

Chess Java Project - Design Document

Short-link: <http://goo.gl/5vBez9>.

Overview

The Chess Project implements a web-service for managing chess games, and two clients, a command-line client and an Android application client.

The project contains a top-level Maven project called “Chess”, and 6 sub-projects:

- Board - chess logic library
- Protocol - chess service protocol library
- Server - chess service server
- Client - chess service client library
- AndroidApp - the android application.
- Shell - the shell client

High Level Design

Relationships between modules

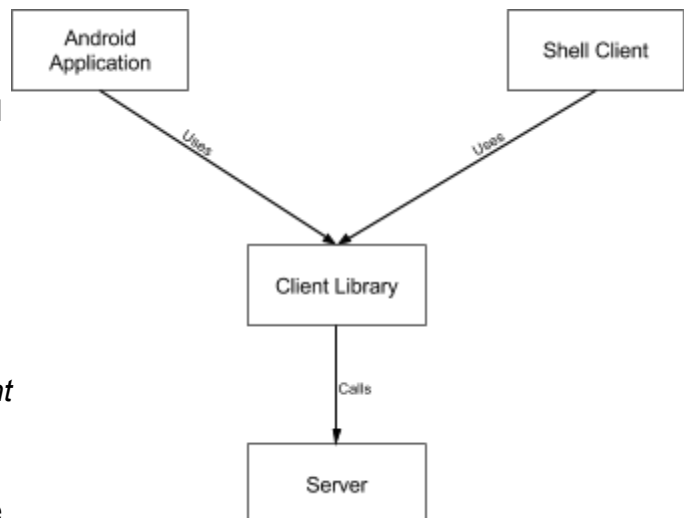
The *server* provides methods that allow the client to create an ad-hoc session and play chess games.

The *client library* manages the communication with the server, hiding the implementation details from the client applications, the *android application* and the *shell client*.

The *board* library is used by both the *client library* and the *server* in order to enforce the rules of chess and to provide classes needed in order to manage a chess game.

The client applications also use classes from the *board* library, such as *Coordinate* and *Piece*.

The *Protocol* library is a thin library that defines the communication protocol and provide basic tools for managing the requests. It is used by the *client library* and the *server*, and is in its own project only to prevent a dependency between the *client* project and the *server* project.



Protocol

Requests and Responses Structure

The requests and responses are defined using Google Protocol-Buffers.

The messages are defined in [chess.proto](#) in the *Protocol* project.

Network Protocol

While internally both the client and server use Protocol-Buffers, the end-point modules on both sides translate outgoing messages to JSON and incoming messages back from JSON to Protocol-Buffer messages.

The translation is done using [protobuf-java-format](#), which determines the exact JSON format.

Methods

The server exposes the following methods:

login(LoginRequest) → LoginResponse

The *login* method is used for logging in and starting a new session. The session token, returned as part of the LoginResponse, is the identifier of the session, and must be used in any future requests in the same session.

seek(SeekRequest) → SeekResponse

The *seek* method is used for seeking an opponent. This method may halt for a long time, and even timeout, in case there are no opponents looking for a game.

The game identifier, returned as part of the SeekResponse, is the identifier of the game, and must be used in any future requests in the context of the game.

Once the game started, it is possible to start listening to game event (see *getEvents*).

getEvents(EventsRequest) → EventsResponse

The *getEvents* method is used for polling the server for game events. Game events are any moves made, draw offers, resignation, etc. The *client* library starts listening for event automatically at the beginning of a game.

This method may not return immediately, but only when a new event happens in the game. It may also timeout in case no events happen for a long time. This is fine in the case of *getEvent*, and the *client* library will continue polling for events in case of timeout.

All events are sent to all players, including events initiated by the player. This allows clients to know that the event was handled and added to the game events list, thus the other player will also get the event.

Game actions

- *move(MoveRequest) → ErrorResponse* - Used for making a move.
- *resign(GameInfo) → ErrorResponse* - Used for resigning the game.
- *offerDraw(GameInfo) → ErrorResponse* - Used for offering or accepting a draw. Offering and accepting draw are considered logically equal, as the condition for draw by agreement is that both players agree on draw at the same time, regardless of who was the side that offered and who was the side that accepted.
- *declineDrawOffer(GameInfo) → ErrorResponse* - Used for declining a draw offer.
- *getPosition(GameInfo) → PositionResponse* - Used for getting the current board position of the game.

Detailed Design

Board Library

The board library implements the chess logic. It includes the following classes.

Main classes:

- Game - Represents a chess game.
- Piece - Represents a chess piece. This is a base class with subclasses for each piece type.
- Coordinate - Represents a coordinate on the chess board.
- Position - Represents a chess position.

The library is fully documented [here](#).

Protocol Library

The protocol library contains the protocol definition ([chess.proto](#)), and the class JsonParser that is used for translating JSON to protobuf and protobuf to JSON.

The JsonParser class is used at end-points in order to hide the network protocol from the implementation.

The library is fully documented [here](#).

Server Library

The server library contains several classes. The most important among them are:

ServiceImpl

The ServiceImpl class implements the logic for all exposed methods.

Because of the way the methods are exposed (see RequestHandler), all public methods must have a Status return value and take exactly 2 params, where the first is a proto message and the second is a ServiceImpl.Callback.

When adding a method that comply with that standard, this method will automatically be exposed by the RequestHandler.

This hides the request handling from the ServiceImpl library, completely decoupling it from the network protocol.

RequestHandler

The RequestHandler class is the request handler module that handles all request.

The RequestHandler creates a map of ServiceImpl's public methods.

All requests are handled using '@GET @Path(value="/{method}")'. When receiving a request, the RequestHandler looks for the method name in the map, and if it exists, invokes it with a protocol-buffer message translated from the request's JSON.

The request handler is the only class in the server aware of the JSON network protocol. By designing the server library this way, we allow replacing the network protocol by replacing the request handler, without changing ServiceImpl, or any other class.

The library is fully documented [here](#).

Client Library

The client library contains several classes. The most important among them are:

HttpGetter

The HttpGetter class is the only class aware of the JSON network protocol on the client side. It exposes a thin interface with just a single method that allows calling a method by name, passing the request as protocol-buffer message and receiving the response as one.

By designing the client library this way, we allow replacing the network protocol by replacing the http getter with another class providing a similar method (at which case, an interface should be extracted).

Client

The client class exposes all the methods that the client application needs in order to communicate with the server. It manages the session and the game in a way that does not require other modules to be aware of the session token or the game id.

The startListeningToEvents method tells the client to start listening to events. This invokes a new thread dedicated to listening to game events and handling them as they happen.

The library is fully documented [here](#).

AndroidApp

The Android application contains two activities and a service:

LoginActivity

The login activity is a relatively simple activity, responsible for logging in to the server. Once logged in, the board activity is activated.

BoardActivity

The board activity is the main activity of the application, allowing the use to seek an opponent and play a game.

ChessClientService

The chess client service is an Android service for the chess client. It uses the client library for communication with the chess server.

The library is fully documented [here](#).

Shell

The ShellClient is a small library, containing just a single class called "ShellClient".

This class contains a main function that instantiates it and runs it. Once running, it provides full client functionality, using the client library, in an interactive shell.

The library is fully documented [here](#).

Links

- Design document (this document): <http://goo.gl/5vBez9>
- Javadoc documentation: <http://gilran.github.io/chess/>
- GitHub project: <https://github.com/gilran/chess>
- HTTPS git clone url: <https://github.com/gilran/chess.git>
- Project sources ZIP: <https://github.com/gilran/chess/archive/master.zip>
- Project README: <https://github.com/gilran/chess/blob/master/README>