Informatik B

Vorlesung 16
Netzwerkprogrammierung

Netzwerkprogrammierung

- Mit Java-Programmen ist es möglich, Verbindungen über Netze aufzubauen
- Die Basisfunktionalität zur Netzwerkprogrammierung stellt das Paket **`java.net`** bereit
- Genau wie in anderen Bereichen gibt es in der Netzwerktechnik eine Reihe von Begriffen, deren Bedeutung bekannt sein sollte

Was ist ein Netzwerk?

- Verbindung zwischen zwei oder mehr Computern um Daten auszutauschen
- Beispiele:
  - Entfernter Dateizugriff
  - Gemeinsame Nutzung eines Druckers
  - Unternehmensweiter Informationszugriff/-austausch
  - Email
  - Remote Login auf andere Rechner
  - Clusterung von Rechnern zur Leistungssteigerung
  - Etc.

Begriffe

- *Host*:
  - Eine Maschine im Netzwerk, die mit einer eindeutigen Adresse (*IP-Nummer*) angesprochen werden kann
- *IP-Nummer*:
  - Eine eindeutige Adresse, die jeden Host im Internet kennzeichnet
  - Die Adresse ist im IPv4-Format eine 32-Bit-Zahl, auf einen Quadratkilometer Erdoberfläche kommen gerade mal 8,4 Adressen im IPv4-Format
  - Die Adresse ist im IPv6-Format eine 128-Bit-Zahl, auf jeden Quadratmillimeter Erdoberfläche kommen ca. 667 Billiarden IPv6-Adressen

Begriffe

- *Host-Name*:
  - Ein symbolischer Name für die *IP-Nummer*
  - Durch Techniken wie *DNS* (*Domain Name Service*) und Suns *NIS* (*Network Information Services*) werden diese Namen auf die *IP-Nummern* abgebildet
- *Paket* (engl. *packet*):
  - Teilmenge einer Nachricht, die mit Adressinformationen versehen wird und durch das Netzwerk geroutet wird
  - Der Weg zweier Pakete mit gleichem Bestimmungsort muss nicht gleich sein

Begriffe

- *Port*:
  - Ein *Host* bietet verschiedene *Ports* an, über die Informationen ausgetauscht werden können
  - Die gleichzeitige Kommunikation mit mehreren Verbindungen an verschiedenen *Ports* ist möglich
  - Im allgemeinen verbirgt sich hinter einem *Port* eine bestimmte Serveranwendung (z.B. http auf *Port* 80)
  - Beispielhaft: Ein Bürogebäude mit verschiedenen Zimmern, hinter einer Zimmernummer wird etwas Bestimmtes bearbeitet

Begriffe

- *Protokoll* (eng *protocol*):
  - Um Daten auszutauschen muss man sich auf ein Protokoll einigen
  - Es handelt sich um Regeln, welche das Format, den Inhalt, die Bedeutung und die Reihenfolge gesendeter Nachrichten zwischen verschiedenen Instanzen festlegen
  - Ein Protokoll kann wie eine menschliche Sprache verstanden werden
  - Es gibt eine große Protokollvielfalt mit sehr unterschiedlichen Ansätzen und Ansprüchen (http, ftp, smtp, ntp, …)

Begriffe

- *URI* (*Uniform Resource Identifier*)
  - Zu betrachten wie ein Dateiname für das Dateisystem oder eine Ressource auf einem anderen Rechner
  - **`www.inf.uos.de/index.html`**
- *URL* (*Uniform Resource Locator*)
  - Die Spezialisierung einer *URI*, die an ein Protokoll gebunden ist (http, ftp,…)
  - **`http://www.inf.uos.de/index.html`**
  - Eine *URL* ist formal im *RFC* (*Request For Comment*) 1738 beschrieben

Begriffe

- *IP*:
  - *Internet Protocol*
  - Das Internet Protocol stellt die Grundlage des Internet dar
- *TCP*:
  - *Transmission Control Protocol*
  - Verbindungsorientiertes Protokoll, welches auf Basis von IP eine sichere und fehlerfreie Punkt-zu-Punkt-Verbindung realisiert
  - *TCP* stellt einen virtuellen Kanal zwischen zwei Endpunkten einer Netzwerkverbindung her

Begriffe

- *UDP*
  - *User Datagram Protocol*
  - Verbindungsloses Protokoll um Daten, die über das Internet übertragen werden, der richtigen Anwendung zukommen zu lassen
  - Dazu wird direkt ein Port angegeben, an dem die Daten angeliefert werden sollen
  - Eine Anwendung, die UDP nutzt, muss gegenüber verloren gegangenen und umsortierten Paketen unempfindlich sein oder selbst entsprechende Korrekturmaßnahmen beinhalten
  - Da vor Übertragungsbeginn nicht erst eine Verbindung aufgebaut werden muss, können die Hosts schneller mit dem Datenaustausch beginnen

Begriffe

- *Client/Server*:
  - Die Kommunikation zwischen zwei Rechnern läuft oftmals auf Basis einer Client/Server Beziehung ab
  - *Server*
    - Stellt einen Dienst zu Verfügung, der von anderen Rechnern genutzt werden kann
    - Ein Prozess läuft im Hintergrund und wartet auf einen Verbindungsaufbau
  - *Client*
    - Nutzer eines oder mehrerer Dienste
    - Bei Bedarf wird eine Verbindung zu einem Server aufgebaut
    - Der Client hält sich an das vom Server vorgegebene Protokoll

Begriffe

- *Localhost*:
  - Pseudo-Adresse für den eigenen Host (Rechner)
  - Die IP-Adresse für localhost lautet: 127.0.0.1

InetAddress

- In Java wird die Adressierung von Rechnern über die Klasse **`java.net.InetAddress`** geregelt
- Ein Rechner kann über die IP-Adresse (IPv4/IPv6) oder über seinen Hostnamen adressiert werden
- Die Klasse bietet keine Konstruktoren an
- Objekte der Klasse können über statische Methoden erfragt werden:
  - **`static InetAddress[] getAllByName(String host)`**
  - **`static InetAddress getByAddress(byte[] addr)`**
  - **`static InetAddress getByAddress(String host, byte[] addr)`**
  - **`static InetAddress getByName(String host)`**
  - **`static InetAddress getLocalHost()`**
- Beispiel: **`inetaddress1`**

`InetAddress`

- Mittels eines **`InetAddress`**-Objektes kann überprüft werden, ob ein Rechner erreichbar ist oder nicht
- Dies kann mit der Methode **`boolean isReachable(int millis)`** überprüft werden
- Die Methode sendet einen Request (ICMP ECHO REQUEST) und wartet auf eine Antwort
- Erfolgt innerhalb des angegebenen Zeitraumes keine Antwort, wird vermutet, dass der Rechner nicht erreichbar ist
- Achtung: Ein Rechner muss nicht auf den Request antworten (z.B. wegen einer Firewall)
- Beispiel: **`inetaddress2`**

`InetAddress`

- Mit dem **`InetAddress`**-Objekt kann auch die lokale Internetadresse ermittelt werden
- Dazu kann mit der statischen Methode **`getLocalHost()`** ein **`InetAddress`**-Objekt der lokalen Klasse beschafft werden
- Beispiel: **`inetaddress3`**

URL

- Eine URL hat immer ein Protokoll und ist die Spezialisierung einer URI (Uniform Resource Identifier)
- Um ein **`URL`**-Objekt zu erzeugen, ist es am einfachsten, über eine String-Repräsentation der **`URL`**-Adresse zu gehen
- **`URL url = new URL(„http://www.inf.uos.de/index.html“);`**
- **`URL url = new URL(„http“,“www.inf.uos.de“, „index.html“);`**
- Jeder der Konstruktoren löst eine **`MalformedURLException`** aus, wenn das Argument im Konstruktor entweder **`null`** ist oder er ein unbekanntes Protokoll beschreibt
- Ein **`URL`**-Objekt kann diverse Informationen liefern
- Beispiel: **`url1`**

URL

- Existiert ein **`URL`**-Objekt kann darüber ein Datenstrom geöffnet werden
- Dazu muss zunächst ein **`URLConnection`**-Objekt erstellt werden, indem eine Verbindung geöffnet wird: **`openConnection()`**
- Diese Verbindung kann nach einem **`InputStream`** gefragt werden: **`getInputStream()`**
- Verkürzt kann an einem **`URL`**-Objekt auch die Methode **`openStream()`** aufgerufen werden
- Ein **`OutputStream`** kann nicht erzeugt werden
- Beispiel: **`url2`**

URL

- Die URL-Verbindungen sind Highlevel-Verbindungen
- Man muss sich nicht um Übertragungsprotokolle wie HTTP oder TCP/IP kümmern
- Alle höheren Verbindungen bauen auf Sockets auf, und auch die Verbindung zu einem Rechner über eine URL ist mit Sockets realisiert

Socket

- Ein Socket ist eine Programmierschnittstelle zur Kommunikation zweier Rechner mittels TCP/IP
- Die Datenübertragung ähnelt dem Zugriff auf eine Datei:
  - Verbindungsaufbau
  - Daten lesen und/oder schreiben
  - Verbindung beenden

Socket

- Werden mehrere Computer verbunden, implementiert jeder Rechner einen Socket:
  - Client baut eine Verbindung auf, um Anfragen zu stellen
  - Der Server öffnet eine Verbindung zum horchen auf Anfragen
- Es lässt sich in der Realität nicht immer ganz trennen, wer Client und wer Server ist
- Für den Betrachter von außen ist der Server der Wartende und der Client derjenige, der die Verbindung initiiert

Socket

- Zu einem **`Socket`** gehört eine Host-Adresse mit der Angabe des Ports
- Es exisistieren verschiedene Konstruktoren, um ein **`Socket`**-Objekt zu erzeugen
- Bei der Erzeugung wird eine Verbindung aufgebaut, was einige Sekunden Zeit in Anspruch nehmen kann
- Wurde das **`Socket`**-Objekt erfolgreich erzeugt, besteht eine Verbindung per TCP/IP
- Beispiel: **`socket1`**

Socket

- Ein **`Socket`** kann sich nur an einem Host zu einem verfügbaren Port verbinden
- Will man wissen welche Ports auf einem Rechner angeboten werden, kann man dazu **`Socket`**-Objekte verwenden
- Man läuft einen Bereich der Ports eines Hosts ab und versucht eine Verbindung herzustellen
- Der Aufbau einer Verbindung kann einige Zeit in Anspruch nehmen
- Klappt die Verbindung, ist der Port geöffnet
- Ist der Port nicht offen wird eine **`IOException`** geworfen
- Beispiel: **`socket2, socket2_improved`**

Socket

- Ist die Verbindung hergestellt, können Nachrichten über die Verbindung geschickt werden
- Dazu kann man sich einen **`OutputStream`** mit der Methode **`getOutputStream`** beschaffen
- Will man Daten aus dem Socket lesen beschafft man sich mit der Methode **`getInputStream`** ein **`InputStream`**-Objekt
- Die Stream-Objekte können je nach Bedarf in Filterstreams geschachtelt werden
- Beispiel: Echoserver: **`socket3`**
- Beispiel: Beliebiger Port, z.B. Time (Nr.13) **`socket4`**

Socket

- In der Regel können nur eine begrenzte Anzahl von Sockets aufgebaut werden
- Wie bei den Streams gilt zu beachten, dass ein Socket geschlossen wird, wenn es nicht mehr benötigt wird
- Dazu wird am Socket die Methode close() aufgerufen
- Sind noch zu dem Socket gehörige Streams offen, werden diese ebenfalls geschlossen

Webserver aus Clientsicht

- Ein Webserver stellt Webseiten zur Verfügung
- Ein Client kann Anfragen an den Webserver stellen und bekommt das Ergebnis der Anfrage zurück
- Das Protokoll ist HTTP (*Hypertext transfer protocol*)
- Typischerweise horcht der Webserver auf Port 80 (oder 8080) auf Anfragen eines Clients
- Das Protokoll benutzt eine TCP/IP-Socket-Verbindung und ist textbasiert

Webserver aus Clientsicht

- HTTP hat folgendes Format:
  - Eine Zeile am Anfang: Dies kann entweder eine Anfrage (also eine Nachricht vom Client) oder eine Antwort vom Server sein
  - Einige Kopf-Zeilen (engl. *header*): Informationen über den Client oder Server, zum Beispiel über den Inhalt, der Header endet immer mit einer Leerzeile
  - Einen Körper (engl. *body*): Der Inhalt der Nachricht, entweder Benutzerdaten vom Client oder die Antwort vom Server

Webserver aus Clientsicht

- Ist die Verbindung aufgebaut, wird eine Anfrage formuliert, auf welches Objekt zugegriffen werden soll
- Neben der Anfrage wird auch das Protokoll festgelegt, mit dem übertragen wird
- HTTP definiert mehrere Hauptmethoden, von denen drei zu den wichtigsten gehören:
  - GET: Eine einfache Anfrage nach einer Information, der Client kann Daten an die URL anhängen und so zum Server schicken
  - POST: Die POST-Methode erlaubt es dem Client, Daten über einen Datenstrom zum Server zu schicken
  - HEAD: Funktioniert ähnlich wie GET, nur dass nicht das gesamte Dokument verschickt wird, sondern allein Informationen über das Objekt, so sendet diese Methode zum Beispiel innerhalb einer HTML-Seite die innerhalb von <HEAD>....</HEAD> befindlichen Informationen

Webserver aus Clientsicht

- Eine typische GET-Anfrage vom Client an den Webserver sieht wie folgt aus:
  `GET /directory/index.html HTTP/1.0 \r\n\r\n`
- Das erste Wort ist die Methode des Aufrufs (auch Anfrage, engl. *request*)
- Die zweite Angabe bei der Anfrage an den Server ist der relative Dateipfad
- Als nächstes folgt optional die Angabe des verwendeten Protokolls, wird das Protokoll angegeben erhält man in der Antwort zusätzliche Informationen
- Zuletzt folgt ein Zeilenumbruch (*Carriage-Return*, *Newline*)
- Durch den zweiten Zeilenumbruch wird eine Leerzeile eingegeben, die den Abschluss des Befehls signalisiert

Webserver aus Clientsicht

- Diese Funktionsweise kann mit einem **`telnet`** Aufruf sehr einfach getestet werden
- Demo
- In Java muss dazu eine TCP/IP Verbindung mittels eines **`Socket`**-Objektes aufgebaut werden
- Die Streams müssen beschafft werden
- Die **`GET`**-Anfrage wird an den Server geschickt
- Die Antwort wird vom **`InputStream`** ausgelesen

Webserver aus Clientsicht

- Am Anfang des Headers gibt der Webserver weitere Informationen, unter anderem eine Statusmeldung, zurück
  - **`200`** (OK): Die Anfrage vom Client war korrekt, und die Antwort des Servers stellt die gewünschte Information bereit
  - **`400`** (Bad Request): Die Anfrage war fehlerhaft formuliert
  - **`404`** (Not Found): Das referenzierte Dokument kann nicht gefunden werden
  - **`500`** (Internal Server Error): Meistens durch fehlerhafte CGI-Programme hervorgerufen
- Beispiel: **`socket5`**

Socket auf Clientseite

- Mit einem Socket kann also eine Verbindung zu einem Server an einem bestimmten Port aufgebaut werden
- Es können Dateiströme in beide Richtungen erfragt werden
- Daten können gesendet und empfangen werden

Zusammenfassung

- Begriffe
- URL/URI
- Socket-Objekte

Ausblick

- Serverimplementierung
- UDP