ThreadとRunnable

オブジェクト指向プログラミング特論

2020年度

只木進一：理工学研究科

今日のテーマ

- ThreadとRunnable
- Thread 間の同期
  - キーワード“synchronized”による保護

今日のサンプルプログラム

- https://github.com/oop-mc-saga/Thread

Thread

- 一つのアプリケーション中で、処理を分割し、非同期的に実行する単位
- thread間でメモリ（変数領域）を共有できる
- javaでは、
  - GUIはthreadで動いている
  - 任意のクラスをthread化できる

Runnableインターフェース

- Threadインスタンスとして指定
- run()メソッド
  - Threadから一度だけ起動される
- 制御にはvolatileな変数を用いる
  - volatile : 変わりやすい
  - 値の変化が直ちに行われる

Threadのメソッド

- start()
  - 指定されたRunnableインスタンスのrun()メソッドを起動
- sleep()
  - 指定された時間(ミリ秒)の間、停止する
- stop()は使わない

クラスをThreadとして実行する方法

- Runnableインターフェースを付ける
  - run()メソッドの実装
- Runnableインターフェースの実装として、対象クラスを起動

Thread.example0

- SampleWithThread
  - RunnableでないSampleクラスを起動する例
- SampleRunnable
  - SampleにRunnableインタフェースを付けて起動する例

```
public class Sample {

    protected volatile boolean running = true;
    protected int c = 0;
    private final int id;

    public Sample(int id) {
        this.id = id;
    }

    public void update() {
        Date date = new Date();
        System.out.println(id + ":" + c + " " + date.toString());
        c++;
        if (c > 10) {
            running = false;
        }
    }

    public boolean isRunning() {
        return running;
    }

}
```

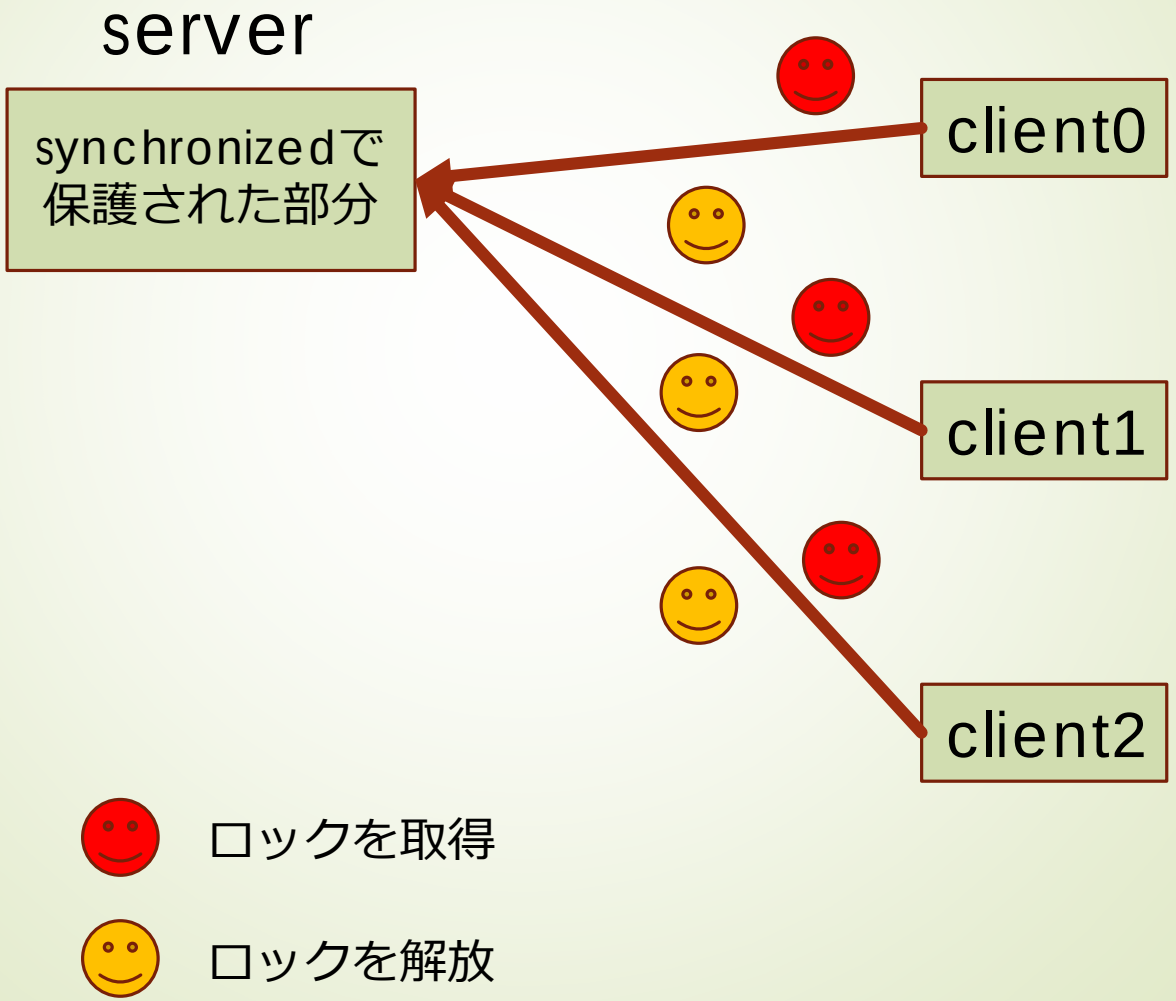
```
12      public static void main(String[] args) {
13          Thread thread0 = new Thread(new Runnable() {
14              Sample s = new Sample(1);
15
16              public void run() {
17                  while (s.isRunning()) {
18                      s.update();
19                      try {
20                          Thread.sleep(1000);
21                      } catch (InterruptedException e) {
22                      }
23                  }
24              }
25          });
26          thread0.start();
27      }
```

```
7   public class SampleRunnable extends Sample implements Runnable {
8
9       public SampleRunnable(int id) {
10          super(id);
11      }
12
13      /**
14       * ランダムな時間間隔でupdate()を実行
15       */
16      @Override
        public void run() {
18          while (running) {
19              update();
20              int t = (int) (1000 * Math.random());
21              try {
                    Thread.sleep(t);
23              } catch (InterruptedException e) {
24              }
25          }
26      }
27
28      /**
29       * @param args the command line arguments
30       */
31      public static void main(String[] args) {
32          new Thread(new SampleRunnable(1)).start();
33          new Thread(new SampleRunnable(2)).start();
34          Thread t = new Thread(new SampleRunnable(3));
35          t.start();
36      }
37
38  }
```

同期

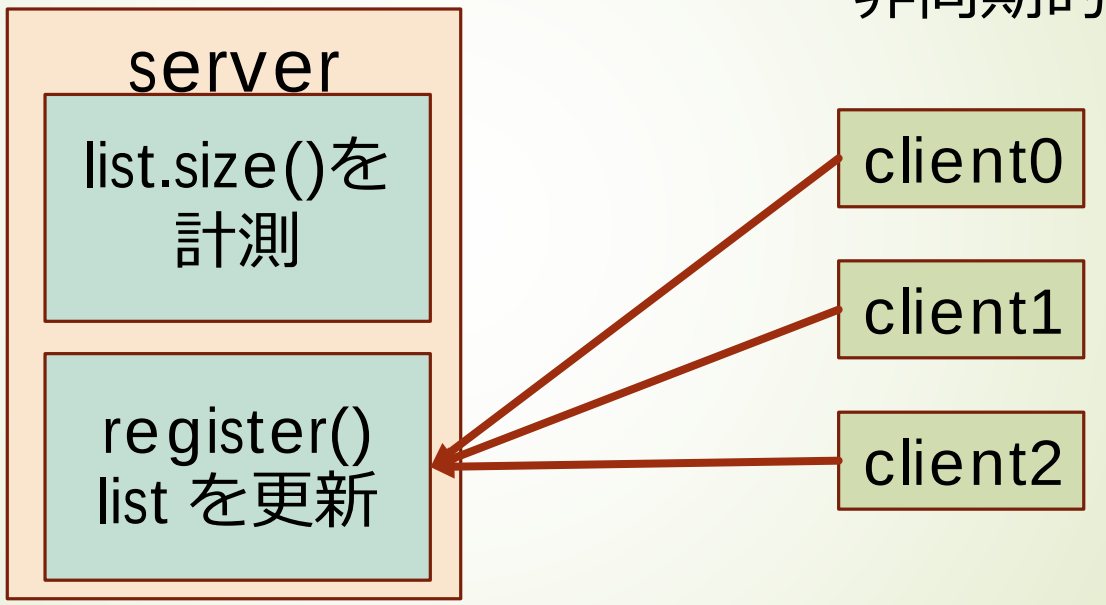
- threadは、同一アプリケーション内のデータを触る可能性がある
  - 必要に応じて同期が必要
- メソッド・オブジェクトを保護
  - synchronized修飾子
  - メソッドを一度に一つのスレッドからのみ利用

クラスインスタンスの保護



例：非同期的クライアントがサーバへアクセス

非同期的に動作



一度に一つだけ利用
できるように制限

Thread.example1

一つのクラスインスタンス内に複数のsynchronized

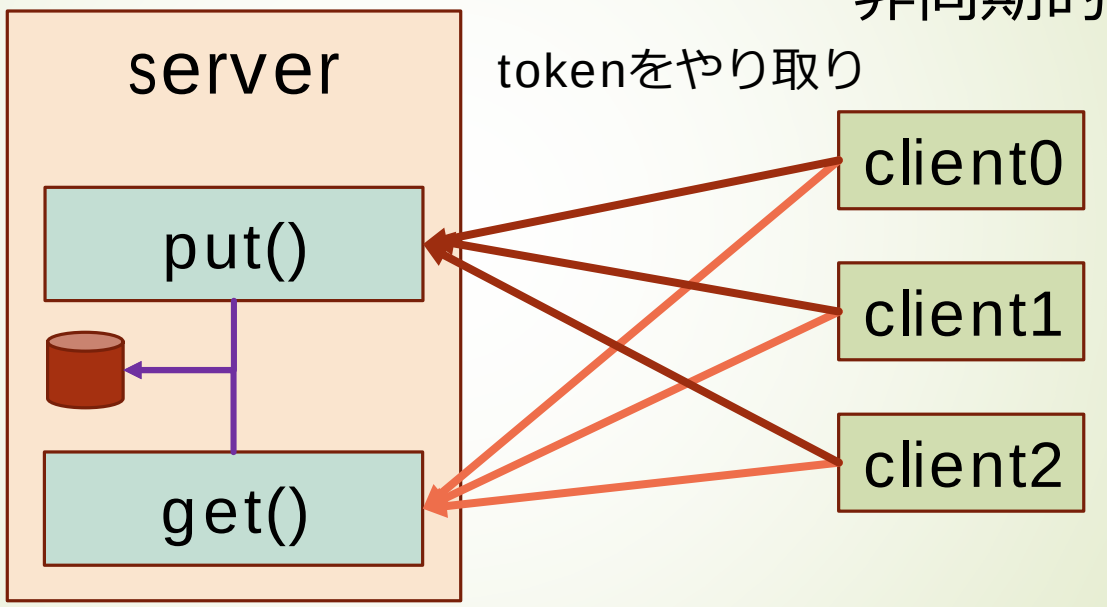
- synchronizedで保護されたメソッド・オブジェクトのうちの一つしかアクセスできない
  - クラスインスタンスには、一つのロック
- 例：リストを読むメソッドと書くメソッドを排他的に制御

```
    public void run() {
        while (running) {
            //listのロックが外れるのを待つ
            synchronized (messageList) {
                if (messageList.size() == max) {
                    running = false;
                }
            }
            try {
                Thread.sleep(10);
            } catch (InterruptedException e) {
            }
        }
    }
```

```
    synchronized public void register(Client client,
            int c, String dateStr) {
        Date date = new Date();
        //クライアントが接続しようとした時刻と実際に接続した時刻を記
        String ss = client + ":" + c + " "
                + dateStr + "->" + date.toString();
        messageList.add(ss);
        System.out.println(ss);
        try {
            Thread.sleep(1000);
        } catch (InterruptedException e) {
        }
    }
```

例：非同期的クライアントがサーバへアクセス

非同期的に動作



一度に一つだけ利用できるように制限

Thread.example2