



پر نامه سازی پیش رفته

همروندی

مهدی مصطفی زاده

سرفصل مطالب

- مفهوم همروندي (Concurrency)
- مفهوم نخ (ریسمان، Thread)
- تعریف و اجرای نخ جدید در جاوا
- متدهای ردی (join, sleep) و (...)
- مفاهیم شرایط مسابقه، ناحیه‌ی بحرانی و انحصار متقابل
- متدها و نواحی همگامشده (synchronized methods & blocks)
- همگامسازی نخها با استفاده از متدهای wait و notify



طرح مساله

- پر نامه هایی که تا الان نوشته ایم، همگی از یک مسیر اجرایی تشکیل شده اند.
- در هر لحظه، فقط یک بخش از کد در حال اجراست و تا زمانی که اجرای آن خاتمه نیاید، بخش دیگری اجرا نمی شود.
- در پسیاری از اوقات، لازم است که پر نامه هایی پیویسیم که چند مسیر اجرایی، توامان پا هم اجرا شوند.
- مثال) در یک نرم افزار ویرایشگر متن (مثل Word)، توامان پا نمایش کلمات نوشته شده، املای کلمات پررسی می شود.
- به این دسته از پر نامه ها، همروند (Concurrency) و به این شیوه هی اجرا، همروندی (Concurrency) می گویند.



همروندی

- همروندی به این معنایست که دو یا چند کار (مسیر اجرایی) توانان پا هم اجرا شوند.
- الاما نیست که حتماً مسیرهای اجرایی به شکل همزمان اجرا شوند تا پگوییم همروندی داریم (کفايت همزمانی از دید کاربر).
- اگر مسیرهای اجرایی واقعاً به شکل همزمان اجرا شوند، علاوه پر همروندی، تواری نیز داریم.
- هر تواری، همروندی نیز است؛ اما هر همروندی الاما تواری نیست.



تعريف نخ

- به طور کلی، دو روش پرای تعریف نخ داریم:
- گسترش ردهی Thread
- پیاده‌سازی واسط Runnable
- مستقل از انتخاب هریک از دو روش فوق، لازم است منطق قابل اجرا در متدهای run پیاده‌سازی کنیم.



مثال: تعریف، ایجاد و اجرای نخ با استفاده از گسترش ردهی Thread

```
public class MyThread extends Thread {  
    @Override  
    public void run() {  
        for(int i = 0; i < 10; i++) {  
            System.out.println("MyThread: " + i);  
        }  
    }  
}
```

```
5   public static void main(String[] args) {  
6       Thread myThread = new MyThread();  
7       myThread.start();  
8       for(int i = 0; i < 10; i++) {  
9           System.out.println("main: " + i);  
10      }  
11  }
```



```
<terminated> Main [Java Application] F:\Mahdi\Java\jdk1.8.0_112\bin\javaw.exe (Jun 5, 2020, 10:39:16 F  
main: 0  
MyThread: 0  
main: 1  
MyThread: 1  
main: 2
```



مثال: تعریف، ایجاد و اجرای نخ با استفاده از پیاده‌سازی Runnable واسط

```
public class MyThread implements Runnable {  
    @Override  
    public void run() {  
        for(int i = 0; i < 10; i++) {  
            System.out.println("MyThread: " + i);  
        }  
    }  
}
```

```
5   public static void main(String[] args) {  
6       Thread myThread = new Thread(new MyThread());  
7       myThread.start();  
8       for(int i = 0; i < 10; i++) {  
9           System.out.println("main: " + i);  
10      }  
11  }
```



Mark... Pro... Serv... Data... Snip... Pro... Con... Sear... JUnit Deb...

```
terminated> Main [Java Application] F:\Mahdi\Java\jdk1.8.0_112\bin\javaw.exe (Jun 5, 2020, 10:29:05 PM)  
main: 0  
MyThread: 0  
main: 1  
MyThread: 1  
MyThread: 2  
main: 2  
MyThread: 3
```



متدهای ردی (sleep) Thread

```
7* public static void main(String[] args) {
8    try {
9        System.out.println(new Date());
10       Thread.sleep(10_000);
11       System.out.println(new Date());
12   } catch (InterruptedException e) {
13       e.printStackTrace();
14   }
15 }
```

The screenshot shows a Java application running in an IDE. The code prints the current date twice, with a 10-second delay between them. The terminal output shows the first print statement at 23:05:07 and the second at 23:05:17, confirming the sleep duration.

```
<terminated> Main [Java Application] F:\Mahdi\Java\jdk1.8.0_112\bin\javaw.exe (Jun 5, 2020, 11:05:07)
Fri Jun 05 23:05:07 GMT+03:30 2020
Fri Jun 05 23:05:17 GMT+03:30 2020
```

sleep

- متدهای ایستاده در ردی Thread که نخ چاری را به اندازه زمان مشخص، در نقطه‌ی فراموشانی نگه می‌دارد.



متدهای ردی (join) Thread

```
5* public static void main(String[] args) {  
6     Thread myThread = new MyThread();  
7     myThread.start();  
8     try {  
9         myThread.join();  
10    } catch (InterruptedException e) {  
11        e.printStackTrace();  
12    }  
13    for(int i = 0; i < 10; i++) {  
14        System.out.println("main: " + i);  
15    }  
16}
```

- متدهای join در ردی Thread که نجات چاری را در نقطه‌ی فراموشی نگه می‌دارد تا نهی که متدهای join روی آن صدازده شده، مخاتمه یا پد.

The screenshot shows a Java application running in an IDE. The code in the editor is as follows:

```
5* public static void main(String[] args) {  
6     Thread myThread = new MyThread();  
7     myThread.start();  
8     try {  
9         myThread.join();  
10    } catch (InterruptedException e) {  
11        e.printStackTrace();  
12    }  
13    for(int i = 0; i < 10; i++) {  
14        System.out.println("main: " + i);  
15    }  
16}
```

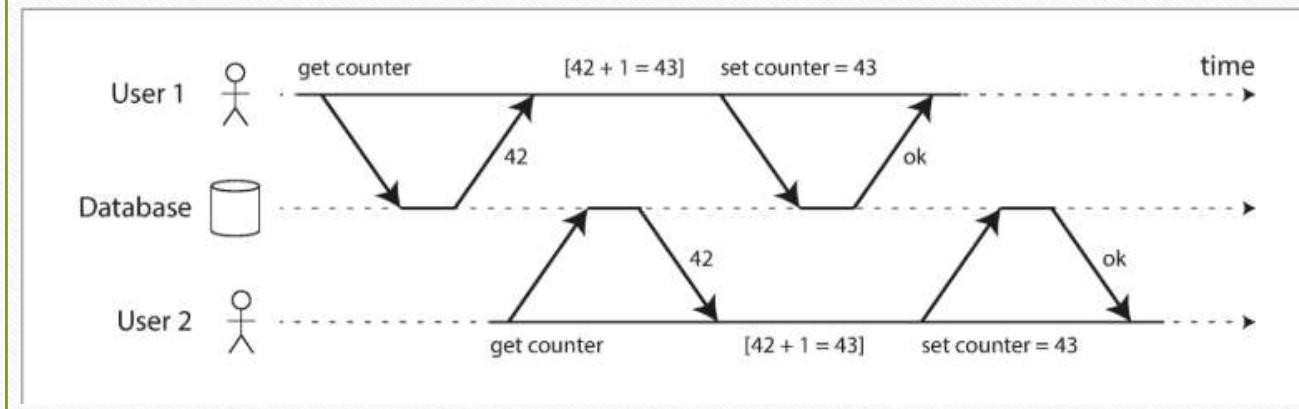
The terminal window below shows the execution of the program:

```
<terminated> Main [Java Application] F:\Mahdi\Java\jdk1.8.0_112\bin\javaw.exe (Jun 5, 2020, 11:13:33 F  
MyThread: 8  
MyThread: 9  
main: 0
```



شایط مسابقه، ناچیهی بحرانی و انحصار متقابل

- اگرچه هر نخ، حافظه‌ی Stack مجددی خود را دارد، اما حافظه‌ی Heap پین همه‌ی نخ‌های پرنامه مشترک است.
- گاهی اوقات، دسترسی هم‌مان دو یا چند نخ به یک شئ (منبع) مشترک مشکل آفرین مواهد پود.



- مثال) خرد اینترنتی پلیت هوایپما



شرايط مسابقه، ناچيهی بحراني و انحصار متقابل (ادامه)

- به شرایط ناشی از دسترسی همزمان دو یا چند نخ به منبع مشترک به شکلی که حداقل یکی از نخها، تغییردهندهی منبع پاشد، شرایط مسابقه (Race Condition) گفته می‌شود.
- به پختشی از کد که ورود همزمان پیش از یک نخ به آن موجب پذور شرایط مسابقه می‌شود، ناچيهی بحرانی (Critical Section) گفته می‌شود.
- به منظور چلوگیری از رخداد شرایط مسابقه، پایستی پا استفاده از امکانات زبان پردازه‌نویسی (چاوا)، مانع ورود همزمان پیش از یک نخ به ناچيهی بحرانی شویم و به عبارتی، انحصار متقابل (Mutual Exclusion) پذرار کنیم.



استفاده از متدهای synchronized برای انحصار متقابل

```
public class Flight {  
    private int capacity;  
    private List<Person> passengers;  
  
    public Flight(int capacity) {  
        this.capacity = capacity;  
        passengers = new ArrayList<>(capacity);  
    }  
  
    public synchronized void register(Person p) {  
        if(passengers.size() == capacity)  
            return;  
        passengers.add(p);  
    }  
  
    public int getNumberOfPassengers() {  
        return passengers.size();  
    }  
}
```

```
public static void main(String[] args) {  
    Flight flight = new Flight(250);  
    Thread myThread = new MyThread(flight);  
    myThread.start();  
    for(int i = 100; i < 200; i++) {  
        flight.register(new Person(i));  
    }  
    System.out.println(flight.getNumberOfPassengers());  
}  
  
public class MyThread extends Thread {  
    private Flight flight;  
  
    public MyThread(Flight flight) {  
        this.flight = flight;  
    }  
  
    @Override  
    public void run() {  
        for(int i = 0; i < 100; i++) {  
            flight.register(new Person(i));  
        }  
    }  
}
```



استفاده از متدهای synchronized برای انحصار متقابل (ادامه)

- خروجی مثل قبیل، همواره ۲۰۰ است.
- اگر وارثی synchronized را از تعریف متدهای register حذف کنیم، به ازای هربار اجراء، خروجی‌های مختلفی می‌گیریم.
- امکان تعریف چند متدهای synchronized در رده وجود دارد. مثلاً می‌توان متدهای unregister را نیز به شکل مشابه تعریف کرد.
- مجموعه‌ی همه‌ی متدهای synchronized با هم یک ناحیه‌ی پسندی دارند. بنابراین، حداقل یک نخ درون این ناحیه از یک شی فعال مواهد بود و مابقی نخ‌ها تا اتمام کار آن نخ، منتظر می‌مانند.
- اگر متدهای synchronized ایستا باشند، حداقل یک نخ درون این ناحیه از رده فعال مواهد بود.



استفاده از نواحی synchronized برای انحصار متقابل

- گاهی اوقات بخشی از کد متند (و نه همه‌ی آن) ناجیهی پس از این موضع می‌توان از نواحی (blocks) استفاده کرد.

```
public void register(Person p) {  
    synchronized(passengers) {  
        if(passengers.size() == capacity)  
            return;  
        passengers.add(p);  
    }  
}
```

استفاده synchronized

- هر نجف برای ورود به ناجیهی synchronized با پیستی قفل شیء تعیین شده را دریافت کند.

- متند synchronized پر روی قفل this تعریف می‌شود.

- حداکثر یک نجف درون مجموعه‌ی نواحی synchronized تعریف شده روی یک شیء فعال مخواهد بود.



همگام‌سازی نخ‌ها با استفاده از متدهای `notify` و `wait`

- پیش از این، نمونه‌ای محدود از همگام‌سازی پیش دو نخ را که توسط متدهای `join` محقق می‌شود، مشاهده کردیم.
- در پسیاری از اوقات، نیازمند همگام‌سازی منعطف‌تری هستیم: یک نخ در یک نقطه از اجراء منتظر ماند تا نخ دیگری او را پیدار کند.
- بدین منظور، دو متدهای `wait` و `notify` در رده‌ی `Object` تعریف شده‌اند.
- اگر یک نخ، متدهای `wait` و `notify` را صدا پزند، در همان نقطه منتظر می‌ماند تا زمانیکه نخ دیگری، متدهای `notify` را روی همان شیء، صدا پزند.



همگام‌سازی نخ‌ها با استفاده از متدهای notify و wait (مثال)

```
public static void main(String[] args) {
    Object lock = new Object();
    Thread myThread = new MyThread(lock);
    myThread.start();
    try {
        Thread.sleep(5000);
    } catch (InterruptedException e) {
        e.printStackTrace();
    }
    System.out.println("Hi!");
    synchronized (lock) {
        lock.notify();
    }
}
```

```
public class MyThread extends Thread {
    private Object lock;

    public MyThread(Object lock) {
        this.lock = lock;
    }

    @Override
    public void run() {
        synchronized (lock) {
            try {
                lock.wait();
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
        }
        System.out.println("Bye!");
    }
}
```



همگام‌سازی نخ‌ها با استفاده از متدهای wait و notify (چند نکته)

- متند wait و notify حتماً پاید درون ناحیه‌ی synchronized تعریف شده پر روی شئی‌ای که wait یا notify روی آن صدازده می‌شود، مورد استفاده قرار گیرند؛ در غیر آنصورت استثنای IllegalMonitorStateException پرتاپ می‌شود.
- سوال: اگر wait درون ناحیه‌ی synchronized صدازده شود، پظبور نخ دیگر، notify را که آن هم درون ناحیه‌ی synchronized است، صدازند؟
- پاسخ: به محض فراخوانی wait، قفل شئ آزاد می‌شود و نخ دیگر می‌تواند وارد ناحیه‌ی synchronized شود. (نقض انجصار متقابل؟؟)
- پاسخ: خیر. چرا که پس از فراخوانی notify توسط نخ دوم، نخ اول قفل را به دست می‌آورد، اما در همان نقطه‌ی wait پاکی می‌مائد تا نخ دوم از ناحیه‌ی synchronized خارج شود.



همگام‌سازی نخ‌ها با استفاده از متدهای notify و wait (مثال)

```
5 public static void main(String[] args) throws InterruptedException {  
6     Container container = new Container();  
7     Thread myThread = new MyThread(container);  
8     myThread.start();  
9     for(int i = 1; true; i++) {  
10         container.setValue(i);  
11     }  
12 }
```

```
<terminated> Main [Java Application] F:\Mahdi\Java\jdk1.8.0_112  
52926422  
52927191  
52928198  
52928517  
52928517  
52928517
```

- یک نخ به عنوان نویسندهٔ مقادیر صحیح در یک طرف و نخ دیگر به عنوان مواندۀٔ مقادیر.



همگام‌سازی با استفاده از notify و wait (ادامه‌ی مثال)

```
public static void main(String[] args) throws InterruptedException {
    Container container = new Container();
    Object rLock = new Object();
    Object wLock = new Object();
    Thread myThread = new MyThread(container, rLock, wLock);
    myThread.start();
    for(int i = 1; true; i++) {
        synchronized (wLock) {
            if(container.getValue() != null)
                try {
                    wLock.wait();
                } catch (InterruptedException e) {
                    e.printStackTrace();
                }
        }
        container.setValue(i);
        synchronized (rLock) {
            rLock.notify();
        }
    }
}

public void run() {
    while(true) {
        synchronized (rLock) {
            if(container.getValue() == null) {
                try {
                    rLock.wait();
                } catch (InterruptedException e) {
                    e.printStackTrace();
                }
            }
            System.out.println(container.getValue());
            synchronized (wLock) {
                container.setValue(null);
                wLock.notify();
            }
        }
    }
}
```



InterruptedException

- سوال: آیا یک نخ، می‌تواند نخ دیگر را خاتمه دهد؟

```
5 public static void main(String[] args) throws InterruptedException {
6     Thread myThread = new MyThread();
7     myThread.start();
8     myThread.interrupt();
9     System.out.println(1);
10 }
```

```
<terminated> Main [Java Application] F:\Mahdi\Java\jdk1.8.0_112\bin\javaw.exe (Jun 8, 2020, 12:19:13 PM)
1
2
java.lang.InterruptedException: sleep interrupted
at java.lang.Thread.sleep(Native Method)
at main.MyThread.run(MyThread.java:17)
```

```
public class MyThread extends Thread {
    @Override
    public void run() {
        for(int i = 0; i<= 100000; i++) {

        }
        System.out.println(2);
    }
    try {
        sleep(10_000);
    } catch (InterruptedException e) {
        e.printStackTrace();
    }
}
```

- پاسخ: خیر. فقط می‌تواند از نخ دوم مُواهش کند که خاتمه یابد (مُتوقف شود). با فراخوانی متده interrupt پر روی نخ موردنظر.



(ادامه) InterruptedException

The screenshot shows an IDE interface with two tabs: Main.java and MyThread.java. The MyThread.java tab is active, displaying the following code:

```
5  @Override
6  public void run() {
7      for(int i = 0; i<= 100000; i++) {
8
9      }
10     if(interrupted());
11     System.out.println(2);
12     try {
13         sleep(10_000);
14     } catch (InterruptedException e) {
15         e.printStackTrace();
16     }
17 }
```

The Main.java tab shows a partial implementation of the run() method:

```
5  @Override
6  public void run() {
7      for(int i = 0; i<= 100000; i++) {
8
9      }
10     if(interrupted());
11     System.out.println(2);
12     try {
13         sleep(10_000);
14     } catch (InterruptedException e) {
15         e.printStackTrace();
16     }
17 }
```

The output window at the bottom shows the terminal output of the application:

```
<terminated> Main [Java Application] F:\Mahdi\Java\jdk1.8.0_112\bin\javaw.exe (Jun 8, 2020, 12:12:12)
1
2
java.lang.InterruptedException: sleep interrupted
    at java.lang.Thread.sleep(Native Method)
    at main.MyThread.run(MyThread.java:13)
```



متدهای setPriority و setDaemon

- با استفاده از فراخوانی متدهای `setDaemon` و `setPriority` می‌توان آن را به شکل شیخ تعریف کرد.
- نخهایی را شیخ گوییم که په تنها یک نخ های غیرشیخ وجود داشته باشند و حاتمه یاپند.
- مثال) نخی که عمل زبانه روپی خودکار را انجام می‌دهد.
- با استفاده از فراخوانی متدهای `setPriority` و `setDaemon` می‌توان اولویت نسبی نخ را تعیین کرد.

