Verteilte Systeme und Kommunikationsnetze

Testumgebung

Praktikum 1

Fachhochschule Bielefeld
Campus Minden
Studiengang Informatik

Beteiligte Personen:

Name	Matrikelnummer
Christian Krebel	1151165
Dennis Petana	1157886
Hannes Rüffer	1151954

Aufgaben:

Aufgabe	Gelöst
Aufgabe 1	a
Aufgabe 2	a, b, c

---

14. November 2018

Inhaltsverzeichnis

Bearbeitung der Aufgaben - E-Mail

1 Aufgabe - eMail

Aufgabenstellung

Wie funktioniert das SMTP Protokoll? (1 Punkt)

Vorbereitung

Für die Recherche wurde folgende Quelle verwendet: [1]. Außerdem floß das eigene Wissen mit ein.

Durchführung

SMTP bedeutet /Simple Mail Transfer Protocol|, ist ein TCP/IP Protokoll und wird dafür verwendet zwischen Servern E-Mails zu senden und zu erhalten.

Da es jedoch in seiner Fähigkeit beschränkt ist, Nachrichten auf der empfangenden Seite zu verarbeiten, wird es üblicherweise mit einem zweiten Protokoll wie POP3 oder IMAP genutzt (Zu diesen Protokollen später mehr).

Ein Mail Client verbindet sich zu dem Beispiel-SMTP-Server auf *mail.example.de* und benutzt den Port 25.
Nun führt das Mail Programm eine Konversation mit dem SMTP-Server durch und teilt dem SMTP-Server die Adresse des Absenders und die des Empfängers sowie den Nachrichtentext mit.
Der SMTP-Server nimmt die Beispiel-Empfänger-Adresse /foo@maildomain.com| und teilt sie in zwei Teile auf.
Einmal in den Namen des Empfängers /foo| und den Domain-Namen /maildomain|.
Da der Empfänger eine andere Domain besitzt, muss der SMTP-Server mit der fremden Domain kommunizieren und spricht deshalb den DNS-Server an.
Der eigene SMTP Server fragt nach der IP Adresse des SMTP-Server von /maildomain.com|.
Der DNS-Server antwortet mit der gewünschten IP Adresse. Nun verbindet sich der eigene SMTP Server mit dem von /maildomain.com| mit dem Port 25 und sendet diesen die Nachricht des Benutzers, danach wird die Verbindung getrennt.
Wenn aus irgendwelchen Gründe sich der eigene SMTP Server nicht mit dem des Empfängers verbinden kann, wird die Nachricht in eine Warteschlange geschoben.

In den einfachsten Implementierungen von POP3 verwaltet der Server eine Sammlung von Textdateien. Eine für jedes E-Mail-Konto. Wenn eine Nachricht eingeht, hängt der POP3-Server sie einfach am Ende der Empfängerdatei an.

Wenn man seine E-Mails abrufen, stellt der eigene E-Mail-Client über Port 110 eine Verbindung zum POP3-Server her.
Der POP3-Server erfordert einen Kontonamen und ein Kennwort. Sobald man sich angemeldet hat, öffnet der POP3-Server die Textdatei und ermöglicht den Zugriff darauf. Der E-Mail-Client stellt eine Verbindung zum POP3-Server her und gibt eine Reihe von Befehlen aus, um Kopien der E-Mail-Nachrichten auf dem lokalen Computer zu bringen. In der Regel werden die Nachrichten dann vom Server gelöscht.

IMAP (Internet Mail Access Protocol) ist ein fortgeschrittenes Protokoll, zur Verwaltung von E-Mail Nachrichten und nutzt den Port 143. Der größte Vorteil von IMAP gegenüber POP3 ist die Möglichkeit seine Nachrichten auf dem Server zu belassen.
Der Hauptgrund dafür ist, dass die E-Mails auf dem Server verbleiben, und Benutzer von verschiedenen Computern aus eine Verbindung herstellen können. So sind die E-Mails überall aus zugreifbar.
Sobald man eine E-Mail heruntergeladen hat, bleibt sie bei POP3 auf dem Rechner hängen, auf dem sie heruntergeladen wurde.
Man kann seine Nachrichten in Ordnern verwalten die dann auch auf dem Server verbleiben.

Fazit

Wie man sehen kann ist der POP3 Server eine sehr simple Schnittstelle zwischen dem E-Mail Client und der Textdatei in der die eigenen Nachrichten liegen.
Es ist wichtig sich den Unterschied zwischen POP3 und IMAP vor Augen zu führen um nicht ausversehen seine E-Mails zu `löschen`. Ein Problem kann auftreten wenn die E-Mails auf dem entfernten Server liegen und man nur über eine Internetverbindung darauf zugreifen kann. Aber dafür haben E-Mail Clients vorgesorgt und bieten die Möglichkeit an, E-Mails auf dem Computer zu cachen. Dadurch besitzt man eine Kopie auf der lokalen Maschine. Bei der Recherche traten keine Probleme auf.

2 Aufgabe - eMails versenden mit Javamail

2.1 Punkt a

Aufgabenstellung

Verwenden Sie als SMTP Ausgangsserver einen Ihrer eigenen Mailaccounts, z.B. den der FH Bielefeld. Die Konfi

guration kennen Sie aus den Hilfeseiten der DVZ. Das Programm soll in der Lage sein, eine Mail mit einem von Ihnen vorgegebenen Inhalt an eine beliebige Mailadresse zu senden. Der Inhalt der Mail soll aus einer Datei inhalt.txt aus dem Dateisystem Ihres Rechners eingelesen werden. (5 Punkte)

Vorbereitung

Zunächst muss man wissen, wie man den FH-Bielefeld SMTP-Server erreicht. Dazu steht auf der E-Mail Info-Seite der FH-Bielefeld[1], dass der SMTP Server zu erreichen ist unter:

- Hostname: smtp.fh-bielefeld.de
- Port: 587 (Standard für STARTTLS)

Anschließend benötigen wir Informationen dazu, wie man die Javamail API mit TLS nutzen kann. Dazu wenden wir uns an den JavaMail Guide von Pankaj [2].Asdasd

Durchführung

Damit man eine E-Mail über den SMTP-Server der FH schicken kann, muss zuerst ein *Authenticator* eingerichtet werden. Dazu erstellt man ein neues *Properties* Objekt, setzt die Werte Host und Port, sowie die Flags *auth* und *starttls.enable*. Anschließend erstellt man den *Authenticator* und übergibt ihn an die Session:

```
Authenticator auth = new Authenticator() {
protected PasswordAuthentication getPasswordAuthentication() {
return new PasswordAuthentication(fromEmail, password);
}
};
Session session = Session.getInstance(props, auth);
```

Nun wird eine *MimeMessage* mit Hilfe der Session erstellt. In diese *MimeMessage* schreibt man alle Daten, die man versenden möchte:

- Header (Content-Type, Mime-Type, Charset)
- Charset
- Sender Adresse, Name und Antwortadresse
- Betreff
- Empfänger (Können auch mehrere sein, getrennt mit einem Komma)
- Inhalt der Mail (Text, Bilder, Anhänge)

Abschließend wird die E-Mail mit der *send*-Methode aus der Klasse *Transport* versendet. Insgesamt sieht das Erstellen und Versenden der Nachricht so aus:

```
MimeMessage msg = new MimeMessage(session);

msg.addHeader("Content-type", "text/HTML; charset=UTF-8");
msg.addHeader("format", "flowed");
msg.addHeader("Content-Transfer-Encoding", "8bit");
```

1 https://www.fh-bielefeld.de/dvz/it-services/e-mail

```
msg.setFrom(new InternetAddress("hannes.rueffer@fh-bielelf.de", "Hannes Rüffer"));
msg.setReplyTo(InternetAddress.parse("hannes.rueffer@fh-bielefeld.de", false));

msg.setSubject(subject, "UTF-8");
msg.setText(body, "UTF-8");
msg.setSentDate(new Date());


msg.setRecipients(Message.RecipientType.TO, InternetAddress.parse(toEmail, false));
Transport.send(msg);
```

In der `main`-Methode der Hauptklasse `Mailsystem` werden mithilfe der Klasse `FileReader` die nötigen txt.Dateien ausgelesen:

```
1 FileReader fr = new FileReader();
2         ArrayList<String> empflist; Empfngerliste
3         try{
4             empflist = fr.readFileByLines(empf);
5         } catch (FileNotFoundException exc) {
6             System.out.println("Datei " + empf + " nicht gefunden!");
7             return;
8 }
```

`FileReader` besitzt die zwei Methoden `readEntireFile` und `readFileByLines`, welche ähnlich aufgebaut sind. Sie finden den Pfad der zu benutzenden Datei, scannen sie (benutzt den `Java.util.Scanner`) und geben das Gewünschte als String bzw. ArrayList mit dem Typen String zurück. Dabei darf man die Exceptions nicht vergessen, denn diese könnten auftreten, wenn z.B. keine Datei gefunden wird.

Fazit

Das finden von Dateien mithilfe von relativen Pfaden ist auf verschiedene Arten umsetzbar in Java und kann auch für verwirrung sorgen. Jedoch hat dies nach kurzem Ausprobieren keine Probleme mehr bereitet.

2.2 Punkt b

Aufgabenstellung

Erweitern Sie ihr Programm so, dass automatich eine Liste von Empf¨angern aus der Datei empfaenger.txt eingelesen wird. (1 Punkt)

Vorbereitung

Bevor man eine Bibliothek nutzt, sollte man sie sich angucken. Dies haben wir auch mit javax.mail getan: https://docs.oracle.com/javaee/7/api/javax/mail/package-summary.html

Durchführung

Möchte man dann die Empfänger übergeben (siehe vorherige Aufgabe), kann man die ArrayListe der eingescannten Empfänger in einen String umwandeln, denn laut Dokumentation soll eine mit Komma separierte Liste innerhalb eines Strings verwendet werden.

```
1 String addresses = "";
2 for(String mail : toEmail){
3 addresses += mail+",";
4 }
```

Das Einlesen an sich wurde bereits in der vorherigen Aufgabe geschildert.

Fazit

Das finden von Dateien mithilfe von relativen Pfaden ist auf verschiedene Arten umsetzbar in Java und kann auch für verwirrung sorgen. Jedoch hat dies nach kurzem Ausprobieren keine Probleme mehr bereitet.

2.3 Punkt c

Aufgabenstellung

Fügen Sie ihrer Mail einen Anhang hinzu (zum Beispiel ein Bild oder PDF-Dokument). Erläutern Sie, in welcher Form der Anhang übertragen wird.

Vorbereitung

Um Anhänge in einer E-Mail zu versenden, muss man wissen, wie der Inhalt einer E-Mail mit Javamail funktioniert. Dazu haben wir die offizielle Dokumentation von Javamail herangezogen [3]:

- Der Inhalt einer Mail kann ein *Multipart* sein, welcher das rekursive Hinzufügen anderer *Parts* erlaubt, wie z.B. *BodyPart*.
- *BodyParts* enthalten den Inhalt, der in der Mail enthalten ist, also jeglicher Text, Bilder oder andere Dateien
- Einem *MimeBodyPart* (die Implementierung der abstrakten Klasse *BodyPart*) kann dann eine *DataSource* mitgegeben werden, indem man den *DataHandler* des *MimeBodyParts* mit der *DataSource* setzt
- Zuletzt wird der *MimeBodyPart* dem *Multipart* hinzugefügt

So erhält man einen *MultiPart*, welcher dann der Nachricht hinzugefügt wird.

Durchführung

Dort wo in der *sendEmailWithAttachment* Methode vorher der E-Mail-Körper wie folgt gesetzt wurde,

```
msg.setSubject(subject, "UTF-8");
msg.setText(body, "UTF-8");
msg.setSentDate(new Date());
```

wird jetzt zusätzlich erst der *MultiPart* erstellt:

```
msg.setSubject(subject, "UTF-8");
msg.setSentDate(new Date());

BodyPart messageBodyPart = new MimeBodyPart();
messageBodyPart.setText(body);

Multipart multipart = new MimeMultipart();
multipart.addBodyPart(messageBodyPart);

messageBodyPart = new MimeBodyPart();
DataSource source = new FileDataSource(filename);
messageBodyPart.setDataHandler(new DataHandler(source));
messageBodyPart.setFileName(filename);
multipart.addBodyPart(messageBodyPart);

msg.setContent(multipart);
```

Man übergibt anschließend an die Methode den Dateinamen (Parameter *filename*), der angehängt werden soll.

Fazit

Beim Anhängen von Dateien muss man darauf achten, dass die Dateien vom System gefunden werden. Das System sucht in dem Order, von dem aus man das Programm gestartet hat, nach den Dateien, sollten sie nicht gefunden werden, wird auch keine E-Mail verschickt.

Literatur

[1] M. Brain und T. Crosby. (). How E-mail Works, Adresse: `https://computer.howstuffworks.com/e-mail-messaging/email.htm`. (accessed: 14.11.2018).

[2] Pankaj. (). JavaMail Example – Send Mail in Java using SMTP, Adresse: `https://www.journaldev.com/2532/javamail-example-send-mail-in-java-smtp`. (accessed: 14.11.2018).

[3] J. Mani, B. Shannon und K. Oberoi. (). JavaTM Platform, Enterprise Edition 6 API Specification, Adresse: `https://docs.oracle.com/javaee/6/api/javax/mail/internet/MimeBodyPart.html`. (accessed: 14.11.2018).