TCP-Schnittstelle

Verteilte Systeme und Komponenten

Gruppe 5 (Patrick Bucher, Pascal Kiser, Fabian Meyer, Sascha Sägesser)

01.04.2018

Die Kommunikationsschnittstelle zwischen `logger-component` und `logger-server` ist mit TCP umgesetzt. Lognachrichten sind als Klasse `Message` (aus dem `logger-common`-Projekt) umgesetzt. Diese Klasse ist beiden Seiten – Client und Server – bekannt und serialisierbar. Sie besteht aus drei `String`-Attributen: `level` (`LogLevel.name()`), `timestamp` (mittels `DateTimeFormatter.ISO_DATE_TIME` formatierter Zeitpunkt) und `message` – der eigentlichen Lognachricht.

Die Client-Server-Kommunikation besteht darin, Instanzen der Klasse `Message` vom Client auf den Server zu übertragen, wo sie mittels `StringPersistor` festgehalten werden. Eingehende Lognachrichten werden auf dem Server mit dem String `"OK"` quittiert. Bleibt diese Antwort aus, wird dies clientseitig über die Konsole gemeldet. (Für die Schlussabgabe wird das Protokoll dahingehend erweitert, dass der Client unquittierte Meldungen in einer lokalen Warteschlange behält und diese – evtl. nach wiederhergestellter Verbindung zum Server – erneut zu senden versucht.)

Zwar gibt es mit RMI und HTTP komfortablere und stabilere Protokolle als “blankes” TCP, diese haben aber einen grösseren Overhead und benötigen zusätzliche Komponenten: bei RMI dies spezielle Stub-Klassen, bei HTTP ein Webserver, der das entsprechende Protokoll “spricht”. Da die Implementierung mittels TCP auf Anhieb und problemlos funktioniert hat, gab es keinen Grund, auf ein höherwertiges Protokoll (mit entsprechendem Overhead) zu wechseln.

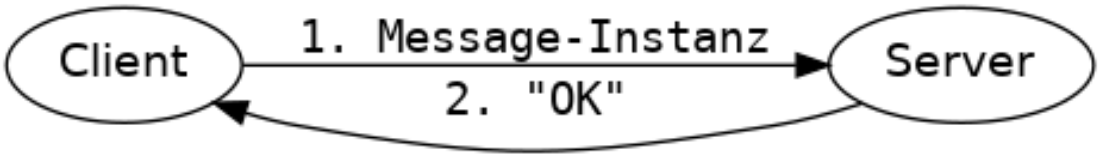

Abbildung 1: Das – denkbar einfache – TCP-Protokoll

1 Server

Der Server wird mit der Klasse `ConcurrentLoggerServer` (`logger-server`) umgesetzt. Der Port, auf den der Server hören soll, kann per Konstruktor übergeben werden. Derzeit wird er über die

Konstante `DEFAULT_PORT` mit dem Wert `1234` belegt. (TODO: Konfiguration über Kommandozeilenparameter!) Sobald der `ServerSocket` and den jeweiligen Port gebunden ist, nimmt er Verbindungen entgegen.

Für jede eingegangene Verbindung wird ein neuer `ConcurrentClientHandler` erstellt. Diese Klasse implementiert das Interface `Runnable`, sodass jeder Client in einem eigenen Thread bedient werden kann. Der Client-Handler dekoriert den `OutputStream` und den `InputStream` des Client-Sockets mit einem `ObjectOutputStream` bzw. einem `ObjectInputStream`, sodass er Objekte senden und empfangen kann. In der `run()`-Methode werden Objekte in einer Endlosschleife vom Socket entgegengenommen, zu einer `Message` gecastet und über den `StringPersistorAdapter` gespeichert. Nach der Speicherung wird der Eingang der Meldung mit dem String `"OK"` quittiert.

Wird der Socket clientseitig geschlossen, tritt beim Versuch ein Objekt vom Socket zu lesen eine `EOFException` auf. Die `run()`-Methode wird in diesem Fall mittels `return`-Anweisung beendet, sodass der jeweilige Thread endet. Eine Verbindung kann nur clientseitig geschlossen werden.

2 Client

Der clientseitige TCP-Code ist in der Klasse `LoggerComponent` umgesetzt. Die Verbindung zum Server wird in der Klasse `LoggerComponentSetup` verwaltet, sodass `LoggerComponent` bei der Instanzierung eine Referenz auf den Socket erhält. (Ändert sich die Konfiguration auf `LoggerComponentSetup`, wird die Verbindung entsprechend neu aufgebaut.)

`LoggerComponent` implementiert vier verschiedene `log`-Methoden: für eine “blanke” Nachricht, für eine formatierte Nachricht mit Parametern, für eine Nachricht mit `Throwable` und schliesslich für eine Kombination aller erwähnten Elemente. Jede dieser Methoden erstellt aus den gegebenen Parametern eine `Message`-Instanz. Diese wird an die `send`-Methode weitergegeben, welche sie über den `ObjectOutputStream` an den Server schickt und anschliessend blockierend auf das `"OK"` des Servers wartet. (Auf diese Weise geht immer nur höchstens eine Nachricht “verloren”, die jedoch clientseitigen vorhanden ist und später erneut gesendet werden könnte.)

Die Konfiguration des Clients erfolgt über die Konfigurationsdatei `config.xml`, welche unter anderem die Koordinaten des Servers (Hostname und Portnummer) enthält. (Die Klasse `Logging` im `game`-Projekt kapselt das Auslesen der Konfiguration und die Erstellung der `LoggerComponentSetup`-Instanz.)

3 Schwierigkeiten

TODO: beschreiben

- Der Socket des Loggers kann nicht explizit über die Schnittstelle geschlossen werden.
- Lösung mit Hack nötig (`null`-Konfiguration an `LoggerComponentSetup` übergeben, sodass dort der Socket geschlossen wird)

- Werden die Sockets nicht geschlossen, laufen die serverseitigen Handler-Threads immer weiter.
- `EOFException` beim serverseitigen Lesen signalisiert unterbrochene Socket-Verbindung.