Dokumentation Netzwerkprotokoll

Definition

Das Netzwerkprotokoll wird verwendet, um Commands zwischen Client und Server auszutauschen und entsprechend darauf zu reagieren.

Auf Server- und Clientseite existieren die Packets als Klassen mit unterschiedlichen Inhalten (sog. ContentTypes). Um diese zu übermitteln werden Sie in Strings umgewandelt in folgender Form:

COMMAND_NAME -/- {Inhaltfeld1 -/- Inhaltfeld2} -/- ContentType

-/- Separator: Als Separator wird folgende Zeichenfolge verwendet “-/-”. Es wurde eine selten verwendete Abfolge von Zeichen verwendet, um Fehler bei der Übermittlung zu vermeiden.
COMMAND_NAME: vordefinierte Befehle (siehe Befehle)
{Inhalt}: Der Inhalt wird durch den ContentType vorgegeben. Der Inhalt ist von {} umschlossen und die Felder ebenfalls durch den gewählten Separator unterteilt.
ContentType: gibt Form des Inhalts vor, wird für das Entpacken verwendet.

Commands

Befehle, die vom Server zum Client und vom Client zum Server geschickt werden.

SEND_BROADCAST_CHAT *(implementiert für command_line Applikation)*	Chatnachrichten an alle Benutzer versenden
SEND_GAME_INTERNAL_CHAT	Chatnachrichten an alle Benutzer in einem Spiel versenden.
SEND_WHISPER_CHAT	Chatnachricht an nur eine Person senden.
BUILD_ROAD	Anfrage, um Road zu bauen. Bestätigung, dass Road gebaut wurde.
GAME_JOINED	Spieler ist einem Spiel beigetreten. Server sendet Game, Client reagiert darauf.

Befehle, die vom Client zum Server geschickt werden.

PONG	Implementation der Verbindungsüberprüfung.
SET_USERNAME	Anfrage, den Benutzernamen zu ändern
CREATE_GAME	Anfrage, ein neues Game zu erstellen.
REQUEST_GAMES	Anfrage, entweder alle offenen, alle begonnenen oder alle geschlossenen Games zu senden.

REQUEST_ALL_CONNECTED_PLAYERS	Anfrage, Liste aller verbundenen Spieler zu senden.
WANT_JOIN_GAME	Anfrage, Spiel beizutreten.
SHORT_DESTINATION_CARDS_SELECTED_IN_PREPARATION	Informieren darüber, welche Short-Destinationcards im Spiel-Status PREPARATION gewählt wurden.
TAKE_WHEEL	Anfrage, eine Radkarte aufzunehmen

Befehle, die vom Server zum Client geschickt werden.

CHANGED_USERNAME	Server teilt allen Nutzern mit, wenn ein Benutzername geändert wurde.
USERNAME_CONFIRMED	Server bestätigt dem Client die Änderung des Benutzernamens, oder teilt mit, dass der Benutzername bereits vergeben war und deswegen abgeändert wurde.
USER_DISCONNECTED	Mitteilung, dass eine Clientverbindung getrennt wurde
GAME_CREATED	Bestätigung, dass ein Game erstellt wurde.
SEND_ALL_CONNECTED_PLAYERS	Sende alle verbundenen Spieler.
PING	Implementation der Verbindungsüberprüfung.
INVALID_ACTION_WARNING	Wird gebraucht, wenn der User eine invalide Aktion durchzuführen versucht, welche das grundsätzliche Vorgehen nicht stört.
INVALID_ACTION_FATAL	Wird gebraucht, wenn der User eine invalide Aktion durchführt und dadurch das Funktionieren des Programms beeinflusst wird.
SEND_GAMES SEND_WAITING_GAMES SEND_STARTED_GAMES SEND_FINISHED_GAMES	Senden von entweder aller offenen, aller begonnenen oder aller geschlossenen Games.
NEW_PLAYER	Wird an alle sich im Game befindenden Players, inklusive dem neuen Player geschickt, um mitzuteilen, dass ein neuer Player verbunden ist.
GAME_STARTED_SELECT_DESTINATION_CARDS	Nachdem genug Spieler in einem Game sind, werden Short-Destination-Cards ausgeteilt und die Player müssen sie auswählen.
WHEEL_TAKEN	Benachrichtigung, dass Radkarte aufgenommen wurde.

* Befehle werden erweitert während der Umsetzung des Projektes.

Content-Types:

Content-Type	Klasse, deren Instanz mit diesem Content-Type geschickt wird
CHAT_MESSAGE	ChatMessage
STRING	String
INTEGER	Integer
USERNAME_CHANGE	UsernameChange, eine Klasse, die einen alten und einen neuen Benutzernamen als Felder besitzt.
GAME	ClientGame, eine Game-Repräsentation, welche für den Client sichtbar sein darf, also keine Informationen wie verdeckte Karten oder Karten eines bestimmten Players enthält.
STRING_LIST	List<String>
GAME_INFO_LIST	List<GameBase>, Liste von Gamebase, eine Klasse, von der Game oder ClientGame erben.
GAME_INFO_LIST_STARTED	List<GameBase> mit Status STARTED und PREPARATION
GAME_INFO_LIST_WAITING	List<GameBase> mit Status WAITING_FOR_PLAYERS
GAME_INFO_LIST_FINISHED	List<GameBase> mit Status FINISHED
GAME_STATUS	GameStatus
NONE	null
WHEEL_CARD	WheelCard

coder: En- und Decoder für die jeweiligen ContentTypes

Implementierung

Im Package ***shared*** implementiert, damit Server und Client beide darauf Zugriff haben.

Commands Implementierungen & Beispiele

Da wir für jeden ContentType der Packets Coder verwenden, ist es schwierig, hier die tatsächlichen versendeten Strings darzustellen. Es bietet sich eher an, die Packets darzustellen, welche gesendet werden. Wir werden es also folgendermaßen darstellen: **Command, Content, Contenttype**, weil die Packets ebenso mit new Packet(Command, Content, Contenttype) konstruiert werden.

Ping/Pong:
Server: PING, null, NONE
Client1: PONG, null, NONE
Server, PING, null, NONE
Fall, dass Client kein PONG zurückschickt

Server → Client2: USER_DISCONNECTED, username, STRING

Chat in der Lobby:
Client1: SEND_BROADCAST_CHAT, “Hallo”, STRING
Server → Client2: SEND_BROADCAST_CHAT, “Hallo”, STRING

Chat im Game:
Client1: SEND_GAME_INTERNAL_CHAT, “Hallo”, STRING
Server → Client2: SEND_GAME_INTERNAL_CHAT, “Hallo”, STRING

Whisper-Chat:
Client1: SEND_WHISPER_CHAT, new ChatMessage(“Player2”, “Hallo”), CHAT_MESSAGE
Server → Client2: SEND_WHISPER_CHAT, new ChatMessage(“Player2”, “Hallo”), CHAT_MESSAGE

Neuer Username:
Client1: NEW_USER, “myUsername”, STRING
Fall1:
Server: INVALID_ACTION_FATAL, “Username invalid, please try it again.”, STRING
Fall2:
Server → Client1: USERNAME_CONFIRMED, new Username(“myUsername”, “myUsername”), USERNAME_CHANGE
Server → Client2: NEW_USER, “myUsername”, STRING

Username ändern:
Client1: SET_USERNAME, “myName”, STRING
Fall1:
Server: INVALID_ACTION_FATAL, “Username invalid, please try it again.”, STRING
Fall2:
Server → Client1: USERNAME_CONFIRMED, new Username(“myUsername”, “myUsername”), USERNAME_CHANGE
Server → Client2: NEW_USER, “myUsername”, STRING.

Neues Game kreieren:
Client: CREATE_GAME, 3, INTEGER (hier heisst 3, dass dieses Spiel für drei Spieler ist)
Server: GAME_CREATED, clientGame, GAME

Game-Liste anfragen:
Fall 1:
Client: REQUEST_GAMES, WAITING_FOR_PLAYERS, GAME_STATUS
Server: SEND_GAMES, gameRepository.getWaitingGames(), GAME_INFO_LIST_WAITING
Fall 2:
Client: REQUEST_GAMES, STARTED, GAME_STATUS
Server: SEND_GAMES, gameRepository.geStartedGames(), GAME_INFO_LIST_STARTED
Fall 3:
Client: REQUEST_GAMES, FINISHED, GAME_STATUS
Server: SEND_GAMES, gameRepository.getFinishedGames(), GAME_INFO_LIST_FINISHED

Player-Liste anfragen:
Client: REQUEST_ALL_CONNECTED_PLAYERS, null, NONE
Server: SEND_ALL_CONNECTED_PLAYERS, listOfUsernames, STRING_LIST

Spiel beitreten:
Client1: WANT_TO_JOIN_GAME, “gameId”, STRING
Server → Client1: NEW_PLAYER, clientGame, GAME
Server → Client2: NEW_PLAYER, clientGame, GAME

Short-Destination-Cards ausgewählt im Spielstatus PREPARATION:
Server: GAME_STARTED_SELECT_DESTINATION_CARDS, clientGame, GAME

Client: SHORT_DESTINATION_CARDS_SELECTED_IN_PREPARATION, listOfIDs, STRING_LIST

Road bauen:
Client1: BUILD_ROAD, roadId, STRING
Server → Client1: BUILD_ROAD, clientGame, GAME
Server → Client2: BUILD_ROAD, clientGame, GAME

Radkarte aufnehmen:
Client: TAKE_WHEEL, null, NONE
Server: WHEEL_TAKEN, wheelCard, WHEEL_CARD

Spiel beitreten
Server: GAME_JOINED, clientGame, GAME