Week 9

2020年10月12日 9:41

Java 网络编程
https://www.runoob.com/java/java-networking.html

java反射机制（1）- 知识点总结Java Reflection API操作
https://blog.csdn.net/Mark_LQ/article/details/50085465

Java URL处理
https://www.runoob.com/java/java-url-processing.html

Java多线程总结（1） — 创建线程的两种方式
https://blog.csdn.net/Mark_LQ/article/details/50292969

Java LinkedList
https://www.runoob.com/java/java-linkedlist.html

Java HashSet
https://www.runoob.com/java/java-hashset.html

Java HashMap
https://www.runoob.com/java/java-hashmap.html

Java Iterator（迭代器）
https://www.runoob.com/java/java-iterator.html

Java Object 类
https://www.runoob.com/java/java-object-class.html

命令行 -od -c

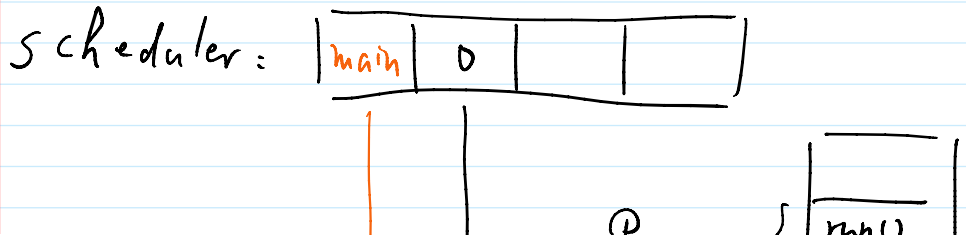
Output:

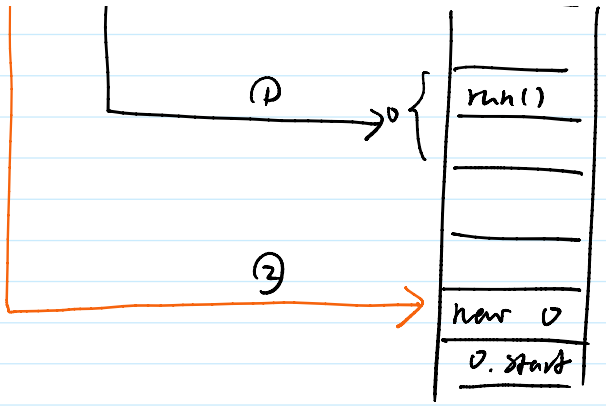
```
% java Self
% od -c self.ser
0000000 254 355  \0 005   s   r  \0 023   j   a   v   a   .   u   t   i
0000020   l   .   H   a   s   h   t   a   b   l   e 023 273 017   %   !
0000040   J 344 270 003  \0 002   F  \0  \n   l   o   a   d   F   a   c
 ...
0000160       M   o   v   i   e   t  \0 016   A       l   i   t   t   l
0000200   e       V   o   i   c   e   x
0000210
```

-od convert file to readable, but not comprehensiable
-c convert to character form

```
public class Thread_0 extends Thread    {
    private String info;
public Thread_0 (String info) {
        this.info    = info;
    }
public void run () {
      System.err.println(info);
    }
public static void main (String args []) {
        Thread_0 aT4_0 = new Thread_0("first");
        // Thread_0 aT4_1 = new Thread_0("second");
        // Thread   aT4_2 = new Thread_0("third");

        aT4_0.start();
        // aT4_1.start();
        // aT4_2.start();
    }
}
```

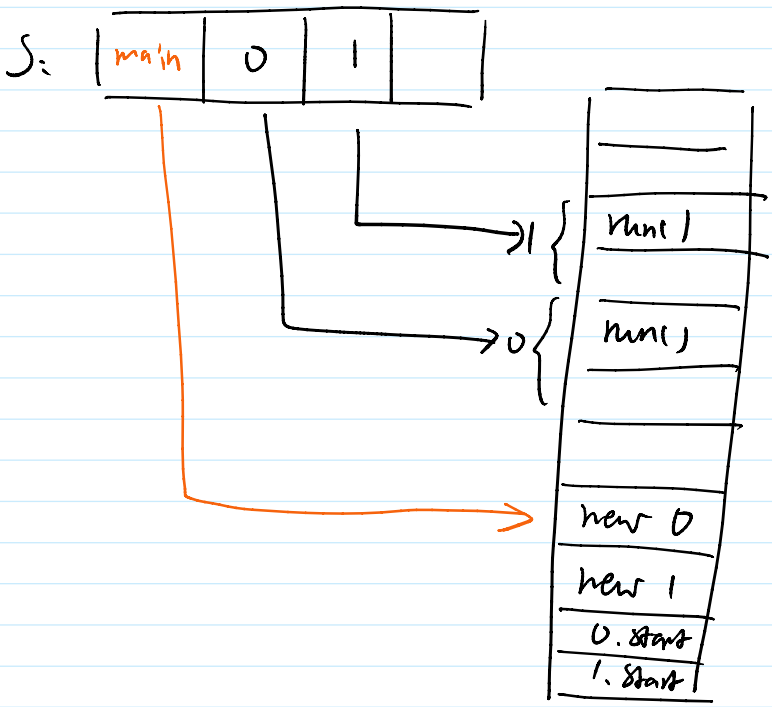




either run ① or ②

```
public class Thread_0 extends Thread    {
    private String info;
public Thread_0 (String info) {
        this.info    = info;
    }
public void run () {
      System.err.println(info);
    }
public static void main (String args []) {
        Thread_0 aT4_0 = new Thread_0("first");
        Thread_0 aT4_1 = new Thread_0("second");
        // Thread   aT4_2 = new Thread_0("third");

        aT4_0.start();
        aT4_1.start();
        // aT4_2.start();
    }
}
```



output : first second OR second first

```
public class Thread_0 extends Thread    {
    private String info;
public Thread_0 (String info) {
        this.info    = info;
    }
public void run () {
      System.err.println(info);
    }
public static void main (String args []) {
        Thread_0 aT4_0 = new Thread_0("first");
        Thread_0 aT4_1 = new Thread_0("second");
        // Thread   aT4_2 = new Thread_0("third");

        aT4_0.setDaemon(true);
        aT4_0.start();
        aT4_1.start();
        // aT4_2.start();
    }
}
```

The output would be:

1)main -> 1 -> 2
first
second

2)second -> 1 -> main
second
first

3)main -> 2 -> 1
second

```
public class Thread_0 extends Thread    {
    private String info;
public Thread_0 (String info) {
        this.info    = info;
    }
public void run () {
      System.err.println(info);
    }
public static void main (String args []) {
        Thread_0 aT4_0 = new Thread_0("first");
        Thread_0 aT4_1 = new Thread_0("second");
        // Thread   aT4_2 = new Thread_0("third");

        aT4_0.setDaemon(true);
        aT4_1.setDaemon(true);
        aT4_0.start();
        aT4_1.start();
        // aT4_2.start();
    }
}
```

The output would be:

1)main -> 1 -> 2
first
second

2)2 -> 1 -> main
second
first

3)main -> 2 -> 1
Nothing

4)1 -> main -> 2
first

```
public class Thread_0 extends Thread    {
    private String info;
public Thread_0 (String info) {
        this.info    = info;
    }
public void run () {
      try { sleep(1000); } catch ( InterruptionException e) {}
      System.err.println(info);
    }
public static void main (String args []) {
        Thread_0 aT4_0 = new Thread_0("first");
        Thread_0 aT4_1 = new Thread_0("second");
        // Thread   aT4_2 = new Thread_0("third");


        aT4_0.setDaemon(true);
        aT4_1.setDaemon(true);
        aT4_0.start();
        aT4_1.start();
        // aT4_2.start();
    }
}
```

`try { sleep(1000); } catch ( InterruptionException e) {}` tells the scheduler that it cannot execute this thread for a while, so this operation will force the main thread to terminate first, but not guarantee to do so.

The output would be:

1)main -> 1 -> 2
first
second

2)2 -> 1 -> main
second
first

3)main -> 2 -> 1
Nothing

4)1 -> main -> 2
first

System.exit() will terminate all threats if one of threats execute it.

```
public class Thread_example extends Thread     {
      int value = 0;    // static?
      int id;
public Thread_example (int id) {
            this.id = id;
      }
public int getValue() {
            return value;
      }
      public void compute()   {
            if ( id == 1 )    {
                  value = 1;
            }
            if ( id == 2 )    {
                  value = 2;
            }
      }
      public void run () {
            compute();
      }
public static void main (String args []) {
            Thread_example aT1  = new Thread_example(1);
            Thread_example aT2  = new Thread_example(2);
            aT1.start();
            aT2.start();
            System.out.println(aT1.getValue() + aT2.getValue());
      }
}
```

Possible output:

1) 0 + 0; main -> 1 -> 2
2) 1 + 0; 1 -> main -> 2
3) 1 + 2; 1 -> 2 -> main
4) 0 + 2; 2 -> main -> 1

```
public class Thread_example extends Thread      {
      static int value = 0;
      int id;
public Thread_example (int id) {
            this.id = id;
      }
public int getValue() {
            return value;
      }
      public void compute()   {
            if ( id == 1 )    {
                  value = 1;
            }
            if ( id == 2 )    {
                  value = 2;
            }
      }
      public void run () {
            compute();
      }
public static void main (String args []) {
            Thread_example aT1  = new Thread_example(1);
            Thread_example aT2  = new Thread_example(2);
            aT1.start();
            aT2.start();
            System.out.println(aT1.getValue() + aT2.getValue());
      }
}
```

Possible output:

1) 0 + 0; main -> 1 -> 2
2) 1 + 1; 1 -> main -> 2
3) 2 + 2; 2 -> main -> 1
4) 0 + 1; main ->aT1.getValue() -> 1 -> aT2.getValue()
5) 1 + 2; 1 -> main -> aT1.getValue() -> 2 -> aT2.getValue()
6) 1 + 0; not possible; prerequisite: in Java, executing order will be always from left to right; if not, this answer would be possible;
7) 2 + 1; 2 -> main -> aT1.getValue() -> 1 -> aT2.getValue()

```
public class Thread_example extends Thread      {
      static int value = 0;
      int id;
public Thread_example (int id) {
            this.id = id;
      }
public int getValue() {
            return value;
      }
      public void compute()   {
            if ( id == 1 )    {
                  value = 1;
            }
            if ( id == 2 )    {
                  value = 2;
            }
      }
      public void run () {
            compute();
      }
public static void main (String args []) {
            Thread_example aT1  = new Thread_example(1);
            Thread_example aT2  = new Thread_example(2);
            aT1.run();
            aT2.run();
            System.out.println(aT1.getValue() + aT2.getValue());
      }
}
```

不执行start()，直接执行run()，不会产生多线程的问题，这种调用的方法和调用其他普通方法是一样的，所以最后的output是确定且唯一的

homework: use sleep() to generate all possible outputs above

```
public class Thread_example extends Thread      {
      static int value = 0;
      int id;
public Thread_example (int id) {
            this.id = id;
      }
public int getValue() {
            return value;
      }
      public void compute()   {
            if ( id == 1 )    {
                  value = 1;
            }
            if ( id == 2 )    {
                  value = 2;
            }
      }
      public void run () {
            compute();
      }
public static void main (String args []) {
            Thread_example aT1  = new Thread_example(1);
            Thread_example aT2  = new Thread_example(2);
            aT1.start();
            aT2.run();
            System.out.println(aT1.getValue() + aT2.getValue());
      }
}
```

Possible output:

@fengkeyleaf