Concept

Room: multiple users can enter a room, each user in this room is either organizer or normal user. Organizer can start game. When start game, all the users in this room will automatically join the same game.

Game: Game contains multiple questions and multiple users.

Requirement

1. User can show rooms which are waiting for other user to join.
2. User can create a room.
3. User can join a exist room.
4. User can leave a room.
5. User can start a game if user is organizer in a room.
6. User can submit answer in a game.
7. User can leave a game.

Component

**Based on requirement, the server is divided to multiple modules.**

**Each module is separate from other modules and can scale it without influent other modules.**

Those modules either belongs to framework or normal module.

Framework modules

Cache

Message Queue

Session (Manage data persistence)

Normal Modules

Notification

Question

User

User State

Waiting

Waiting Action

Room

Room Action

Game

Game timer

Game Action

Below are the details for each module.

Cache module

Cache module can cache some frequently used data. Right now, the server used process memory to cache those data, but later can relace it to other cache service, such as Redis or MemoryCached.

Message Queue module

Message queue module can decouple module function. And also improve the system performance. Right now, the server used a customized message queue ( use net delegate to realize it). But the interface of message queue is compatible with Kafka, so later can replace it to kafka.

Session module

The server used some database function. This session module provide some common functions of database, such as transaction management, default interface and implementation.

Notification Module

The server used WebSocket to notify client about data change. This module is the main implementation of notification. Other module can call this module to send data to client.

Implementation

This module used SignalR (a net websocket library) to support client’s connection. When client connected to the server, the server will save some connection data to process memory. And when other module need to send data to a specific user, this module can scan the connection data, then send data to the user.

Scalability

Right now, the server is a single server, so it’s not high availability. To improve performance and availability, the system can add multiple server to support connect with client, and save the connection information in distributed storage engine, such as Redis cluster.

Question Module

The question module can save all questions and allow other module to get question based on some rules randomly.

Implementation

Right now, this module used database to save all questions and question statistic information. When other module requires a random question, this module can select a random record from database.

This module uses database because database support index.

Scalability

To improve scalability, the database can config as one master and multiple slaves. Another way is to load all questions to memory(if one question needs 1K and there are 10m questions, the total memory cost is 10g, it’s not too much), then define a B-tree to improve the random search speed. Mysql memory tables or other memory database are good choices.

User module

This module support user register, login and request token. Right now, this module is not finished.

User state module

**User state information, room information and game information are three main data structures to maintain user status. User state information can save the user current status (waiting a room, in a room or in a game).**

Use state module is the module that manage user state information.

Implementation

Right now, the server used process memory to save the user state module, and used lock to make sure the data’s correct.

Scalability

User state information can save to memory database, mysql cluster is a good choice for this.

Waiting module

When user start the game, user need to find a room to join or create a room, the server calls this stage waiting stage.

Waiting module manage the room user visited and user information. And also, this module monitor room changed message.

This module don’t need to support ACID because even some data is wrong, the influence is very small.

Implementation

This module used two dictionaries and lock to save data.

Scalability

The server can saved those two dictionaries separately, because those data is very small, so memory storage is enough. Redis cluster is a good choice.

Waiting action module

Waiting action module can support user’s action in waiting stage. Right now, **the server support load rooms, create room or join room. When a user got not enough rooms and when new room created, this module can notify the user automatically.**

Implementation

This module is a stateless module. It just operates data in room module, user state module and waiting module.

Room module

**Room is the data structure that contains room information, such as how many users in this room, the organizer of the room.**

Room module manage how to mange the room information.

This module is very similar to user state module.

Room Action module

Room action module is the module which support user action in room stage. Right now, this module support leave room, and start game command.

This module is similar to waiting action module.

Game module

**Game structure contains the game related information, such as questions, users answer result, winners.**

**Each user has separately expired time and user need to request question by a special call, this can make the game fairer.**

Other part of this module is similar to waiting module.

Game Action module

Game action module support user’s action in game stage, such as submit answer, left game.

This module is similar to room waiting action module.

Game timer module

Game timer module can check expired time and send message to game.

Implementation

The server used queue and dictionary as bucket to improve the speed.