03-05-H -709.53 Informatik für Nichtinformatiker (12)

Prof. Dr. Udo Frese
Tobias Hammer

OSI-7-Schichtenmodell

Sockets (TCP/IP) in Ruby

Beispiel: Chatserver in Ruby

Was bisher geschah

- **Ausnahmebehandlung (Exception Handling) ist ein Sprachmechanismus die Behandlung von Fehlern vom normalen Programmablauf zu trennen und dadurch übersichtlicher zu machen**
  - Ausnahme wird ausgelöst (raise ExceptionClass, message)
  - Programm ablauf wird unterbrochen
  - Methoden werden nacheinander abgebrochen, bis…
  - … die Ausnahme behandelt wird (rescue ExceptionClass=>error)
  - Programmteile zum „aufräumen“ werden immer ausgeführt (ensure)
- **Vorteile gegenüber traditioneller Vorgehensweise mit Fehlercodes**
  - Keine Kaskaden von if-then Anweisungen
  - Keine Weiterleitung von Fehlern nötig
  - Man kann Fehlerbehandlung nicht vergessen
  - Daten zum Fehler sind verfügbar (im Exception Objekt)
  - Programmteile zum Aufräumen nur einmal
- **Eigene Fehlermeldungen: Unterklasse von `StandardError` suchen**

OSI-7-Schichtenmodell

OSI-7-Schichtenmodell

Open Systems Interconnection Reference Model

- **Quelle: wikipedia/de/OSI-Modell**
- **7 aufeinander aufbauende Schichten von Netzwerkkommunikation**
- **Lösung für die einzelnen Schichten sollen austauschbar sein**
  - Email soll funktionieren, unabhängig davon, ob die Rechner per Funk, Kabel, Lichtleiter, DSL oder einer Kombination angebunden sind
  - Unabhängig vom Betriebssystem
  - Verschiedene Anwendungen sollen die selben Basisdienste verwenden können
- **In der Praxis meist nicht so fein unterteilt**
  - Gerät / System deckt mehrere Schichten ab

OSI-7-Schichtenmodell

Schichten

- **1. Physikalische Schicht**
  - Wie werden Bits wirklich übertragen?
  - Kabel, Stecker, Leitungen, Signalspannungen, …
  - Bsp.: Ethernet , IEEE 802.11 (WLAN)
- **2. Sicherungsschicht**
  - Wie werden Daten unterteilt und Fehler erkannt?
  - Paketgröße, Prüfsummen
  - Bsp.: Ethernet, IEEE 802.11 (WLAN)
- **3. Vermittlungsschicht**
  - Wie werden Daten über Zwischenstationen geleitet?
  - Identifizierung von Rechnern, Weiterleitungsstrategie
  - Kann verschiedene OSI-1/2 Netze verknüpfen
  - Bsp.: IP

OSI-7-Schichtenmodell

Schichten

- **4. Transportschicht**
  - Wie werden große Daten in Pakete zerlegt und korrekt wieder zusammengesetzt?
  - Wie wird Datenverlust eines Paketes korrigiert?
  - Wie werden verschiedene Datenströme an einem Rechner kombiniert und getrennt?
  - Protokoll mit Paketzähler, Wiederholaufforderungen, Timeout
  - Bsp.: TCP
- **5. Kommunikationssteuerungsschicht**
  - Wie setze ich die Kommunikation nach kurzfristigem Abbruch der Verbindung fort?

OSI-7-Schichtenmodell

Schichten

- **6. Darstellungsschicht**
  - Wie übersetzt man systemabhängige Kodierungen?
  - Ziffernreihenfolge bei mehrstelligen Zahlen
  - Kodierung von Buchstaben
- **7. Anwendungsschicht**
  - Wie kommunizieren Computer bzgl. bestimmter Aufgaben
  - Eine Vielzahl unterschiedlicher Protokolle für verschiedene Aufgaben
  - Bsp.: FTP, HTTP (Web), POP/SMTP (Email), SSH (Sicherheit), …

OSI-7-Schichtenmodell

- **Praktisch wichtige Schichten**
- **1./2. Netzwerk (Ethernet, WLAN, DSL, Router, …)**
- **3./4. Internet (TCP/IP)**
- **5./6./7. Dienste (FTP, Email, Web, IRC, Onlinegames, …)**

- **Netzwerke und Dienste gibt es viele, dazwischen liegt meist TCP/IP**

Sockets (TCP/IP) in Ruby

Sockets (TCP/IP) in Ruby

- **Datenaustausch zwischen Rechnern aufbauend auf Ebene OSI 4**
- **selbstdefiniertes Protokoll**
- **Socket (“Sockel”), eine virtuelle Datenverbindung zwischen zwei Programmen**
  - Daten (Zeichen), die an einem Ende geschrieben werden können am anderen Ende gelesen werden
- **Praxis: Oft Text**
- **Datenstrom (Streaming), keine inhaltliche Einteilung in Pakete**
  - Technische Einteilung in Pakete transparent
  - Wenn Daten aus Einheiten bestehen, muss das Protokoll das vorsehen (z.B. Text zeilenweise)
- **Rechnername mit domain**
  - z.B. `einstein.informatik.uni-bremen.de`
- **Portnummer zur Unterscheidung mehrerer Server auf gleichem Rechner (80:HTTP, FTP: 20,21)**

Sockets (TCP/IP) in Ruby

Client/Server Rollenverteilung

- **Aufgabenteilung zwischen vernetzten Rechnern**
  - Ein Rechner bietet einen Dienst an (Server)
  - Ein anderer Rechner verwendet diesen Dienst (Client)
- **Beispiel: Webserver**
  - Server (www….com Rechner, apache) speichert Webseiten auf Festplatte und versendet diese per Netzwerk auf Anfrage per Netzwerk
  - Client (Netscape, Internet explorer) versendet Anfrage an Server und erhält dadurch Webseiten
- **Technischer Unterschied**
  - Server steht zur Verfügung (läuft dauernd)
  - Client initiiert Verbindung
  - $\Rightarrow$ Client muss Servernamen kennen, nicht umgekehrt

Sockets (TCP/IP) in Ruby

Beispiel: Einfaches Client/Server Paar

- **Client**
  - Liest Textzeile von Tastatur
  - Sendet Textzeile an Server
  - Empfängt Antwort von Server
  - Gibt Antwort aus
  - Bricht bei „QUIT“ ab
- **Server**
  - Empfängt Textzeile von Client
  - Gibt Textzeile aus
  - Sendet Anzahl Textzeilen als Bestätigungsantwort
  - Bricht bei „QUIT“ ab

Sockets (TCP/IP) in Ruby

Client

- **`socket`: Bibliothek für Sockets**
- **`TCPSocket`: „normaler“ Socket**
- **`localhost`: Rechnername, immer der eigene Rechner**
- **12345: Portnummer (unsere)**
- **Lesen/Schreiben, wie `File`**
  - `Socket.puts`: Text schreiben
  - `Socket.write`: Binärdaten schreiben
  - `Socket.gets`: Textzeile lesen
  - `Socket.read`: Binärdaten lesen
  - `Socket.close`: Ende

```
require 'socket'

socket = TCPSocket.new
       ("localhost", 12345)
puts "Connection established"
while true do
  line = gets
  socket.puts line
  response = socket.gets
  puts "Answer: #{response}"
  if (line=="QUIT") then
       break
  end
end
socket.close
```

Sockets (TCP/IP) in Ruby

Client

- **Fehlerbehandlung wichtig!**
- **Allgemein**
  - SystemCallError
- **Speziell**
  - ENETDOWN
  - EHOSTDOWN
  - EAGAIN
  - ETIMEDOUT
  - EINVAL
  - ECONNABORTED
  - ECONNREFUSED
  - ECONNRESET

```
require 'socket'

begin
    …

rescue Errno::ECONNREFUSED
  puts "Connection refused"
rescue SystemCallError=>err
  puts "Error in network
       connection\n #{err.message}
       #{err.class.name}"
end
```

- **Tipp: Erst allgemein behandeln, dann speziell wenn nötig.**

Sockets (TCP/IP) in Ruby

Server

- **TCPServer: Dienst an einem Port zu dem Clients sich verbinden können**
- **server.accept: Wartet auf einen verbindenden Client, baut eine Verbindung auf und liefert TCPSocket zurück**
- **Ein- / Ausgabe, wie gehabt**
- **socket.close: Schließt Verbindung zu einem Client**
- **Server.close: Schließt ganzen Server**
- **socket=server.accept bis socket.close in Schleife**

```
require 'socket'

server = TCPServer.new(12345)
while true do
  socket = server.accept
  ctr=0
  while true do
    line = socket.gets
    line.chomp!
    if line=="QUIT" then
      break
    end
    puts line
    ctr += 1
    socket.puts ctr
  end
  socket.close
end
server.close
```

Sockets (TCP/IP) in Ruby

Fehlerbehandlung

- **Im Server noch wichtiger, als im Client, weil Server unbeaufsichtigt läuft**
- **Rescue in innerer Schleife**
  - Verbindung wird abgebrochen
  - Server akzeptiert aber neue Verbindungen
- **`socket.eof?`: Wurde die Verbindung beendet?**

```
…
while true do
  begin
    socket = server.accept
    …
  rescue Errno::EINVAL,
         Errno::ECONNRESET
    puts "Connection closed.“
  rescue SystemCallError=>err
    puts "Error in connection\n
     #{err.message}
     #{err.class.name}“
  ensure
     socket.close
  end
end
…
```

Sockets (TCP/IP) in Ruby

Mehrere Verbindungen gleichzeitig

- **Server sollen mehrere Clients bedienen**
- **`TCPSocket.gets`, `TCPSocket.read`, `TCPServer.accept` warten, bis Daten kommen**
- **`TCPSocket.puts`, `TCPSocket.write`, warten ggf. bis Gegenseite angenommen hat**
- **Man weiß nicht, wo die nächsten Daten kommen**
- **Threads: Mehrere Ruby Programme werden gleichzeitig ausgeführt**
  - `Thread.new(data) do |threadData| … end`
  - Elegant
  - Vielseitig
  - Nutzt Mehrkernprozessoren aus
  - Hat einige Tücken ⇒ nicht in dieser Vorlesung

Sockets (TCP/IP) in Ruby

Mehrere Verbindungen gleichzeitig

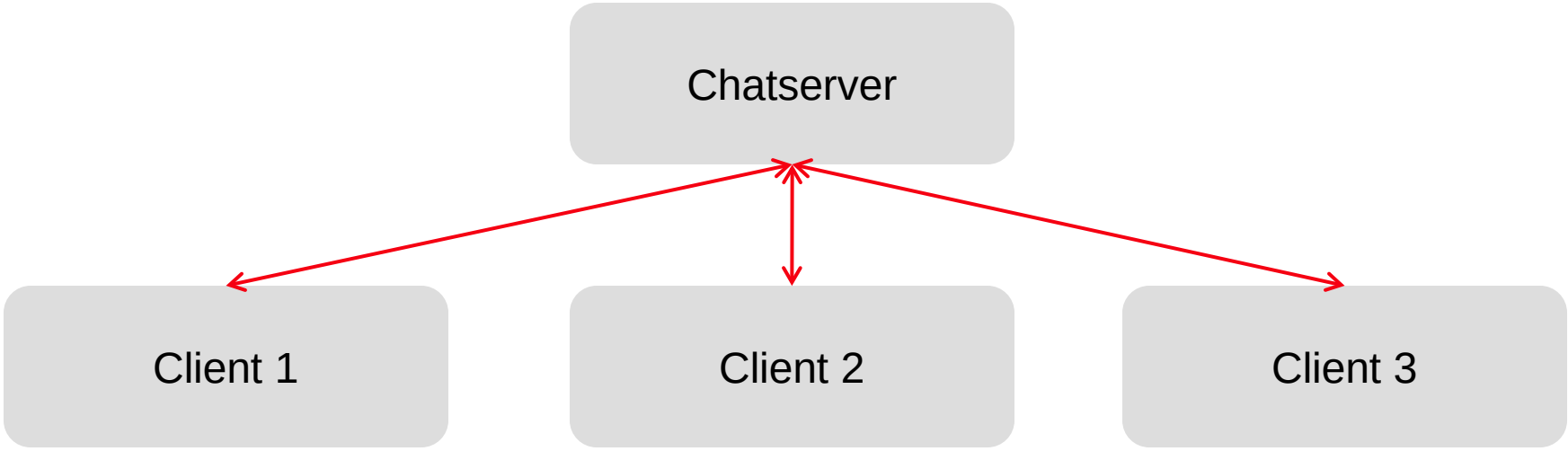
- **`select` Befehl wartet, ob einer einer Liste von Sockets/Dateien/etc. lesbar oder schreibbar wird**
- **`readable, writable = select (readlist, writelist, nil, timeout)`**
- **`readlist`: Liste von potentiell zu lesenden Verbindungen**
  - `TCPSocket`
  - `TCPServer` (lesbar, wenn `TCPServer.accept` möglich)
  - `STDIN`: Tastatur
- **`writelist`: Liste von potentiell zu schreibenden Verbindungen**
  - `TCPSocket`
  - `STDOUT`: Bildschirm
- **`timeout`: Warte bis zu timeout Sekunden**
- **Ergebnis: Liste der les-/schreibbaren Verbindungen oder `nil`**

Beispiel: Chatserver in Ruby

Beispiel: Chatserver in Ruby

Aufgabe

- **Mehrere Clients verbinden sich zu einem Chatserver**
- **Client leitet Tastatureingaben an Server weiter**
- **Client gibt Serverausgaben aus**
- **Server soll die Eingaben aller Clients an alle Clients weiterleiten**
- **Server soll melden, wenn neuer Client hinzukommt/hinfortgeht**



Beispiel: Chatserver in Ruby

- **Lösung: inifrese0912_chatserver**

Zusammenfassung

- **OSI 7-Schichtenmodell spezifiziert Netzwerkkommunikation**
  - 1./2. Netzwerk (Ethernet, WLAN, DSL, Router, …)
  - 3./4. Internet (TCP/IP)
  - 5./6./7. Dienste (FTP, Email, Web, IRC, Onlinegames, …)
- **Ein Socket verbindet zwei Programme auf zwei Rechnern**
  - Server stellt sich bereit: `server=TCPServer.new(port)`
  - Client verbindet zu Server: `clsocket=TCPSocket.new(host, port)`
  - Server akzeptiert Client: `svsocket=server.accept`
  - Client/Server schreiben/lesen: `socket.puts, socket.gets`
  - Client/Server schließen: `socket.close`
  - Server schließt Port: `server.close`
- **Socket geschlossen: `socket.eof?`**
- **Warten auf Daten von mehreren Sockets: `select(…)`**