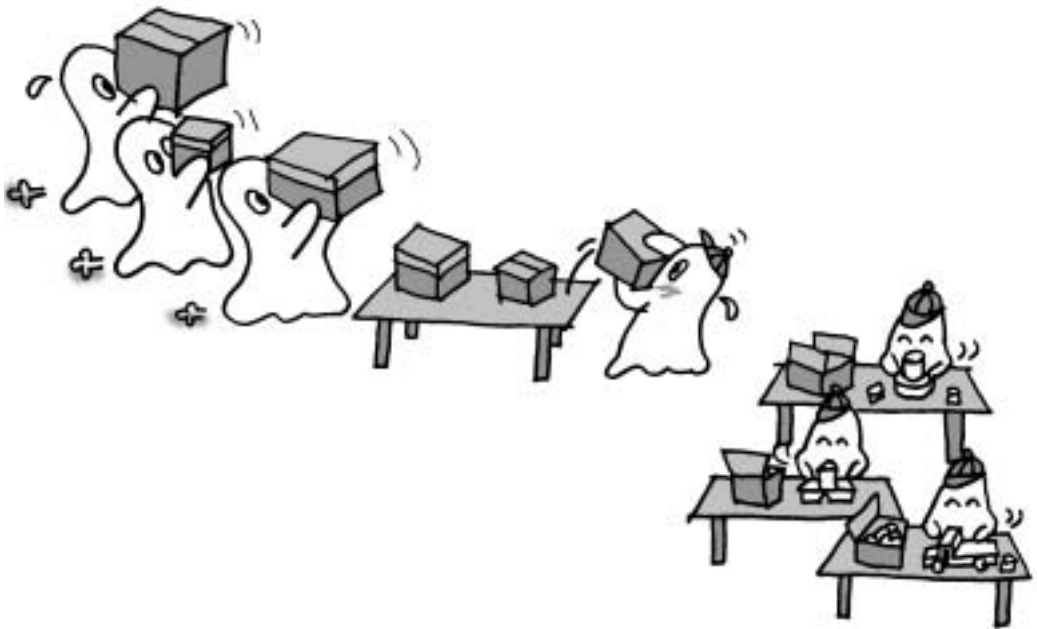


第8章

Worker Thread

仕事ができるまで待ち、
仕事きたら働く



Worker Threadパターン

ある作業所の話です。ここでは作業者がプラモデルを組み立てています。

作業を依頼する人が、プラモデルの箱をたくさん作業所に持ってきて、机の上に並べます。

届けられたプラモデルは、作業者が1つ1つ組み立てなければなりません。作業者は、まず机に並べられたプラモデルの箱を取りにいきます。そして、箱の中に入っている説明書を読んで組み立てはじめます。作り終えた作業者は、また次の箱を取りにいきます。箱がなかったら、新しいプラモデルの箱が届くまで待ちます…。

この章では、

ワーカー・スレッド
Worker Threadパターン

について学びましょう。

workerというのは「仕事をする人」「作業者」という意味です。Worker Threadパターンでは、ワーカースレッド (worker thread : 作業者スレッド) が仕事を1つずつ取りにいき、処理を行います。仕事がなかったら、ワーカースレッドは、新しい仕事が届くまで待ちます。

Worker Threadのことを、バックグラウンド・スレッド Background Thread(背景スレッド)と呼ぶこともあります。また、ワーカースレッドをたくさん保持している場所のほうに注目して、スレッド・プール Thread Poolと呼ぶこともあります。

サンプルプログラム

Worker Thread パターンを使ったサンプルプログラムを読んでみましょう。登場するクラス一覧を Table 8-1 に示します。サンプルプログラムは以下のような動作をします。

ClientThread クラスのスレッドが、Channel クラスに仕事のリクエスト（依頼）を出します（仕事といっても、依頼者の名前とリクエストの番号を表示するだけですけれど）。

Channel クラスのインスタンスはワーカースレッド（WorkerThread）を5人かかえています。ワーカースレッドはみんな、仕事のリクエストがくるのを待っています。

仕事のリクエストがやってくると、ワーカースレッドは、Channel から仕事のリクエストを1個もらって処理します。処理が終わったワーカースレッドはChannel のところに戻ってきて、「次のリクエストがこないかな」と待ちます。

Table 8-1 クラス一覧

名前	解説
Main	動作テスト用のクラス
ClientThread	仕事のリクエストを出すスレッドを表すクラス
Request	仕事のリクエストを表すクラス
Channel	仕事のリクエストを受け取り、ワーカースレッドに渡すクラス
WorkerThread	ワーカースレッドを表すクラス

Fig.8-1 サンプルプログラムのクラス図

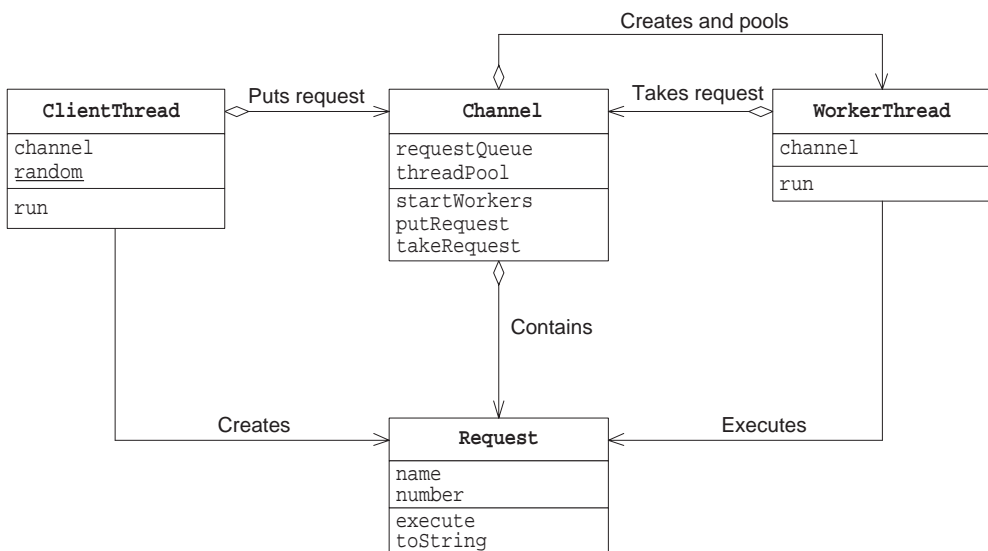
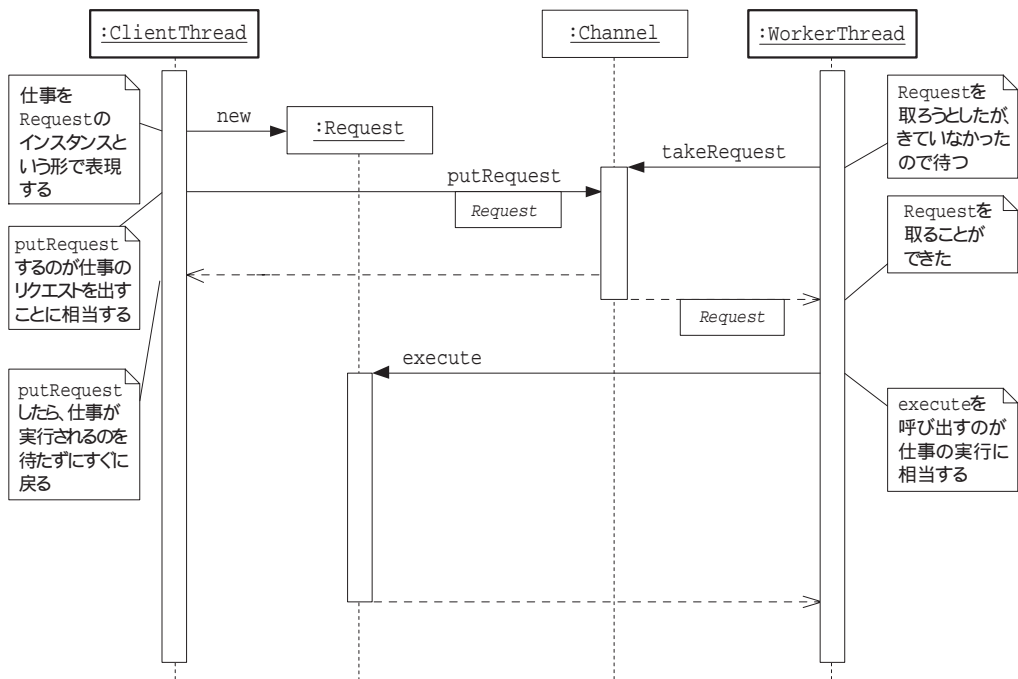


Fig.8-2 サンプルプログラムのシーケンス図



Mainクラス

Mainクラス (List 8-1) は、5人のワーカーレッドを持つChannelのインスタンスを1個作ります。そしてそれを、3人のClientThreadのインスタンス (Alice , Bobby , Chris) と共有させます。

List 8-1 実行テスト用 Main クラス (Main.java)

```

public class Main {
    public static void main(String[] args) {
        Channel channel = new Channel(5); // ワーカーレッドの個数
        channel.startWorkers();
        new ClientThread("Alice", channel).start();
        new ClientThread("Bobby", channel).start();
        new ClientThread("Chris", channel).start();
    }
}

```

ClientThread クラス

ClientThread クラス (List 8-2) は、仕事のリクエストを出すクラスです。「仕事のリクエストを出す」という行為は、このサンプルプログラムでは、

- Request のインスタンスを作る
- そのインスタンスを Channel クラスの putRequest メソッドに渡す

という処理に対応しています。動きに変化をつけるため、乱数を使って sleep しています。

List 8-2 仕事のリクエストを出す ClientThread クラス (ClientThread.java)

```
import java.util.Random;

public class ClientThread extends Thread {
    private final Channel channel;
    private static final Random random = new Random();
    public ClientThread(String name, Channel channel) {
        super(name);
        this.channel = channel;
    }
    public void run() {
        try {
            for (int i = 0; true; i++) {
                Request request = new Request(getName(), i);
                channel.putRequest(request);
                Thread.sleep(random.nextInt(1000));
            }
        } catch (InterruptedException e) {
        }
    }
}
```

Request クラス

Request クラス (List 8-3) は、仕事のリクエストを表すクラスです。

name フィールドはリクエストの依頼者の名前、number フィールドはリクエストの番号です。このサンプルプログラムでは、name の値は Alice, Bobby, Chris のいずれかになります。number の値は 0, 1, 2, ... になります。execute メソッドは、このリクエストの「処理」を記述しているメソッドです。処理といっても、実際にやっているのは実行し

ているスレッドの名前とリクエスト内容（依頼者名と番号）の表示だけです。「処理」に時間がかかることを表現するためにsleepしています。

List 8-3 仕事のリクエストを表している Request クラス (Request.java)

```
import java.util.Random;

public class Request {
    private final String name;        // 依頼者
    private final int number;        // リクエストの番号
    private static final Random random = new Random();
    public Request(String name, int number) {
        this.name = name;
        this.number = number;
    }
    public void execute() {
        System.out.println(Thread.currentThread().getName() + " executes " + this);
        try {
            Thread.sleep(random.nextInt(1000));
        } catch (InterruptedException e) {
        }
    }
    public String toString() {
        return "[ Request from " + name + " No." + number + " ]";
    }
}
```

Channel クラス

Channel クラス (List 8-4) は、仕事のリクエストの受け渡しとワーカースレッドの保持を行うためのクラスです。

Channel クラスは、仕事のリクエストの受け渡しのためにrequestQueue というフィールドを持っています。このフィールドはリクエストを保持しておくキューの動きをします。キューに入れるのは、putRequest メソッドで、キューから取り出すのは、takeRequest メソッドです。ここでは、Producer-Consumer パターン (第5章) が使われており、putRequest メソッドとtakeRequest を実現するためにGuarded Suspension パターン (第3章) が使われています。

Channel クラスは、ワーカースレッドの保持のためにthreadPool というフィールドを持っています。threadPool はWorkerThreadの配列です。コンストラクタの中では、このthreadPoolを初期化し、WorkerThreadのインスタンスを生成しています。配列の大

きさは、MAX_REQUEST で定めます。

ワーカースレッドにはWorker-0, Worker-1, Worker-2, ... という名前をつけています。
startWorkers メソッドは、ワーカースレッドをすべて起動するためのメソッドです。

List 8-4 仕事のリクエストの受け渡しと、ワーカースレッドの保持とを行う Channel クラス (Channel.java)

```
public class Channel {
    private static final int MAX_REQUEST = 100;
    private final Request[] requestQueue;
    private int tail;        // 次にputRequest する場所
    private int head;       // 次にtakeRequest する場所
    private int count;      // Request の数

    private final WorkerThread[] threadPool;

    public Channel(int threads) {
        this.requestQueue = new Request[MAX_REQUEST];
        this.head = 0;
        this.tail = 0;
        this.count = 0;

        threadPool = new WorkerThread[threads];
        for (int i = 0; i < threadPool.length; i++) {
            threadPool[i] = new WorkerThread("Worker-" + i, this);
        }
    }

    public void startWorkers() {
        for (int i = 0; i < threadPool.length; i++) {
            threadPool[i].start();
        }
    }

    public synchronized void putRequest(Request request) {
        while (count >= requestQueue.length) {
            try {
                wait();
            } catch (InterruptedException e) {
            }
        }
        requestQueue[tail] = request;
        tail = (tail + 1) % requestQueue.length;
        count++;
        notifyAll();
    }
}
```

```
public synchronized Request takeRequest() {
    while (count <= 0) {
        try {
            wait();
        } catch (InterruptedException e) {
        }
    }
    Request request = requestQueue[head];
    head = (head + 1) % requestQueue.length;
    count--;
    notifyAll();
    return request;
}
}
```

WorkerThreadクラス

WorkerThreadクラス (List 8-5) は、ワーカースレッドを表すクラスです。

ワーカースレッドは仕事を実行します。「仕事を実行する」という処理は、このサンプルプログラムでは以下の処理に対応しています。

- ・ Channelのインスタンスから takeRequestメソッドを使ってRequestのインスタンスを1個もらう
- ・ そのインスタンスのexecuteメソッドを呼び出す

ワーカースレッドは、一度起動すると永遠に仕事を実行し続けます。すなわち「新しいRequestのインスタンスを1つもらってはexecuteメソッドを呼び出す」という処理を繰り返すのです。

Thread-Per-Messageパターン (第7章) では、仕事を実行するたびに新しいスレッドを起動していました。しかしWorker Threadパターンでは、ワーカースレッドが繰り返し仕事を実行するので、新しいスレッドを起動する必要はありません。

WorkerThreadが持っているフィールドは、自分が仕事のリクエストを得るためのChannelのインスタンス (channel) だけです。WorkerThreadは、リクエストの具体的内容 (文字列を表示するという仕事) は何も知りません。WorkerThreadが知っているのは、「Requestクラスはexecuteメソッドを持っている」ということだけです。

List 8-5 ワーカースレッドを表す WorkerThread クラス (WorkerThread.java)

```

public class WorkerThread extends Thread {
    private final Channel channel;
    public WorkerThread(String name, Channel channel) {
        super(name);
        this.channel = channel;
    }
    public void run() {
        while (true) {
            Request request = channel.takeRequest();
            request.execute();
        }
    }
}

```

サンプルプログラムを実行すると、Fig.8-3のようになります。Worker-0 ~ Worker-4 という5人のWorkerThreadが、Alice, Bobby, Chris という3人のClientThreadからのリクエストを実行しているのがわかると思います。

リクエストを出しているClientThreadと、リクエストを実行しているWorkerThreadの間には固定的な関係はありません。AliceのリクエストNo.0はWorker-0が実行しましたが、同じAliceによる次のリクエストNo.1はWorker-3が実行し、さらに次のリクエストNo.2はWorker-2が実行しています。

ワーカースレッドは、そのリクエストを誰が発行したのかを考えず、受け取ったリクエストをただ実行するだけなのです。

Fig.8-3 実行例

Worker-0 executes [Request from Alice No.0]	Worker-0が、AliceのリクエストNo.0を実行
Worker-1 executes [Request from Bobby No.0]	Worker-1が、BobbyのリクエストNo.0を実行
Worker-2 executes [Request from Chris No.0]	Worker-2が、ChrisのリクエストNo.0を実行
Worker-3 executes [Request from Alice No.1]	Worker-3が、AliceのリクエストNo.1を実行
Worker-4 executes [Request from Bobby No.1]	Worker-4が、BobbyのリクエストNo.1を実行
Worker-1 executes [Request from Chris No.1]	Worker-1が、ChrisのリクエストNo.1を実行
Worker-1 executes [Request from Bobby No.2]	Worker-1が、BobbyのリクエストNo.2を実行
Worker-2 executes [Request from Alice No.2]	Worker-2が、AliceのリクエストNo.2を実行

(CTRL + Cで終了)

Worker Threadパターンの登場人物

Worker Threadパターンの登場人物は、次のとおりです。

- ◆ クライアント Client (依頼者) 役
Client 役は、仕事のリクエストを Request 役として作成し、Channel 役に渡します。サンプルプログラムでは、ClientThread クラスがこの役をつとめました。
- ◆ チャンネル Channel (通信路) 役
Channel 役は、Client 役から Request 役を受け取り、Worker 役に渡します。サンプルプログラムでは、Channel クラスがこの役をつとめました。
- ◆ ワーカー Worker (作業員) 役
Worker 役は Channel 役から Request 役をもらい、その仕事を実行します。仕事が終わったら、次の Request 役をもらいにいきます。サンプルプログラムでは、WorkerThread クラスがこの役をつとめました。
- ◆ リクエスト Request (依頼) 役
Request 役は、仕事を表すためのものです。Request 役は、その仕事を実行するのに必要な情報を保持しています。サンプルプログラムでは、Request クラスがこの役をつとめました。

Worker Threadパターンのクラス図をFig.8-4に、タイムスレッド図をFig.8-5に示します。

あなたの考えを広げるためのヒント

スレッド起動は重い処理

自分が抱えている仕事を人に任せることができたら、自分は次の仕事をすることができます。スレッドも同じです。他のスレッドに仕事を任せることができたら、自分は次の仕事に進むことができます。これがThread-Per-Messageパターン(第7章)のテーマでした。

けれども、スレッドを新しく起動するというのは時間がかかる処理です。ですから、Worker Threadパターンではスレッドを使いまわし、リサイクルすることがテーマのひとつになっています。

Fig.8-4 Worker Thread パターンのクラス図

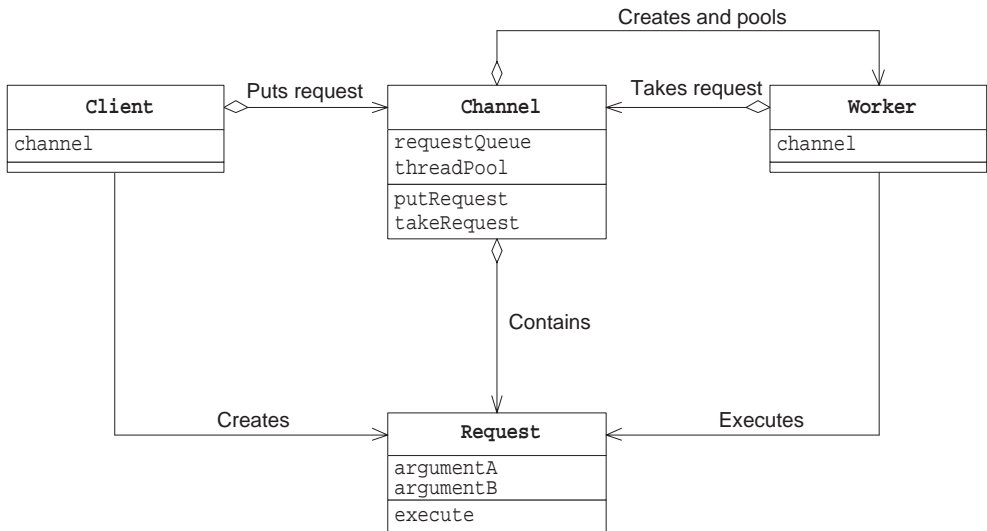


Fig.8-5 Worker Thread パターンのタイムスレッド図

