

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	2
2	Architektur	2
3	Details der Programme	2
3.1	Client	2
3.1.1	MandelClient.java	2
3.1.2	MandelClientImpl.java	3
3.1.3	MandelClientMain.java	5
3.2	Server	6
3.2.1	MandelServer.java	6
3.2.2	MandelServerImpl.java	7
3.2.3	MandelServerMain.java	9
4	Link und Literatur	10

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-Dateien 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,
11    double zoom, int animationSteps) throws RemoteException;
12
13    // public String getName() throws RemoteException;
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.

```
1  public MandelClientImpl(MandelServer server) throws RemoteException {
2      pool = Executors.newCachedThreadPool();
3      image = new BufferedImage(WIDTH, HEIGHT, BufferedImage.TYPE_INT_RGB);
4      // Client sends image's width, height and detail to server. This
5      // is necessary to set the size of int[][] bild, which contains
6      // the color information of the mandelbrot image and for the
7      // calculation of RGB Values.
8      server.setDetail(WIDTH, HEIGHT, DETAIL);
9      frame = new JFrame("Mandelbrot");
10     frame.setLocationRelativeTo(null);
11     JLabel label = new JLabel(new ImageIcon(image));
12     JButton button = new JButton("Start");
13     tfx = new JTextField("-0.75");
14     tfy = new JTextField("0");
15     label.addMouseListener(new MouseAdapter() {
16         // When client clicks on a point, this will
17         // calculate the value of top, left and zoom therefore
18         // client needs to send tasks to server.
19         @Override
20         public void mouseReleased(MouseEvent event) {
21             if (event.getButton() == MouseEvent.BUTTON1) {
22                 left = (event.getX() - WIDTH / 4.0) * zoom + left;
23                 top = (event.getY() - HEIGHT / 4.0) * zoom + top;
24                 zoom = zoom / 2.0;
25             } else {
26                 left = (event.getX() - WIDTH) * zoom + left;
27                 top = (event.getY() - HEIGHT) * zoom + top;
28                 zoom = zoom * 2.0;
29             }
30             try {
31                 sendTasks(server);
32             } catch (RemoteException re) {
33                 re.printStackTrace();
34             }
35         }
36     });
37     button.addActionListener(new ActionListener() {
38         public void actionPerformed(ActionEvent e) {
39             try {
40                 setZoomDestination(server, Double.parseDouble(tfx.getText()),
41 Double.parseDouble(tfy.getText()),
42                 zoom, 50);
43             } catch (RemoteException re) {
44                 re.printStackTrace();
45             }
46         }
47     });
48     frame.setLayout(new FlowLayout());
49     frame.add(label);
50     frame.add(tfx);
51     frame.add(tfy);
52     frame.add(button);
53     frame.pack();
54     frame.setDefaultCloseOperation(JFrame.DO_NOTHING_ON_CLOSE);
55     frame.addWindowListener(new WindowAdapter() {
56         @Override
```

```

56         public void windowClosing(WindowEvent e) {
57             if (JOptionPane.showConfirmDialog(frame, "Are you sure?") ==
JOptionPane.OK_OPTION) {
58                 pool.shutdown();
59                 frame.dispose();
60                 System.exit(0);
61             }
62         }
63     });
64     frame.setVisible(true);
65 }

```

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

```

1     public void sendTasks(MandelServer server) throws RemoteException {
2         // We send the server the coord and the zoom values.
3         server.setImageProperties(top, left, zoom);
4         // Now let the master server do it job.
5         server.startCalculatingRGB();
6         // Let's wait for the result.
7         while (server.isFinish())
8             ;
9         // Server sends the result and client passes the result to ThreadPool
10        // Threads read the array and set the RGB
11        setRGB(server.returnColor());
12        System.out.println("Iteration: " + iter + "| left: " + left + "| top:" +
top + "| zoom: " + zoom);
13        iter++;
14    }

```

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

```

1     public void setRGB(int[][] bild) throws RemoteException {
2         IntStream.range(0, bild.length).forEach((i) -> {
3             Runnable task = () -> {
4                 for (int j = 0; j < bild[i].length; j++) {
5                     image.setRGB(i, j, bild[i][j]);
6                 }
7             };
8             pool.execute(task);
9         });
10        frame.repaint();
11    }

```

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] + "/"
13            " + args[1]);
14            MandelClientImpl client = new MandelClientImpl(server);
15            client.sendTasks(server);
16        } catch (RemoteException re) {
17            System.out.println("Registry " + args[0] + " could not be found.");
18            re.printStackTrace();
19        } catch (NotBoundException nbe) {
20            System.out.println("Name " + args[1] + " is not currently bound");
21            nbe.printStackTrace();
22        } catch (Exception e) {
23            e.printStackTrace();
24        }
25    }
```

3.2 Server

Fuer den Client gibt es drei Javadeiten 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  
10    RemoteException;  
11  
12    public void calculateRGB(int yStart, int yStop, int xStart, int xStop) throws  
13    RemoteException;  
14  
15    public int[][] returnColor() throws RemoteException;  
16  
17    public void startCalculatingRGB() throws RemoteException;  
18  
19    public boolean isFinish() throws RemoteException;  
20  
21    public void initSlavServers() throws RemoteException;  
22 }
```

3.2.2 MandelServerImpl.java

1. Konstruktor definiert Thread-Pool fuer Parallelität des Servers

```
1 public MandelServerImpl() throws RemoteException {
2     // Unbounded queues. Using an unbounded queue (for example a
3     // LinkedBlockingQueue
4     // without a predefined capacity) will cause new tasks to wait in the
5     // queue when
6     // all corePoolSize threads are busy. Thus, no more than corePoolSize
7     // threads
8     // will ever be created. (And the value of the maximumPoolSize therefore
9     // doesn't
10    // have any effect.) This may be appropriate when each task is completely
11    // independent of others, so tasks cannot affect each others execution;
12    for
13    // example, in a web page server. While this style of queuing can be
14    // useful in
15    // smoothing out transient bursts of requests, it admits the possibility
16    // of
17    // unbounded work queue growth when commands continue to arrive on average
18    // faster than they can be processed.
19    //
20    // LinkedBlockingQueue is a FIFO queue, new elements are inserted at the
21    // tail of the queue. Any attempt to retrieve a task out of the queue,
22    // can be seen safe as it will not return empty.
23    // allClients = new ArrayList<MandelClient>();
24    pool = new ThreadPoolExecutor(NUMBER_OF_THREADS, NUMBER_OF_THREADS, 0L,
25    TimeUnit.SECONDS,
26    new LinkedBlockingQueue<>(), new ThreadPoolExecutor.DiscardPolicy
27    ());
28 }
```

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

```
1 public void setDetail(int width, int height, int detail) throws
2 RemoteException {
3     this.detail = detail;
4     this.width = width;
5     this.height = height;
6     this.bild = new int[width][height];
7 }
```

3. setImageProperties setzt die Koordinaten von ActionListener.

```
1 public void setImageProperties(double top, double left, double zoom) throws
2 RemoteException {
3     this.top = top;
4     this.left = left;
5     this.zoom = zoom;
6 }
```

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

```
1 public synchronized int[][] returnColor() throws RemoteException {
2     return this.bild;
3 }
```


5. startCalculatingRGB schnitt das Bild zu kleineren Bildern mit Hilfe der Nummer des Threads im Thread-Pool. Falls alle Threads "busy" sind, wartet Segment des Bildes

```
1 public void startCalculatingRGB() throws RemoteException {
2     // A for-loop but using IntStream (Java 8)
3     // for(int i = 0; i < MandelClientImpl.HEIGHT ; i += 10){}
4     // Client divides the entire Mandelbrot image into multiple
5     // smaller images (yStart, yStop, xStart, xStop).
6     // Client sends the smaller image as a task to server.
7     // Server uses ThreadPool with a fixed number of threads,
8     // When all threads are busy, new task will be queued.
9     //
10    // We will also handle the distributed system hier.
11    IntStream.iterate(0, i -> i + 10).limit(height / 10).parallel().forEach((i
12    ) -> {
13        try {
14            int j = i + 10;
15            calculateRGB(i, j, 0, width);
16        } catch (RemoteException re) {
17            re.printStackTrace();
18        }
19    });
20 }
```

6. isFinish beantwortet, ob Bild fertig bearbeitet hat

```
1 public synchronized boolean isFinish() throws RemoteException {
2     if (pool.getActiveCount() == 0) {
3         return false;
4     }
5     return true;
6 }
```

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

```
1 public void calculateRGB(int yStart, int yStop, int xStart, int xStop) throws
2 RemoteException {
3     Task task = new Task(yStart, yStop, xStart, xStop);
4     pool.execute(task);
5 }
```

8. Task berechnet die Mandelbrotmenge

```
1  class Task implements Runnable {
2      private int yStart;
3      private int yStop;
4      private int xStart;
5      private int xStop;
6
7      public Task(int yStart, int yStop, int xStart, int xStop) {
8          this.yStart = yStart;
9          this.yStop = yStop;
10         this.xStart = xStart;
11         this.xStop = xStop;
12     }
13
14     public void run() {
15         for (int y = yStart; y < yStop; y++) {
16             double ci = y * zoom + top;
17             for (int x = xStart; x < xStop; x++) {
18                 double cr = x * zoom + left;
19                 double zr = 0.0;
20                 double zi = 0.0;
21                 int color = 0x000000;
22                 for (int i = 0; i < detail; i++) {
23                     double zr zr = zr * zr;
24                     double zizi = zi * zi;
25                     if (zr zr + zizi >= 4) {
26                         color = PALETTE[i & 15];
27                         break;
28                     }
29                     zi = 2.0 * zr * zi + ci;
30                     zr = zr zr - zizi + cr;
31                 }
32                 bild[x][y] = color;
33             }
34         }
35     }
36 }
```

3.2.3 MandelServerMain.java

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

```
1  import java.rmi.registry.LocateRegistry;
2  import java.rmi.registry.Registry;
3
4  public class MandelServerMain {
5      public static void main(String[] args) {
6          if (args.length != 1) {
7              System.out.println("Args: Server");
8              return;
9          }
10         try {
11             MandelServerImpl server = new MandelServerImpl();
12             Registry registry = LocateRegistry.createRegistry(1099);
13             registry.rebind(args[0], server);
14             System.out.println("Server is ready.");
15         } catch (Exception e) {
16             e.printStackTrace();
17         }
18     }
19 }
```

4 Link und Literatur

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