Handbuch zur Konfiguration und Nutzung

Befehle

- **Gruppenverwaltung (/gmgroup)**
  - */gmgroup create <Gruppenname> <Prefix...>*
    - Permission: gm.group.create
    - Erstellt eine neue Gruppe mit dem angegebenen Namen und Prefix.
      - Einschränkung Name: Maximal 14 Zeichen.
      - Einschränkung Prefix: Maximal 16 Zeichen inkl. Farbcodes.
  - */gmgroup info <Gruppenname>*
    - Permission: gm.group.info
    - Zeigt Informationen wie Prefix und Priorität an.
  - */gmgroup permadd <Gruppenname> <Berechtigung>*
    - Permission: gm.group.permadd
    - Fügt einer Gruppe eine Berechtigung hinzu.
  - */gmgroup permdel <Gruppenname> <Berechtigung>*
    - Permission: gm.group.permdel
    - Entfernt eine Berechtigung von einer Gruppe.
  - */gmgroup prefix <Gruppenname> [Prefix...]*
    - Permission: gm.group.prefix
    - Je nachdem, ob ein Prefix spezifiziert wurde, wird der Prefix der Gruppe entweder geändert oder angezeigt.
      - Einschränkung Prefix: Maximal 16 Zeichen inkl. Farbcodes.
  - */gmgroup priority <Gruppenname> [Priorität]*
    - Je nachdem, ob eine Priorität spezifiziert wurde, wird die Priorität der Gruppe entweder geändert oder angezeigt.
      - Einschränkung Priorität: Eine Ganzzahl zwischen 0 und 99, Grenzen inklusive.
  - */gmgroup delete <Gruppenname>*
    - Löscht eine Gruppe.
  - */gmgroup list*
    - Listet alle verfügbaren Gruppen auf.

Handbuch zur Konfiguration und Nutzung

- **Benutzerverwaltung (/gmuser)**
  - */gmuser group <Spielername> [Gruppenname] [Dauer...]*
    - Permission: gm.user.group
    - Je nachdem, ob eine Gruppe spezifiziert wurde, wird die Gruppe des Nutzers entweder geändert oder angezeigt.
      - Einschränkung Dauer: Eine valide Zusammensetzung aus Nd, Nh, Nm und Ns, wobei N eine Ganzzahl ist.
        - Beispiele: 2d5s, 2d 5s, 6h2m 3s, 4d7m23s
        - d = Tage, h = Stunden, m = Minuten und s = Sekunden

- **Spielerübersicht (/rankinfo)**
  - */rankinfo*
    - Permission: gm.rankinfo
    - Zeigt einem Spieler seinen aktuellen Rang und die Dauer, wie lange dieser noch anhält, an.

Verhalten & Funktionen

Spieler werden standardmäßig in die Gruppe „default“ eingetragen. Diese wird beim Laden des Plugins automatisch erstellt, falls sie nicht vorhanden ist. Die Gruppe kann außerdem nicht gelöscht werden, ihr Prefix und ihre Priorität sind jedoch variabel.

Die meisten Informationen, wie z.B. die Permissions eines Spielers sind gecacht. Das hat jedoch zur Folge, dass jede Operation, die die Datenbank modifiziert dafür sorgt, dass der Cache gereinigt und neu erstellt wird. In den meisten Fällen passiert dies jedoch asynchron, sodass sich auch im Profiling mit spark keine negativen Effekt erkennen lassen.

Das Plugin bietet derzeigt folgende Funktionen: Gruppenverwaltung im Spiel, Gruppen haben Namen und Prefix, Spieler können permanent und temporär zugewiesen werden, der Prefix der Gruppe eines Spielers wird beim Betreten und Verlassen des Servers, im Chat, über seinem Kopf und in der Tablist angezeigt, Gruppenaktualisierung erfordert keinen Rejoin, Nachrichten sind anpassbar für verschiedene Sprachen und orientieren sich an der Sprache im Client des Anwenders, ein Spieler kann Informationen zu seinem eigenen Rang wie Dauer und Name einsehen, durch Hibernate können die Daten in den allermeisten Datenbank - wie z.B. PostgreSQL oder MariaDB - abgelegt werden. Außerdem werden (Wildcard-)Berechtigungen und #hasPermission() unterstützt.

Handbuch zur Konfiguration und Nutzung

Besonderheiten bestimmter Funktionen (bspw. Time/Space-Complexity oder Verhaltensweisen)

- *GroupDao.deleteGroup(Group group)*
  - Die Laufzeit dieser Funktion ist abhängig von den Spielern in dieser Gruppe und den Permissions in dieser Gruppe. Ihre Laufzeit lässt sich durch $O(n+m)$ beschreiben, wobei n die Anzahl der Spieler in der Gruppe und m die Anzahl der Permissions in der Gruppe beschreibt.

- GmPermissible.hasPermission(String permission)
  - Die Laufzeit dieser Funktion ist im besten Fall $O(1)$, wenn die entsprechende Berechtigung der Gruppe des Spielers sich bereits im Cache befindet.
  - Die Laufzeit dieser Funktion ist im schlechtesten Fall abhängig von den Permissions der Gruppe, der Aufbau der einzelnen Permissions (viele Abschnitte – z.B. test.a.b.c.d.e.f.[…].z – generiert viele Wildcard-Checks) und der Tatsache, ob die Permission denn vorhanden ist und ob sie im Cache ist. Angenommen, jede Permission einer Gruppe hat 3 Abschnitte, die Gruppe hat 10 Permissions und die Permission ist (im Cache) nicht vorhanden, so ergibt sich: $O(4 \cdot 10) = O(40)$. Generischer heißt das: $O((a+1) \cdot n)$, wobei a die durchschnittliche Anzahl an Abschnitten pro Permission ist und n die Anzahl der Permissions beschreibt.
  - ***Lösungsansatz (außer Caching):***
    - *Anstatt ein Set zu verwenden, eine geordnete Datenstruktur einsetzen, die das Suchen in dieser erleichtert, bspw. ein Baum mittels alphabetischer Sortierung. Alternativ bereits geprüfte Permissions cachen.*

- *ScoreboardManager.updateScoreboards()*
  - Die Laufzeit dieser Funktion lässt sich mit $O(n^2 + n \cdot m)$ beschreiben, wobei n die Anzahl der aktiven Spieler ist und m die Anzahl der existierenden Gruppen darstellt. Jedoch wird diese Methode – zumindest, was den beanspruchenden Teil angeht – asynchron ausgeführt und alle von der Datenbank benötigten Gruppen und Nutzer werden in jeweils einer Query abgefragt. Außerdem wird diese Methode nicht (schnell) periodisch aufgerufen, sondern standardmäßig nur alle 5 Minuten einmalig, oder wenn die Datenbank modifiziert wurde. Da für jeden Spieler ein Scoreboard, welche für jede Gruppe ein Team und für jeden aktiven Spieler auch noch einmal den Spieler enthält, erstellt wird, ist die Raumkomplexität ebenso quadratisch.
  - ***Lösungsansatz:***
    - *Anstatt pro Spieler ein Scoreboard zu erstellen und zu aktualisieren, ein einziges Scoreboard für alle Spieler erstellen und die Sidebar mittels Packets modifizieren.*

Handbuch zur Konfiguration und Nutzung

Meine Einschätzung der allgemeinen Anforderungen

Bei der gesamten Entwicklung wurde auf die Einhaltung der allgemeinen Anforderungen geachtet. Zum Einhalten des Google-Java-Codestyles wurde ein Auto-Formatter verwendet, der Code ist kommentiert und an „kniffligen“ Stellen wurden auch Zeilen innerhalb von Methoden mit Kommentaren versehen. Das Profiling mittels spark hat keine Auffälligkeiten gezeigt, auch nicht bei kontinuierlichem Löschen, Erstellen und Ändern von Gruppen und Rang-Schildern. Sämtliches Datenbank-I/O wird über Hibernate und die dafür angelegte Klasse „Dao“ (bzw. die Wrapper wie GroupDao, UserDao, etc.) abgewickelt. Die Klasse Dao ist thread-sicher und bietet außerdem asynchrone Methoden mit integrierten Callbacks. Das gesamte I/O findet standardmäßig asynchron statt. Bzgl. Laufzeit und Space wurden die wichtigsten Funktionen im vorherigen beschrieben und ggf. mit Lösungsansätzen versehen. Bei meiner Betrachtung sind mir keine weiteren Besonderheiten aufgefallen, jedoch will ich nicht ausschließen, dass ich etwas übersehen habe. Daher möchte ich hier nur ungern Vollständigkeit garantieren, ohne dass eine weitere Person sich mit dem Code befasst hat, jedoch gehe ich nicht davon aus, dass sich noch weitere komplexe Funktionen finden.

Weitere Anmerkungen

Dieses Plugin verwendet Hibernate zum Verwalten der Datenbank. Die Konfiguration von Hibernate wird dabei unter „plugins/GroupManager/hibernate.properties“ erwartet. Eine beispielhafte Konfiguration findet sich auch im GitHub-Repository als „hibernate.properties.example“. Alternativ wird auch eine Standardconfig extrahiert, sofern die Datei nicht existiert. Durch Hibernate werden alle möglichen Datenbanken unterstützt. Auch nicht-relationale Datenbank wie Cassandra sollten daher unterstützt sein, jedoch kann ich dies nicht garantieren. Außerdem werden Sprachdateien in „plugins/GroupManager/locales“ in Form von Json-Dateien erwartet. Die Übersetzung ins Deutsche wird standardmäßig dorthin exportiert und dient als Fallback, falls die Sprache des Spielclients eines Nutzers nicht bekannt/nicht verfügbar ist. Für Unit-Tests wird eine Datenbank benötigt. Sollten die Tests fehlschlagen, so ist ein manueller Reset der Datenbank nötig. Code Coverage (ermittelt via IntelliJ IDEA „Run with Coverage“) liegt bei 72% für Klassen, 43% für Methoden und 27% für Zeilen und überschreitet damit in jeder Kategorie die Mindestanforderung von 15%. Alle Methoden für den Datenzugriff, teils auch asynchron via Command-Spoof, werden getestet.

Außerdem gibt es im obigen Ordner auch eine config.json, die (selbsterklärende) Parameter zur Konfiguration bereitstellt. Ich habe mich bewusst gegen Bukkits/Spigots YAML-System entschieden, da ich so die Konfiguration direkt als Klasse in Java darstellen kann, was den Zugriff im Code deutlich erleichtert und übersichtlicher macht. Außerdem bin ich mit entsprechenden Bibliotheken vertrauter.

Handbuch zur Konfiguration und Nutzung

Rank-Signs

Rank-Signs können durch das Platzieren eines Schildes erzeugt werden. Dabei ist wichtig, dass die erste Zeile „[gm:sign]“ enthält, die 2. Zeile muss den Namen des verknüpften Spielers beinhalten.

Sidebar

Der Inhalt der Sidebar kann über die entsprechende Locale-Datei konfiguriert werden. Dabei stehen folgende Platzhalter zur Verfügung:

- *%group%*
  - Die Gruppe des Spielers
- *%id%*
  - Die UUID des Spielers
- *%player%*
  - Der Name des Spielers
- *%duration%*
  - Zeitstempel, bis wann die Gruppe gültig ist
- *%prefix%*
  - Der Prefix der Gruppe