread
                sleepy.interrupt();
            }
}
```

8.2 Kết thúc việc thực thi một tuyến đoạn

Đôi khi cần thiết phải kết thúc một tuyến đoạn trước khi tác vụ của nó hoàn thành. Ví dụ, nếu một client đang gửi các thông điệp tới một mail server trong một tuyến đoạn thứ hai, và người sử dụng muốn hủy bỏ thao tác, tuyến đoạn phải được dừng lại ngay tức thời. Một tuyến đoạn có thể gửi một yêu cầu ngừng việc thực thi tuyến đoạn tới một tuyến đoạn khác nhờ phương thức *Thread.stop().* Điều này đòi hỏi tuyến đoạn điều khiển duy trì một tham chiếu tới tuyến đoạn mà nó muốn dừng.

Ví dụ dưới đây minh họa cách sử dụng phương thức stop:

```
public class StopMe extends Thread
{
// Run method is executed when thread first started
        public void run()
        {
                int count = 1;
                System.out.println ("I can count. Watch me go!");
                        for (;;)
                        {
                                // Print count and increment it
                                System.out.print (count++ + " ");
                                // Sleep for half a second
                                try { Thread.sleep(500); }
                                catch(InterruptedException ie) {}
                                }
        }
// Main method to create and start threads
        public static void main(String args[]) throws java.io.IOException
        {
// Create and start counting thread
                Thread counter = new StopMe();
                counter.start();
// Prompt user and wait for input
                System.out.println ("Press any enter to stop the thread ounting");
                System.in.read();
                // Interrupt the thread
                counter.stop();
        }
}
```

C:\MyJava>java StopMe

Press any enter to stop the thread ounting

I can count. Watch me go!

1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23

Chương trình trên sẽ tiếp tục biến đếm cho tới khi ta nhấn một phím bất kỳ để dừng việc xử lý của tuyến đoạn.

8.3. Tạm dừng và phục hồi việc xử lý các tuyến đoạn

Trước Java 2, ta có thể được phép tạm dừng việc xử lý của một tuyến đoạn. Điều này có thể thực hiện nhờ phương thức Thread.suspend() và phục hồi hoạt động của tuyến đoạn nhờ Thread.resume(). Tuy nhiên trong các phiên bản Java 2 người ta khuyến cáo không nên sử dụng các phương thức này vì chúng có dẫn đến tình trạng hoài vọng (deadlock).

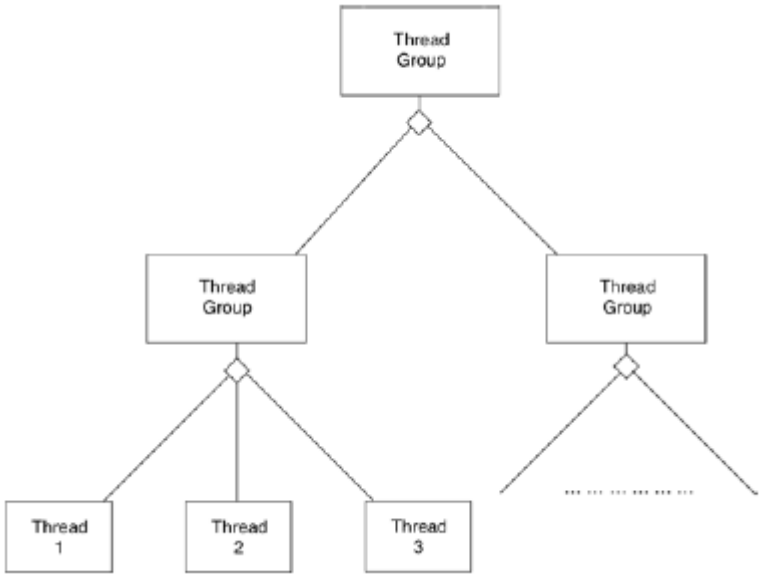
9. Nhóm các tuyến đoạn –ThreadGroup

Các tuyến đoạn có thể được tạo ra và được chạy riêng theo từng tuyến đoạn, song đôi khi làm việc với một nhóm các tuyến đoạn sẽ dễ dàng hơn so với làm việc với từng tuyến đoạn riêng rẽ tại từng thời điểm. Các thao tác ảnh hưởng tới các tuyến đoạn, như tạm dừng và phục hồi hoạt động của các tuyến đoạn, dừng hẳn hoặc ngắt các tuyến đoạn, có thể được thực hiện trên từng tuyến đoạn, nhưng điều này đòi hỏi các lập trình viên phải duy trì một danh sách các tuyến đoạn (bằng cách sử dụng các cấu trúc dữ liệu như vector hoặc mảng). Khi một thao tác được thực hiện, mỗi tuyến đoạn trong danh sách phải được duyệt và được xử lý riêng. Điều này tạo ra một khối lượng công việc khá lớn cho người lập trình vì cần phải viết nhiều đoạn mã phức tạp. Một giải pháp thay thế là nhóm các tuyến đoạn với nhau và áp dụng một thao tác trên nhóm mà không phải thực hiện thao tác trên từng tuyến đoạn riêng lẻ.

Java API hỗ trợ khả năng làm việc với các nhóm tuyến đoạn nhờ lớp ThreadGroup. Mục đích của lớp ThreadGroup là biểu diễn một tập hợp các tuyến đoạn, và cung cấp các phương thức tác động nhanh trên từng tuyến đoạn riêng trong nhóm. Nó còn cung cấp các cách thức để tập hợp các thông tin về các tuyến đoạn có liên quan nói chung. Nếu cần truy xuất tới từng tuyến đoạn riêng lẻ, ta có thể truy xuất thông qua ThreadGroup.

Khi JVM khởi động ứng dụng lần đầu tiên, tuyến đoạn chính sẽ chạy phương thức main(). Sau đó, ứng dụng tạo ra các nhóm tuyến đoạn một cách tự do, hoặc tạo ra các tuyến đoạn riêng không gắn với một nhóm nào cả. Một nhóm có thể bao gồm nhiều tuyến đoạn và ta có thể bổ sung thêm tuyến đoạn vào nhóm trong quá trình xử lý tuyến đoạn khi cần. Tuy nhiên không có cách nào để gỡ bỏ tuyến đoạn ra khỏi một nhóm.

Một nhóm các tuyến đoạn có thể chứa các nhóm tuyến đoạn khác được gọi là các nhóm tuyến con các tuyến đoạn. Các thao tác như dừng hoặc tạm dừng có thể được thực hiện trên các nhóm con hoặc nhóm cha. Khi một thao tác được áp dụng cho nhóm cha, thao tác sẽ được lan truyền tới nhóm con đó. Điều này có nghĩa là một thao tác có thể có tác động đến nhiều tuyến đoạn. Điều này khiến cho việc thiết kế ứng dụng trở nên đơn giản hơn.



Hình 4.4

9.1. Tạo một nhóm Thread

Các constructor

Lớp ThreadGroup cung cấp một số constructor sau:

- public ThreadGroup(String name) throws java.lang.SecurityException

Constructor này tạo ra một nhóm tuyến đoạn mới, nhóm này có tên được xác định bởi một xâu ký tự truyền vào như một tham số.

- ThreadGroup(ThreadGroup parentGroup,String name) throws java.lang.SecurityException

Tạo ra một nhóm tuyến đoạn mới được định danh bởi tên name. Nó cho phép một nhóm được lưu trữ như là một nhóm con.

Sử dụng một ThreadGroup

Mỗi khi được tạo ra, một nhóm tuyến đoạn có thể được sử dụng như một tuyến đoạn bình thường. Ta có thể tạm dừng, phục hồi, ngắt hoặc dừng nhóm tuyến đoạn bằng cách gọi phương thức thích hợp. Để sử dụng nhóm tuyến đoạn hiệu quả thì tuyến đoạn phải khác rỗng .

Các tuyến đoạn không được gắn với một nhóm cụ thể tại thời điểm tạo ra chúng. Vì thế, không thể gán một tuyến đoạn cho một nhóm sau đó, hoặc chuyển một tuyến đoạn từ nhóm này sang nhóm khác. Có ba constructor để thực hiện điều này

- Thread(ThreadGroup group, Runnable runnbale)
- Thread(ThreadGroup group, Runnable name)
- Thread(ThreadGroup group, Runnable runnbale, String name)

Mỗi khi tạo ra một nhóm các tuyến đoạn ta có thể tác động các phương thức trên nhóm.
Các phương thức

- int activeCount(): trả về số tuyến đoạn trong nhóm, và các nhóm con.
- int activeGroupCount(): trả về số nhóm con các tuyến đoạn
- boolean allowThreadSuspension(): chỉ ra tuyến đoạn bị tạm ngừng hay không.
- void checkAccess():
- void destroy(): hủy bỏ nhóm tuyến đoạn
- int enumerate(Thread[] threadList): đưa các tuyến đoạn trong nhóm vào một mảng các tuyến đoạn.
- int enumerate(Thread[] threadList, boolean subgroupFlag): đưa các tuyến đoạn trong nhóm vào một mảng các tuyến đoạn. Phương thức này đưa các tuyến đoạn trong nhóm bao gồm cả các tuyến đoạn nhóm con nếu biến logic boolean subgroupFlag được thiết lập là true.
- int enumerate(ThreadGroup[] groupList): đặt tất cả các nhóm con vào mảng

9.2. Minh họa về lớp ThreadGroup

```
public class GroupDemo implements Runnable{
 public static void main(String args[]) throws Exception
 {
        // Create a thread group
        ThreadGroup parent = new ThreadGroup("parent");
        // Create a group that is a child of another thread group
        ThreadGroup subgroup = new ThreadGroup(parent, "subgroup");
```

```
            // Create some threads in the parent, and subgroup class
            Thread t1 = new Thread ( parent, new GroupDemo() );
            t1.start();
            Thread t2 = new Thread ( parent, new GroupDemo() );
            t2.start();
            Thread t3 = new Thread ( subgroup, new GroupDemo() );
            t3.start();
            // Dump the contents of the group to System.out
            parent.list();
            // Wait for user, then terminate
            System.out.println ("Press enter to continue");
            System.in.read();
            System.exit(0);
        }

            public void run(){
            // Do nothing
            for(;;)
            {
            Thread.yield();
            }
        }
    }
```

Kết quả thực hiện chương trình

```
C:\MyJava>java GroupDemo
java.lang.ThreadGroup[name=parent,maxpri=10]
   Thread[Thread-0,5,parent]
   Thread[Thread-1,5,parent]
   java.lang.ThreadGroup[name=subgroup,maxpri=10]
      Thread[Thread-2,5,subgroup]
Press enter to continue
```

10. Một ví dụ minh họa việc sử dụng tuyến đoạn

Ví dụ

Chương trình vẽ đồng hồ số sử dụng tuyến đoạn và applet

```
import java.awt.*;
import java.util.Date;
import java.applet.*;
public class   Clock extends Applet implements Runnable
{
 Font f = new Font("Times New Roman",Font.BOLD,24);
 Date d = new Date();
```

```
Thread t;
public void init()
{
        resize(400,400);
}
public void start()
{
        if(t==null)
        {
                t= new Thread(this);
                t.start();
        }
}
public void stop()
{
         if(t==null)
        {
                 t.stop();
                 t=null;
        }
}
public void run()
{
        while(true)
        {
                d= new Date();
                repaint();
                try{
                        Thread.sleep(1000);
                }
                catch(InterruptedException e)
                {
                        return;
                }
        }
}
public  void paint(Graphics g)
{
        g.setFont(f);
```

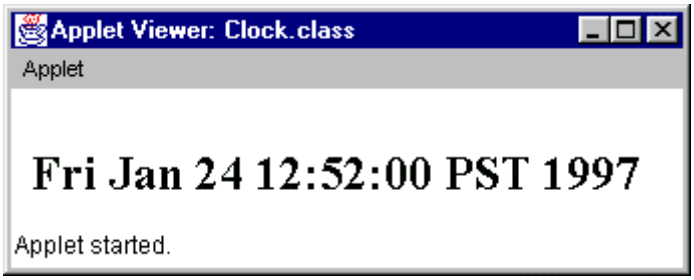
```
            g.drawString(d.toString(),10,50);
     }
}
```

```
<HTML>
<HEAD>
<TITLE> Clock </TITLE>
</HEAD>
<BODY>
<APPLET CODE ="Clock.class" WIDTH =200 HEIGHT = 100>
</APPLET>
</BODY>
</HTML>
```

Thực hiện chương trình

appletviewer Clock.html

Kết quả thực hiện



Hình 4.5

11. Kết luận

Những hiểu biết về lập trình đa tuyến đoạn có tầm quan trọng đối với các ứng dụng và các applet, đặc biệt là đối với môi trường mạng. Các mạng máy tính thường rất chậm và có độ tin cậy không cao, vì vậy chương trình mạng nên chạy trong một tuyến đoạn riêng biệt tách biệt với giao diện người dùng. Hơn thế nữa, phần mềm mạng tương tác với nhiều client hoặc server, ngoại trừ các thao tác đặc biệt nhanh (như nhận và gửi một gói tin). Chương trình cần có nhiều tuyến đoạn để các tương tác có thể xảy ra đồng thời. Trong các chương sau chúng ta sẽ xem xét cách ứng dụng tuyến đoạn trong việc xây dựng các chương trình mạng có xử lý tương tranh.