ThreadとRunnable

オブジェクト指向プログラミング特論

2018年度

只木進一：工学系研究科

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とできる

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
  - wait/notifyを使えばvolatileでなくても可

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

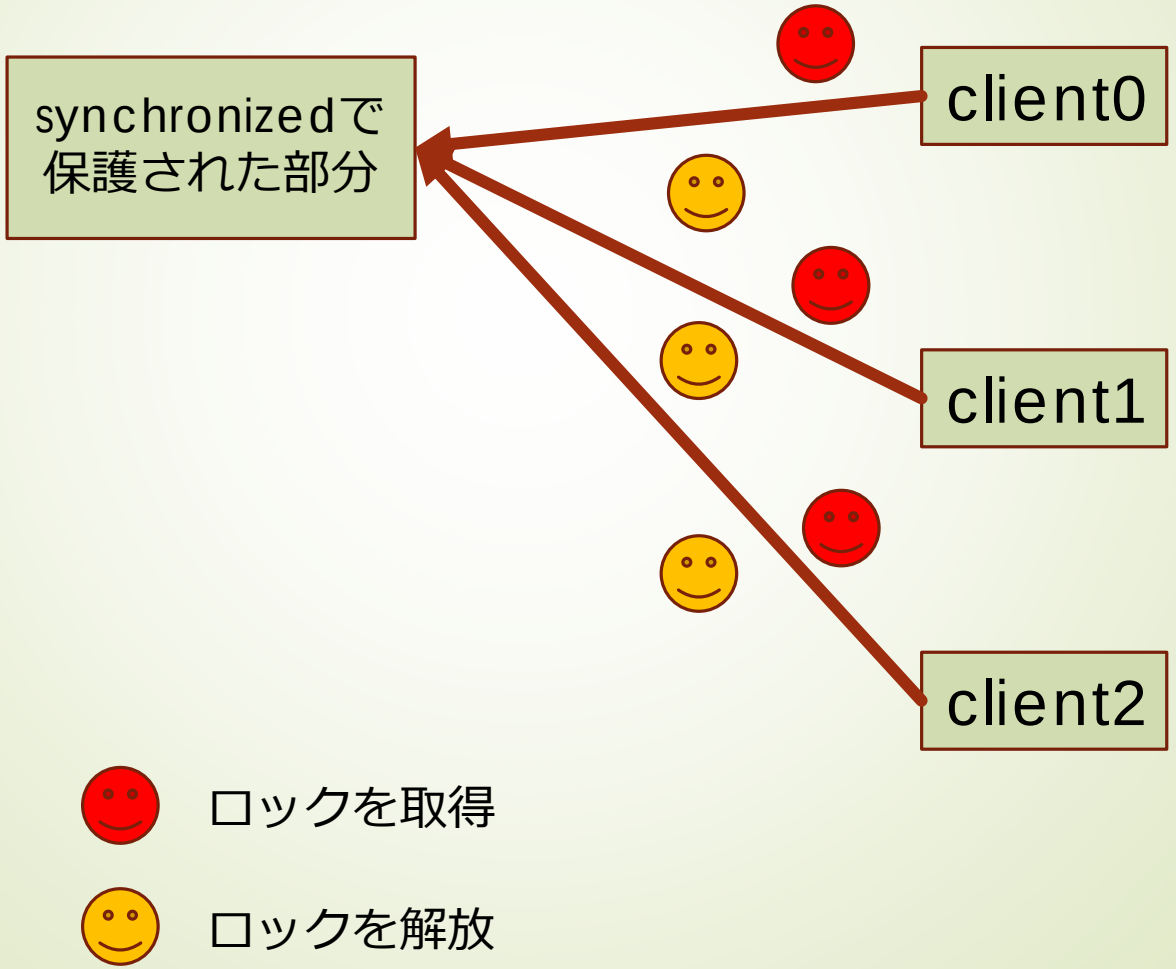
- Runnableインターフェースを付ける
  - run()メソッドの実装
- Runnableインターフェースの実装として、対象クラスを起動

Thread.example0

同期

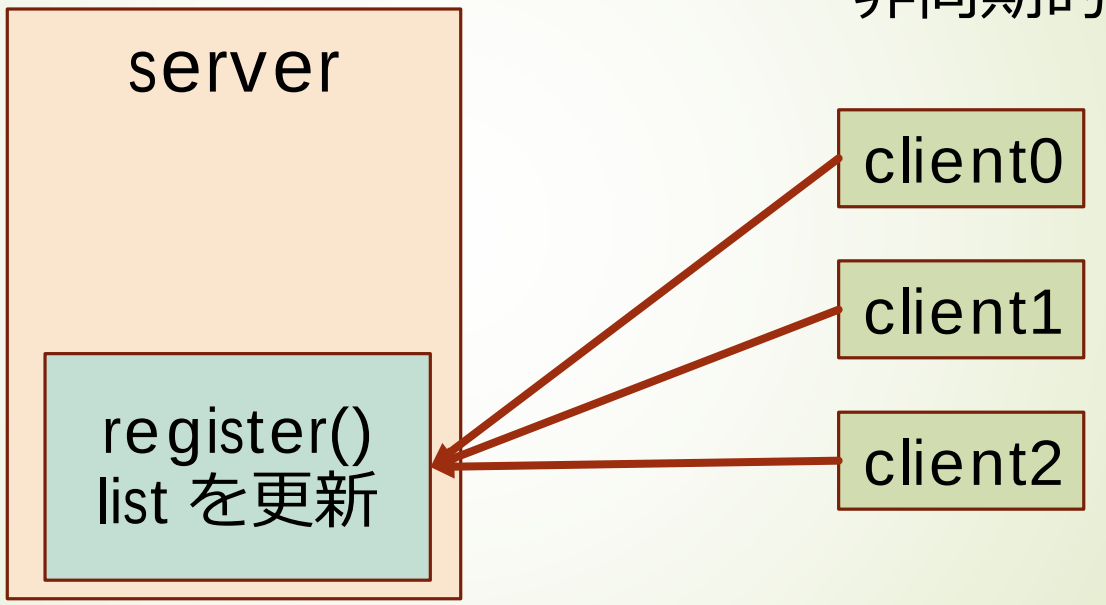
- threadは、同一アプリケーション内のデータを触る可能性がある
  - 必要に応じて同期が必要
- メソッド・オブジェクトを保護
  - synchronized修飾子
  - メソッドが一度に一つのスレッドからのみ利用

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

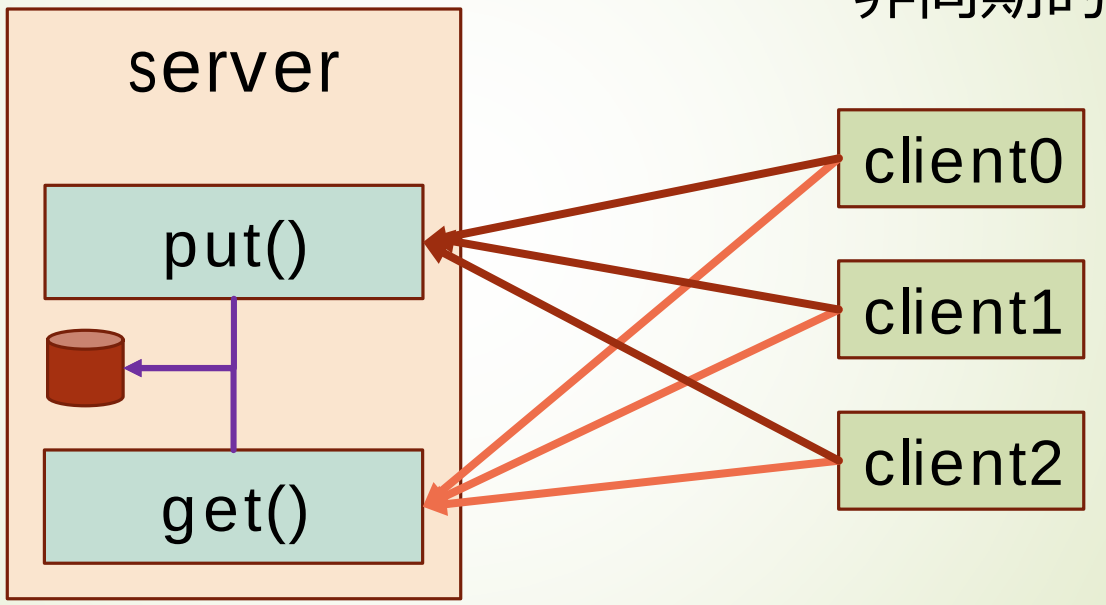
Thread.example1

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

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

さらに柔軟な制御

- wait()
  - sleep状態となり、ロックを解放する
- notify ()
  - wait()でsleepしているスレッドを起こす。
- notifyAll()
  - wait()でsleepしているすべてのスレッドを起こす。

Thread.example3

Sample.java

```java
package example0;

import java.util.Date;

/**
 * これだけではRunnableではないクラス
 * @author tadaki
 */
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;
    }

}
```

SampleRunnable.java

```java
package example0;

/**
 * RunnableにしたSmapleクラス
 * @author tadaki
 */
public class SampleRunnable extends Sample implements Runnable {

    public SampleRunnable(int id) {
        super(id);
    }

    /**
     * ランダムな時間間隔でupdate()を実行
     */
    @Override
    public void run() {
        while (running) {
            update();
            int t = (int) (1000 * Math.random());
            try {
                Thread.sleep(t);
            } catch (InterruptedException e) {
            }
        }
    }

    /**
     * @param args the command line arguments
     */
    public static void main(String[] args) {
        new Thread(new SampleRunnable(1)).start();
        new Thread(new SampleRunnable(2)).start();
    }

}
```

SampleWithThread.java

```
package example0;

/**
 * SampleをRunnableに拡張せずにthreadとして実行
 * @author tadaki
 */
public class SampleWithThread {

    /**
     * @param args the command line arguments
     */
    public static void main(String[] args) {
        Thread thread0 = new Thread(new Runnable() {
            Sample s = new Sample(1);

            public void run() {
                while (s.isRunning()) {
                    s.update();
                    try {
                        Thread.sleep(1000);
                    } catch (InterruptedException e) {
                    }
                }
            }
        });
        thread0.start();
    }

}
```

Client.java

```java
package example1;

import java.util.Date;

/**
 * クライアントクラス
 *
 * @author tadaki
 */
public class Client implements Runnable {

    private final int id;
    private final Server server;//接続先サーバ
    private volatile boolean running = true;
    private int c = 0;//接続回数をカウント
    public static final int Cmax = 5;//最大接続回数

    public Client(int id, Server server) {
        this.id = id;
        this.server = server;
    }

    /**
     * サーバへの接続の具体
     */
    private void connect() {
        Date date = new Date();
        server.register(this, c, date.toString());
        c++;
    }

    /**
     * ランダムな時間間隔でserverへアクセスする
     */
    @Override
    public void run() {
        while (running) {
            connect();
            int t = (int) (1000 * Math.random());
            try {
                Thread.sleep(t);
            } catch (InterruptedException e) {
            }
            if (Cmax <= c) {
                running = false;
            }
```

Client.java

```
        }
    }

    @Override
    public String toString() {
        return "client-" + id;
    }
}
```

Server.java

```java
package example1;

import java.util.ArrayList;
import java.util.Date;
import java.util.List;

/**
 * サーバクラス
 *
 * @author tadaki
 */
public class Server implements Runnable {

    private final List<String> messageList;
    private final int max;
    private volatile boolean running = true;

    public Server(int max) {
        messageList = new ArrayList<>();
        this.max = max;
    }

    /**
     * 定期的にmessageListのサイズを調べ、終了を確認する
     */
    @Override
    public void run() {
        while (running) {
            synchronized (messageList) {//listのロックが外れるのを待つ
                if (messageList.size() == max) {
                    running = false;
                }
            }
            try {
                Thread.sleep(10);
            } catch (InterruptedException e) {
            }
        }
    }

    /**
     * クライアントの接続をmessageListへ登録する。
     * 一度に一つのクライアントが利用できる。
     * 1秒に一つしか登録できない
     *
     * @param client
```

Server.java

```
     * @param c
     * @param dateStr
     */
    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録
        String ss = client + ":" + c + " " + dateStr + "->" +
date.toString();
        messageList.add(ss);
        System.out.println(ss);
        try {
            Thread.sleep(1000);
        } catch (InterruptedException e) {
        }
    }

    /**
     * @param args the command line arguments
     */
    public static void main(String[] args) {
        int n = 5;//クライアント数
        Server server = new Server(n * Client.Cmax);
        //サーバをthreadとして起動
        new Thread(server).start();
        for (int i = 0; i < n; i++) {//クライアントをthreadとして起動
            new Thread(new Client(i, server)).start();
        }
    }

}
```

Token.java

```java
package example2;

/**
 *
 * @author tadaki
 */
public class Token {
    public final int t;

    public Token(int t) {
        this.t = t;
    }

    public String toString(){
        return String.valueOf(t);
    }
}
```

Client.java

```
package example2;

import java.util.LinkedList;
import java.util.Queue;

/**
 * クライアントクラス
 * @author tadaki
 */
public class Client implements Runnable{

    private final int id;
    private final Server server;
    private volatile boolean running = true;
    private final Queue<Token> tokens;

    public Client(int id, Server server) {
        this.id = id;
        this.server = server;
        tokens = new LinkedList<>();
    }

    /**
     * 一回の動作
     */
    private void update(){
            if(!tokens.isEmpty()){//tokenがあればputする
                running=server.put(this, tokens.poll());
            }
            Token t = server.get(this);//サーバからtokenを取得
            if(t!=null){
                if(t==Server.falseToken)running=false;
                else{
                    tokens.add(t);
                }
            }
    }
    @Override
    public void run() {
        while(running){
            update();
            int timeout = (int) (1000 * Math.random());
            try {
                Thread.sleep(timeout);
            } catch (InterruptedException e) {
            }
```

Client.java

```
        }
    }

    @Override
    public String toString(){
        return "client-" + id;
    }
}
```

Server.java

```java
package example2;

import java.util.ArrayList;
import java.util.Date;
import java.util.HashSet;
import java.util.LinkedList;
import java.util.List;
import java.util.Queue;
import java.util.Set;

/**
 *
 * @author tadaki
 */
public class Server implements Runnable {

    protected final Queue<Token> tokens;
    public static final Token falseToken = new Token(-1);
    protected final int numClient;
    protected final Set<Client> endClients;//通信終了クライアント
    protected final List<String> log;//通信記録
    protected boolean running = true;
    protected final Date endTime;

    public Server(int numClient, int maxSecond) {
        this.numClient = numClient;
        log = new ArrayList<>();
        tokens = new LinkedList<>();
        for (int i = 0; i < numClient; i++) {
            tokens.add(new Token(i));
        }
        endClients = new HashSet<>();
        long startTime = new Date().getTime();
        //停止時刻設定
        endTime = new Date();
        endTime.setTime(startTime + maxSecond * 1000);
    }

    @Override
    public void run() {
        while (running) {
            if (endClients.size() == numClient) {
                System.out.println("All clients close");
                running = false;
            }
            //あらかじめ定めた時刻を過ぎたらサーバ停止
```

```
            Date now = new Date();
            if (now.after(endTime)) {
                running = false;
            }
            try {
                Thread.sleep(10);
            } catch (InterruptedException e) {
            }
        }
    }

    /**
     * クライアントがtokenを得る
     *
     * @param client
     * @return
     */
    synchronized public Token get(Client client) {
        Token b = getSub(client);
        try {
            Thread.sleep(1000);
        } catch (InterruptedException e) {
        }
        return b;
    }

    protected Token getSub(Client client){
        Date date = new Date();
        Token b = null;
        StringBuilder sb = new StringBuilder();
        sb.append(date).append(" ").append(client);
        if (!running) {
            sb.append(" close");
            endClients.add(client);
            b = falseToken;
        } else {
            if (tokens.isEmpty()) {
                sb.append(" no token");
            } else {
                b = tokens.poll();
                sb.append(" get ").append(b);
            }
        }
        sb.append(" ").append(token2str());
        System.out.println(sb.toString());
        return b;
```

```
    }

    /**
     * クライアントがtokenを返す
     *
     * @param client
     * @param t
     */
    synchronized boolean put(Client client, Token t) {
        if (running) {
            putSub(client, t);
            try {
                Thread.sleep(1000);
            } catch (InterruptedException e) {
            }
        }
        return running;
    }

    protected void putSub(Client client, Token t) {
        Date date = new Date();
        tokens.add(t);
        StringBuilder sb = new StringBuilder();
        sb.append(date).append(" ").append(client).append(" put ").append(t);
        sb.append(" ").append(token2str());
        log.add(sb.toString());
        System.out.println(sb.toString());
    }

    protected String token2str() {
        StringBuilder sb = new StringBuilder();
        sb.append("[");
        if (tokens.size() > 0) {
            tokens.stream().forEachOrdered(t -> sb.append(t).append(","));
            int last = sb.lastIndexOf(",");
            sb.replace(last, last + 1, "]");
        } else {
            sb.append("]");
        }
        return sb.toString();
    }

    /**
     * @param args the command line arguments
     */
    public static void main(String[] args) {
```

Server.java

```
        int n = 5;
        Server server = new Server(n, 60);
        new Thread(server).start();
        for (int i = 0; i < n; i++) {
            new Thread(new Client(i, server)).start();
        }
    }

}
```

ServerExt.java

```
package example3;

import example2.*;
import java.util.Date;

/**
 *
 * @author tadaki
 */
public class ServerExt extends Server {

    public ServerExt(int numClient, int maxSecond) {
        super(numClient, maxSecond);
    }

    @Override
    synchronized public void run() {
        while (running) {
            if (endClients.size() == numClient) {
                System.out.println("All clients close");
                running = false;
            }
            try {
                System.out.println("Server sleeps");
                wait(2000);
            } catch (InterruptedException ex) {
            }
            Date now = new Date();
            if (now.after(endTime)) {
                System.out.println("Server time out");
                running = false;
            }
        }
    }

    /**
     * クライアントがtokenを得る
     *
     * @param client
     * @return
     */
    synchronized public Token get(Client client) {
        Token b = getSub(client);
        notify();
        try {
            wait(1000);
```

```
            } catch (InterruptedException e) {
            }
            return b;
        }

        /**
         * クライアントがtokenを返す
         *
         * @param client
         * @param t
         */
        synchronized public boolean put(Client client, Token t) {
            if (running) {
                putSub(client, t);
                notify();
                try {
                    wait(1000);
                } catch (InterruptedException e) {
                }
            }
            return running;
        }

        /**
         * @param args the command line arguments
         */
        public static void main(String[] args) {
            int n = 5;
            Server server = new ServerExt(n, 60);
            new Thread(server).start();
            for (int i = 0; i < n; i++) {
                new Thread(new Client(i, server)).start();
            }
        }

}
```