Server-Side Renderer

Ausarbeitung zur Lösung des Blocks Spiels mittels Server-seitigem Rendering

Benedikt Lüken-Winkels - Matrikelnumber
Aaron Winziers - 1176638

Lehrstuhl für Systemsoftware und Verteilte Systeme
Universität Trier
01.04.2019

1 Einleitung

Im Rahmen der Vorlesung “Spieleprogrammierung” im Sommersemester 2019 sollte das Spiel *Blocks* mittels eines Rendering-Servers implementiert werden. *Blocks* war in diesem Fall ein simples “world building” Spiel in dem man sich durch eine Welt mit einem Avatar bewegen kann und in dieser Welt Blöcke setzen und löschen kann.

eine verteilte Lösung für *simple* Sudoku-Rätsel entwickelt werden. *Simple* soll in diesem Fall bedeuten, dass eine eindeutige Lösung für das Sudoku vorhanden ist. Hierfür wurden verschiedene Gruppen gebildet, welche den Lösungs-Algorithmus in einer frei gewählten Programmiersprache implementieren und sich lediglich an vorher definierten Standards orientieren. Zur Kommunikation der verschiedenen Programme untereinander, kam Apache Camel zum Einsatz. Diese Ausarbeitung behandelt die Lösung des Problems in der Programmiersprache Go und beschreibt insbesondere die Entwicklung der Camel-Komponente, welche den durch unsere Gruppe implementierten Kommunikationsweg über den Telegram-Messenger beschreibt.

2 Sudoku Algorithmus

Da die Boxen des Spielfeldes als eigenständige Einheiten handeln sollten, wurde ein trivialer Algorithmus für die Lösung des Problems eingesetzt.

Jede Box erhält zu Beginn eine initiale Konfiguration und die Position an welcher sie sich im Gesamtspielfeld befindet. Diese Konfiguration wird nun an die benachbarten Boxen übertragen und die Lücken des Feldes können mittels Ausschlussverfahren gefüllt werden. Sobald ein Feld mit neuen Informationen gefüllt wurde, wird die Konfiguration erneut an die benachbarten Boxen übertragen

Die genaue Implementierung des Algorithmus ist auf GitHub verfügbar [1].

3 Kommunikationsweg

Zur Kommunikation der Boxen untereinander wurde der Telegram-Messenger verwendet. Die Grundidee war dabei, dass jede Box einen eigenen Telegram-Bot erhält und die jeweiligen Bots ihre Informationen über Chats miteinander teilen.

Telegram-Bots werden mit Befehlen angesprochen. Diese beginnen mit einem “/” und müssen entsprechend in der Konfiguration des Bots definiert werden. Bei den für die

Problemstellung entwickelten Bots sind zwei mögliche Befehle vorgesehen: `/start` und `/fieldconfig` $fieldconfig$, wobei $fieldconfig$ die Konfiguration des jeweiligen Feldes in der durch die Vorlesung definierten Notation enthält.

`/start` gibt jedem Bot ein Start-Signal zur Lösung des Feldes und implementiert damit die Vorgabe aus der Vorlesung.

3.1 Groups und Channel

Während der Entwicklung fiel auf, dass Telegram-Bots keine Nachrichten von anderen Bots empfangen können, um unnötigen Spam zu vermeiden (`https://core.telegram.org/bots/api`). Dies gilt sowohl für direkte Nachrichten von Bot-zu-Bot, als auch für Nachrichten innerhalb eines Gruppenchats, wie er für die Lösung des Sudokus verwendet werden sollte. Telegram bietet die Möglichkeit anstelle eines Gruppenchats sogenannte *Channels* zu verwenden. Diese dienen normalerweise als eine Art Broadcast-Service, welcher von Telegram-Nutzern abonniert werden kann. Der Vorteil liegt hier darin, dass auch Bots diese Channels abonnieren können und den Nachrichteninhalt von anderen Bots lesen können.

4 Apache Camel

Die Camel-Komponente bildet die Basis für die Kommunikation der Telegram-Boxen zu Boxen, welche eine andere Form der Kommunikation implementiert haben. Des Weiteren wird sie verwendet, um den den Go-Prozess zu starten, indem zunächst die initiale Konfiguration der Box beim gegebenen Boxmanager über eine REST-API angefordert wird.

Die Camel-Instanz startet einen Bot und abonniert mittels MQTT die relevanten Topics der Nachbarboxen. Nach dem starten des Bots mit der Initialkonfiguration, sendet die Camel-Komponente das `ready` Signal an den Boxmanager. Die genannte Funktionalität wurde über drei verschiedene Routen definiert. Die Einfachste Route definiert dabei lediglich das Warten auf das Start-Signal vom Boxmanager und gibt dem Bot anschließend mittels des `/start`-Befehls das Kommando die Lösung des Sudokus zu beginnen. Die Definition der Route ist in Listing 1 zu sehen.

```
1 from("mqtt:sudoku?subscribeTopicNames=sudoku/start")
2     .log("Received start signal via MQTT from")
3     .process(new Processor() {
4         public void process(Exchange exchange) throws Exception {
5             exchange.getIn().setBody("/start");
```

```
6           }
7       })
8           .to("telegram:bots/681997552:AAG-Ht8fMY?chatId=-1005485");
```

Listing 1: Route für Start Nachricht

Nachrichten, welche der Bot in den Telegram-Channel schreibt, werden mithilfe der Route in Listing 2 verarbeitet. Die Camel-Komponente stellt dabei einen eigenständigen Bot dar, welcher Nachrichten aus dem Channel liest, diese in das entsprechende, einheitliche Format umformatiert und anschließend über das zugehörige MQTT-Topic publiziert.

Dabei treten zwei verschiedene Arten von Nachrichten auf. Die Erste enthält das Endresultat der Box und die Zweite die Nachrichten über Änderungen innerhalb der einzelnen Felder.

```
1  // Publish with own id
2      from("telegram:bots/681997552:AAG-Ht8fMY?chatId=-1005485")
3          .log("Received Telegram message. Forwarding it to MQTT-
               ↪ Topic.")
4          .choice()
5              .when(body().startsWith("/ready"))
6                  .bean(TelegramBot.class)
7                  .to("mqtt:sudoku?host=" + this.mqttHost
8                      + "&publishTopicName=sudoku/" + boxID + "/
                           ↪ result")
9              .otherwise()
10                 .bean(TelegramBot.class)
11                 .to("mqtt:sudoku?host=" + this.mqttHost
12                     + "&publishTopicName=sudoku/" + boxID);
```

Listing 2: Nachrichten von Telegram zu MQTT

Die letzte Route abonniert die relevanten Topics für die Box und ist in Listing 3 zu sehen. Welche Topics dabei relevant sind wird nicht berechnet, sondern ist durch eine Hashmap für jede Box statisch definiert.

```
1  System.out.println("Subscribing to topics: " + subscribeTopics.
       ↪ toString());
2      from("mqtt:sudoku?host=" + this.mqttHost
3          + "&subscribeTopicNames=" + subscribeTopics.toString())
4          .transform(body().convertToString())
5          .bean(MQTTConverter.class)
6          .log("Received fieldConfiguration via MQTT-Topic.")
7          .to("telegram:bots/681997552:AAG-Ht8fMY?chatId=-1005485");
```

Listing 3: Nachrichten von MQTT zu Telegram

4.1 Umformatierung

Die Umformatierung der Nachrichten in das entsprechend definierte Format fand mittels *Beans* statt. Hier wurde eine einfache Regular-Expression der Form: `/fieldconfig (box_a[147]),([0-2]),([0-2]):([1-9])` dazu verwendet die Nachrichten des Telegram-Bots auszuwerten und in JSON umzuwandeln. Die Rückrichtung folgt analog.

4.2 Probleme

Zunächst kam es bei der Implementierung der Camel-Komponente zu Fehlern, da die Telegram Komponente nicht mit der Java Version 11 zurecht kommt. Da es keine entsprechende Fehlermeldung gab, war es relativ aufwendig diesen Fehler ausfindig zu machen und zu lösen.

5 Anhang

Der gesamte Programmcode ist auf GitHub verfügbar.

[1] GoSudoku (Sudoku-Logik): `https://github.com/michaelwolz/gosudoku`
[2] Camel-Komponente: `https://github.com/michaelwolz/gosudoku-camel`