Hochschule für Technik und Wirtschaft Dresden

Berechnung der Mandelbrotmenge
mit Implementierung von Java-RMI

Development Document

David Kirchner, Maxim Haschke, Tan Minh Ho, Quang Duy Pham

26. Juni 2022

Inhaltsverzeichnis

1 Kurzbeschreibung an Belegarbeit

Um das Modul Programmierung verteilter Systeme zu erfassen, wird eine Gruppenarbeit zur Entwicklung eines verteilten Programms zur Anzeige der Mandelbrotmenge belegt:

- schrittweiser Zoom zu einem bestimmten Bildpunkt
- Nutzung von Java-RMI oder einer anderen geeigneten Technologie zur Verteilung der Aufgaben
- Architektur Model-View-Presenter für den Client
- Dokumentation der Programmstruktur

2 Architektur

Nach der Diskussion mit anderen Gruppenteilnehmer wird das Programm mit folgenden Merkmalen realisiert:

- Übertragung mittels TCP
- Nutzung von 2-dimensionalem Array zur Übertragung der Pixelfarbe
- Thread-Pool im Client zum Anzeigen des berechneten Bildes
- Verwendung von Threads im Server zur Berechnung der Mandelbrotmenge

3 Details der Programme

3.1 Client

Fuer den Client gibt es drei Java-Datein mit folgenden Bezeichnungen:

- MandelClient.java
- MandelClientImpl.java
- MandelClientMain.java

3.1.1 MandelClient.java

Die Datei „MandelClient.java“definiert die Schnittstelle fuer das Clientprogramm. Die Methoden setRGB, sendTasks und setZoomDestination sind hier als Remote mit RemoteException deklariert.

```
1  import java.rmi.Remote;
2  import java.rmi.RemoteException;
3  import java.util.TimerTask;
4
5  public interface MandelClient extends Remote {
6      public void setRGB(int[][] bild) throws RemoteException;
7
8      public void sendTasks(MandelServer server) throws RemoteException;
9
10     public void setZoomDestination(MandelServer server, double left, double top,
       double zoom, int animationSteps) throws RemoteException;
11
12     // public String getName() throws RemoteException;
13
14 }
```

3.1.2 MandelClientImpl.java

Die Datei „MandelClientImpl.java“implementiert oben genannte Methoden fuer den Client.

1. Konstruktor setzt die notwendigen Bildwerte zum Server und setzt JFrame und ActionListener fest.

```
    private JFrame frame;
    public int yStart, yStop, xStart, xStop;
    private ExecutorService pool;
    private int iter = 0;

    private Timer timer;

    public MandelClientImpl(MandelServer server) throws RemoteException {
        pool = Executors.newCachedThreadPool();
        image = new BufferedImage(WIDTH, HEIGHT, BufferedImage.TYPE_INT_RGB);
        // Client sends image's width, height and detail to server. This
        // is necessary to set the size of int[][] bild, which contains
        // the color information of the mandelbrot image and for the
        // calculation of RGB Values.
        server.setDetail(WIDTH, HEIGHT, DETAIL);
        frame = new JFrame("Mandelbrot");
        frame.setLocationRelativeTo(null);
        JLabel label = new JLabel(new ImageIcon(image));
        JButton button = new JButton("Start");
        tfx = new JTextField("-0.75");
        tfy = new JTextField("0");
        label.addMouseListener(new MouseAdapter() {
            // When client clicks on a point, this will
            // calculate the value of top, left and zoom therefore
            // client needs to send tasks to server.
            @Override
            public void mouseReleased(MouseEvent event) {
                if (event.getButton() == MouseEvent.BUTTON1) {
                    left = (event.getX() - WIDTH / 4.0) * zoom + left;
                    top = (event.getY() - HEIGHT / 4.0) * zoom + top;
                    zoom = zoom / 2.0;
                } else {
                    left = (event.getX() - WIDTH) * zoom + left;
                    top = (event.getY() - HEIGHT) * zoom + top;
                    zoom = zoom * 2.0;
                }
                try {
                    sendTasks(server);
                } catch (RemoteException re) {
                    re.printStackTrace();
                }
            }
        });
        button.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e) {
                try {
                    setZoomDestination(server, Double.parseDouble(tfx.getText()),
    Double.parseDouble(tfy.getText()),
                            zoom, 50);
                } catch (RemoteException re) {
```

2. sendTasks schickt Request zum Server und wartet auf Antwort

   (a) Senden der Koordinaten und Werte von ActionListener zum Server

   (b) Server startet Berechnung der Mandelbrotmenge mit Zoom zu den gegebenen Koordinaten

   (c) Client wartet auf Server

   (d) Client zeigt die zurueckgegeben RGB-Werte von Server an

```
        });
        frame.setLayout(new FlowLayout());
        frame.add(label);
        frame.add(tfx);
        frame.add(tfy);
        frame.add(button);
        frame.pack();
        frame.setDefaultCloseOperation(JFrame.DO_NOTHING_ON_CLOSE);
        frame.addWindowListener(new WindowAdapter() {
            @Override
            public void windowClosing(WindowEvent e) {
                if (JOptionPane.showConfirmDialog(frame, "Are you sure?") ==
    JOptionPane.OK_OPTION) {
                    pool.shutdown();
                    frame.dispose();
                    System.exit(0);
```

3. setRGB setzt Pixelfarbe des Bildes mit Nutzung eines Thread-Pools.

```
                    zoom += zoomSteps;

                    try {
                        sendTasks(server);
                    } catch (RemoteException re) {
                        re.printStackTrace();
                    }

                    counter++;
                } else {
                    timer.cancel();
```

3.1.3 MandelClientMain.java

Die Datei „MandelClientMain.java“sucht den Server mit bestimmtem Namen (welcher durch args[]-Variable gesetzt wurde) und schickt dann Request zum Server.

```
1  import java.rmi.Naming;
2  import java.rmi.NotBoundException;
3  import java.rmi.RemoteException;
4
5  public class MandelClientMain {
6      public static void main(String[] args) {
7          if (args.length != 2) {
8              System.out.println("Server agrs: IP/Hostname Servername");
9              return;
10         }
11         try {
12             MandelServer server = (MandelServer) Naming.lookup("rmi://" + args[0] + "/
       " + args[1]);
13             MandelClientImpl client = new MandelClientImpl(server);
14             client.sendTasks(server);
15         } catch (RemoteException re) {
16             System.out.println("Registry " + args[0] + " could not be found.");
17             re.printStackTrace();
18         } catch (NotBoundException nbe) {
19             System.out.println("Name " + args[1] + " is not currently bound");
20             nbe.printStackTrace();
21         } catch (Exception e) {
22             e.printStackTrace();
23         }
24     }
25 }
```

3.2 Server

Fuer den Client gibt es drei Javadateien mit Name:

- MandelServer.java
- MandelServerImpl.java
- MandelServerMain.java

3.2.1 MandelServer.java

Die Datei „MandelServer.java“definiert die Schnittstelle fuer das Serverprogramm. Die Methoden setDetail, setImageProperties, calculateRGB, returnColor, startCalculatingRGB und isFinish sind hier als Remote mit RemoteException deklariert.

```
1  import java.rmi.Remote;
2  import java.rmi.RemoteException;
3
4  public interface MandelServer extends Remote {
5      // public boolean addClient(MandelClient client) throws RemoteException;
6
7      public void setDetail(int width, int height, int detail) throws RemoteException;
8
9      public void setImageProperties(double top, double left, double zoom) throws
       RemoteException;
10
11     public void calculateRGB(int yStart, int yStop, int xStart, int xStop) throws
       RemoteException;
12
13     public int[][] returnColor() throws RemoteException;
14
15     public void startCalculatingRGB() throws RemoteException;
16
17     public boolean isFinish() throws RemoteException;
18
19     public void initSlavServers() throws RemoteException;
20 }
```

3.2.2 MandelServerImpl.java

1. Konstruktor definiert Thread-Pool fuer Parallelität des Servers

```
    private ThreadPoolExecutor pool;

    public MandelServerImpl() throws RemoteException {
        // Unbounded queues. Using an unbounded queue (for example a
    LinkedBlockingQueue
        // without a predefined capacity) will cause new tasks to wait in the
    queue when
        // all corePoolSize threads are busy. Thus, no more than corePoolSize
    threads
        // will ever be created. (And the value of the maximumPoolSize therefore
    doesn't
        // have any effect.) This may be appropriate when each task is completely
        // independent of others, so tasks cannot affect each others execution;
    for
        // example, in a web page server. While this style of queuing can be
    useful in
        // smoothing out transient bursts of requests, it admits the possibility
    of
        // unbounded work queue growth when commands continue to arrive on average
        // faster than they can be processed.
        //
        // LinkedBlockingQueue is a FIFO queue, new elements are inserted at the
        // tail of the queue. Any attempt to retrieve a task out of the queue,
        // can be seen safe as it will not return empty.
        // allClients = new ArrayList<MandelClient>();
        pool = new ThreadPoolExecutor(NUMBER_OF_THREADS, NUMBER_OF_THREADS, 0L,
    TimeUnit.SECONDS,
```

2. setDetails definiert die Laenge, Breite und Details (Iterationen) des Bildern und deklariert ein 2-dimensionales Array mit passender Laenge und Breite.

```
    // return true;

    // }

    public void setDetail(int width, int height, int detail) throws
    RemoteException {
        this.detail = detail;
```

3. setImageProperties setzt die Koordinaten von MouseActionListener.

```
        this.height = height;
        this.bild = new int[width][height];
    }

    public void setImageProperties(double top, double left, double zoom) throws
    RemoteException {
```

4. returnColor schickt ein 2-dimensionales Array zum Client zurueck

```
        this.left = left;
        this.zoom = zoom;
    }
```

5. startCalculateingRBG schnitt das Bild zu kleineren Bilden mit Hilfe der Nummer des Threads im Thread-Pool. Falls alle Threads "busyßind, wartet Segment des Bildes

```
    public synchronized int[][] returnColor() throws RemoteException {
        return this.bild;
    }

    public void startCalculatingRGB() throws RemoteException {
        // A for-loop but using IntStream (Java 8)
        // for(int i = 0; i < MandelClientImpl.HEIGHT ; i += 10){}
        // Client divides the entire Mandelbrot image into multiple
        // smaller images (yStart, yStop, xStart, xStop).
        // Client sends the smaller image as a task to server.
        // Server uses ThreadPool with a fixed number of threads,
        // When all threads are busy, new task will be queued.
        //
        // We will also handle the distributed system hier.
        IntStream.iterate(0, i -> i + 10).limit(height / 10).parallel().forEach((i
    ) -> {
            try {
                int j = i + 10;
                calculateRGB(i, j, 0, width);
            } catch (RemoteException re) {
```

6. isFinish beanwortet, ob Bild fertig bearbeitet hat

```
            }
        });
    }

    public synchronized boolean isFinish() throws RemoteException {
        if (pool.getActiveCount() == 0) {
            return false;
        }
        return true;
```

7. calculateRGB stellt Aufgabe zum Thread fest, um parallele Berechnung zu erreichen

```

    public void calculateRGB(int yStart, int yStop, int xStart, int xStop) throws
    RemoteException {
        Task task = new Task(yStart, yStop, xStart, xStop);
        pool.execute(task);
```

8. Task berechnet die Mandelbrotmenge

```
    class Task implements Runnable {
        private int yStart;
        private int yStop;
        private int xStart;
        private int xStop;

        public Task(int yStart, int yStop, int xStart, int xStop) {
            this.yStart = yStart;
            this.yStop = yStop;
            this.xStart = xStart;
            this.xStop = xStop;
        }

        public void run() {
            for (int y = yStart; y < yStop; y++) {
                double ci = y * zoom + top;
                for (int x = xStart; x < xStop; x++) {
                    double cr = x * zoom + left;
                    double zr = 0.0;
                    double zi = 0.0;
                    int color = 0x000000;
                    for (int i = 0; i < detail; i++) {
                        double zrzr = zr * zr;
                        double zizi = zi * zi;
                        if (zrzr + zizi >= 4) {
                            color = PALETTE[i & 15];
                            break;
                        }
                        zi = 2.0 * zr * zi + ci;
                        zr = zrzr - zizi + cr;
                    }
                    bild[x][y] = color;
                }
            }
        }
    }
```

3.2.3 MandelServerMain.java

Die Datei „MandelServerMain.java“stellt ein Registry fest und verbindet mit Serverschnittstelle

```
import java.rmi.registry.LocateRegistry;
import java.rmi.registry.Registry;

public class MandelServerMain {
    public static void main(String[] args) {
        if (args.length != 1) {
            System.out.println("Args: Server");
            return;
        }
        try {
            MandelServerImpl server = new MandelServerImpl();
            Registry registry = LocateRegistry.createRegistry(1099);
            registry.rebind(args[0], server);
            System.out.println("Server is ready.");
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

4 Link und Literatur

- Parallele und verteilte Anwendungen in Java, Rainer Oechsle
- Berechnung der Mandelbrotmenge: https://www.youtube.com/watch?v=0bgrzmtu4e8