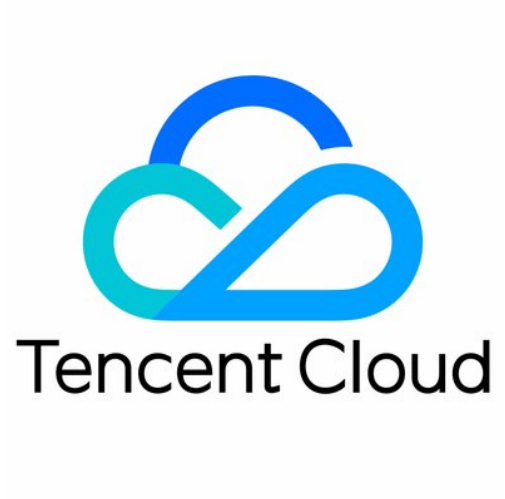
Game Server Engine

SDK Documentation

Product Documentation



Contents

SDK Documentation
Overall gRPC Integration Process

Last updated : 2021-01-21 11:10:01

- gRPC is a high-performance open-source remote procedure call (RPC) framework as well as a language/platform-neutral RPC system designed for mobile devices and HTTP/2. Currently, it supports the following programming languages: C, C++, C#, Node.js, Python, Ruby, Objective-C, PHP, Java, and Go.
- In gRPC, a client application can directly call methods on a server application on a different machine as if it was a local object. gRPC is based around the idea of defining a service, specifying the methods that can be called remotely with their parameters and return types. The server implements this interface and runs a gRPC server to handle client calls.
- Currently, the streaming RPC of gRPC is not used for connecting GSE through the gRPC method.

Note :

For more information on gRPC, please see gRPC official documentation and gRPC @ Linux Foundation.

Integrating gRPC Framework

1. Install gRPC.
2. Define the service.
3. Generate the gRPC code.
4. Integrate the game process.
5. Launch the server for GSE to call.
6. Connect the client to the gRPC server of GSE.

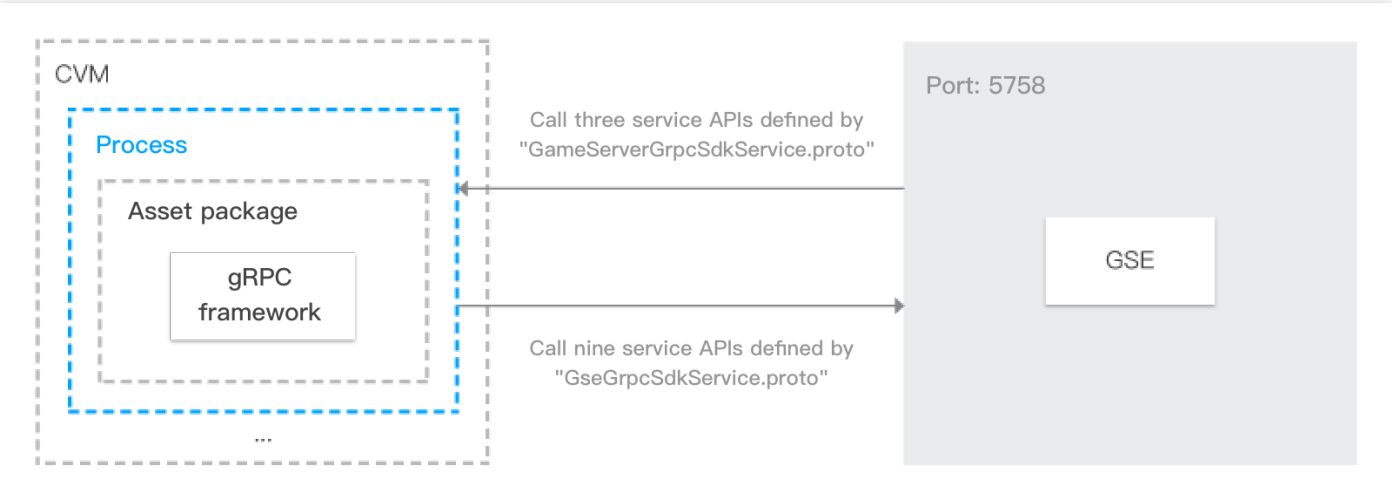
Note :

- For the specific call logic for C++, please see gRPC C++ Tutorial.
- For the specific call logic for C#, please see gRPC C# Tutorial.
- For the specific call logic for Go, please see gRPC Go Tutorial.
- For the specific call logic for Java, please see gRPC Java Tutorial.
- For the specific call logic for Lua, please see gRPC Lua Tutorial.
- For the specific call logic for Node.js, please see gRPC Node.js Tutorial.

proto File

Last updated : 2020-07-28 10:47:28

1. The game process communicates with GSE through gRPC protocol. For gRPC proto files, please see `GameServerGrpcSdkService.proto` and `GseGrpcSdkService.proto` .



2. The three service APIs defined by `GameServerGrpcSdkService.proto` are implemented by the game process, and GSE needs to call them at the appropriate time.
3. The nine service APIs defined by `GseGrpcSdkService.proto` are implemented by GSE, and the game process needs to call them at the appropriate time. The GSE API listens on gRPC at port 5758.

Note :

We provide the proto files for defining services. You can click here to directly download them with no need to generate them by yourself.

gRPC API
Callback
Health Check

Last updated : 2020-07-27 10:26:39

API Name

OnHealthCheck

API Description

This API is used for GSE to get the health status of the current game process every minute, and for the current game process to return its health status.

Request Message

```
message HealthCheckRequest {
}
```

Response Message

```
message HealthCheckResponse {
bool healthStatus = 1;
}
```

Field Description

HealthCheckResponse

Field Name	Type	Description

Field Name	Type	Description
healthStatus	bool	`true` will be returned if the process is healthy; otherwise, `false` will be returned

Sample

```
func (s *rpcService) OnHealthCheck(ctx context.Context, req *grpcsdk.HealthCheckRequest) (*grpcsdk.HealthCheckResponse, error) {
resp := &grpcsdk.HealthCheckResponse{
HealthStatus: s.healthStatus, // Identify the health status of the current process
}

return resp, nil
}
```

Receiving Game Server Session

Last updated : 2021-03-30 10:19:58

API Name

OnStartGameServerSession

API Description

This API is used for GSE to return the `GameServerSession` information to the game process after you create a `GameServerSession` by calling CreateGameServerSession or another TencentCloud API. Then, the game process needs to retain this information and call the GSE ActivateGameServerSession API in its implementation.

Request Message

```
// game server session
message GameServerSession {
string gameServerSessionId = 1;
string fleetId = 2;
string name = 3;
int32 maxPlayers = 4;
bool joinable = 5;
repeated GameProperty gameProperties = 6;
int32 port = 7;
string ipAddress = 8;
string gameServerSessionData = 9;
string matchmakerData = 10;
string dnsName = 11;
}

// Assign `GameServerSession` to the game process
message StartGameServerSessionRequest {
GameServerSession gameServerSession = 1;
}
```

Response Message

```
message GseResponse {
}
```

Field Description

GameServerSession

For more information on game session, please see GameServerSession.

Sample

```
func (s *rpcService) OnStartGameServerSession(ctx context.Context, req *grpcsdk.StartGameServerSessionRequest) (*grpcsdk.GseResponse, error) {
s.GameServerSession = req.GameServerSession // Save `GameServerSession`
conn, _ := grpc.DialContext(context.Background(), LOCAL_ADDRESS, grpc.WithInsecure())
defer conn.Close()
cli := grpcsdk.NewGseGrpcSdkServiceClient(conn)

request := &grpcsdk.ActivateGameServerSessionRequest{ // Call this API to activate the game session
GameServerSessionId: req.GameServerSession.GameServerSessionId,
MaxPlayers: req.GameServerSession.MaxPlayers,
}

return cli.ActivateGameServerSession(getContext(), request)
}
```

Ending Game Process

Last updated : 2021-01-21 11:07:49

API Name

OnProcessTerminate

API Description

This API is used for GSE to tell the game process to end itself during reduction or if health check keeps failing. After the game process receives the request, it needs to end the `GameServerSession` which it sustains on OnStartGameServerSession, and call the two GSE APIs TerminateGameServerSession and ProcessEnding to tell GSE to end the game server session and the process.

Request Message

```
// End the game process
message ProcessTerminateRequest {
int64 terminationTime = 1;
}
```

Response Message

```
message GseResponse
```

Field Description

ProcessTerminateRequest

Field Name	Type	Description
terminationTime	Int64	Time (timestamp) after which GSE will end the process.

		• If the server fleet is under full protection, the time will be ignored. • If the server fleet is under time-period protection, the time will be the protection period. • If the server fleet is under no protection, the time will be 5 minutes by default.

Sample

```
func (s *rpcService) OnProcessTerminates(ctx context.Context, req *grpcsdk.ProcessTerminateReques
t) (*grpcsdk.GseResponse, error) {
s.TerminationTime = req.TerminationTime
conn, _ := grpc.DialContext(context.Background(), LOCAL_ADDRESS, grpc.WithInsecure())
defer conn.Close()
cli := grpcsdk.NewGseGrpcSdkServiceClient(conn)

request := &grpcsdk.ProcessEndingRequest{ // Inform GSE that the process is being ended.
}

return cli.ProcessEnding(getContext(), request)
}
```

Call
Getting Process Ready

Last updated : 2021-01-21 11:08:37

API Name

ProcessReady

API Description

This API is used to notify GSE that a game process is started and ready to sustain a `GameServerSession`. Then, you can call the corresponding Tencent Cloud APIs such as `CreateGameServerSession` to generate a `GameServerSession`. Finally, GSE can call the OnStartGameServerSession API to assign the `GameServerSession` to the process.

Request Message

```
message ProcessReadyRequest {
repeated string logPathsToUpload = 1;
int32 clientPort = 2;
int32 grpcPort = 3;
}
```

Response Message

```
message GseResponse
```

Field Description

ProcessReadyRequest

Field Name	Type	Description
logPathsToUpload	String array	Path of the game process logs to be uploaded to Tencent Cloud. Directory paths and file paths are supported. GSE will upload logs in the specified paths to Tencent Cloud for you to download.
clientPort	Int32	Port to be connected to by the game client
grpcPort	Int32	Port used by GSE to call the service APIs defined by `GameServerGrpcSdkService.proto` in the game process.

Sample

```
func (r *rpcClient) ProcessReady(logPath []string, clientPort int32, grpcPort int32) (*grpcsdk.GseResponse, error) {
conn, _ := grpc.DialContext(context.Background(), LOCAL_ADDRESS, grpc.WithInsecure())
defer conn.Close()
req := &grpcsdk.ProcessReadyRequest{
LogPathsToUpload: logPath, // E.g., Absolute path to logs: `/local/game/logs`; relative path to logs: `./logs`.
ClientPort: clientPort,
GrpcPort: grpcPort,
}

client := grpcsdk.NewGseGrpcSdkServiceClient(conn)
return client.ProcessReady(getContext(), req)
}
```

Activating Game Server Session

Last updated : 2021-03-30 10:16:27

API Name

ActivateGameServerSession

API Description

This API is used to inform GSE of activating the corresponding `GameServerSession` after the game process receives a callback from GSE through the OnStartGameServerSession API.

Request Message

```
message ActivateGameServerSessionRequest{
string gameServerSessionId = 1;
int32 maxPlayers = 2;
}
```

Response Message

```
message GseResponse
```

Field Description

ActivateGameServerSessionRequest

Field Name	Type	Description
gameServerSessionId	string	`GameServerSessionId` which uniquely identifies a `GameServerSession`
maxPlayers	int32	Maximum number of players allowed to join this `GameServerSession`

Sample

For samples, see the OnStartGameServerSession API.

Receiving Player Session

Last updated : 2020-07-21 15:21:52

API Name

AcceptPlayerSession

API Description

This API is used for the game process to notify GSE that a new player has joined. GSE will then authenticate the player using the `gameServerSessionId` and `playerSessionId` parameters passed in through this API.

Request Structure

```
message AcceptPlayerSessionRequest {
string gameServerSessionId = 1;
string playerSessionId = 2;
}
```

Response Structure

```
message GseResponse
```

Field Description

AcceptPlayerSessionRequest

Field Name	Type	Description
gameServerSessionId	string	`GameServerSessionId` which uniquely identifies a `GameServerSession`

Field Name	Type	Description
playerSessionId	string	Unique ID of a player in the corresponding `GameServerSession` returned by the game developer by calling `JoinGameServerSession`

Sample

```
func (r *rpcClient) AcceptPlayerSession(gameServerSessionId, playerSessionId string) (*grpcsdk.GseResponse, error) {
conn, _ := grpc.DialContext(context.Background(), LOCAL_ADDRESS, grpc.WithInsecure())
defer conn.Close()

req := &grpcsdk.AcceptPlayerSessionRequest{
GameServerSessionId: gameServerSessionId,
PlayerSessionId: playerSessionId,
}

client := grpcsdk.NewGseGrpcSdkServiceClient(conn)
return client.AcceptPlayerSession(getContext(), req)
}
```

Removing Player Session

Last updated : 2020-07-21 15:21:52

API Name

RemovePlayerSession

API Description

This API is used for the game process to notify GSE that a player has quit. After GSE receives the request, it will update the current number of players in the corresponding game server session to allow other players to join.

Request Message

```
message RemovePlayerSessionRequest {
string gameServerSessionId = 1;
string playerSessionId = 2;
}
```

Response Message

```
message GseResponse
```

Field Description

For field definitions, see AcceptPlayerSession.

Sample

```
func (r *rpcClient) RemovePlayerSession(gameServerSessionId, playerSessionId string) (*grpcsdk.GseResponse, error) {
conn, _ := grpc.DialContext(context.Background(), LOCAL_ADDRESS, grpc.WithInsecure())
defer conn.Close()
req := &grpcsdk.RemovePlayerSessionRequest{
GameServerSessionId: gameServerSessionId,
PlayerSessionId: playerSessionId,
}

client := grpcsdk.NewGseGrpcSdkServiceClient(conn)
return client.RemovePlayerSession(getContext(), req)
}
```

Getting Player Session List

Last updated : 2020-07-21 15:21:53

API Name

DescribePlayerSessions

API Description

This API is used for the game process to get a list of player sessions that are currently in the `GameServerSession` .

Request Message

```
message DescribePlayerSessionsRequest {
string gameServerSessionId = 1;
string playerId = 2;
string playerSessionId = 3;
string playerSessionStatusFilter = 4;
string nextToken = 5;
int32 limit = 6 ;
}
```

Response Message

```
message PlayerSession {
string playerSessionId = 1;
string playerId = 2;
string gameServerSessionId = 3;
string fleetId = 4;
string ipAddress = 5;
string status = 6;
int64 creationTime = 7;
int64 terminationTime = 8;
int32 port = 9;
string playerData = 10;
```

```
string dnsName = 11;
}

message DescribePlayerSessionsResponse {
string nextToken = 1;
repeated PlayerSession playerSessions = 2;
}
```

Field Description

DescribePlayerSessionsRequest

For field definitions, see Input Parameters.

DescribePlayerSessionsResponse

For field definitions, see PlayerSession.

Sample

```
func (r *rpcClient) DescribePlayerSessions(gameServerSessionId, playerId, playerSessionId, player
SessionStatusFilter, nextToken string,limit int32) (*grpcsdk.DescribePlayerSessionsResponse, erro
r) {
conn, _ := grpc.DialContext(context.Background(), LOCAL_ADDRESS, grpc.WithInsecure())
defer conn.Close()

req := &grpcsdk.DescribePlayerSessionsRequest{
GameServerSessionId: gameServerSessionId,
PlayerId: playerId,
PlayerSessionId: playerSessionId,
PlayerSessionStatusFilter: playerSessionStatusFilter,
NextToken: nextToken,
Limit: limit,
}

client := grpcsdk.NewGseGrpcSdkServiceClient(conn)
return client.DescribePlayerSessions(getContext(), req)
}
```

Updating Player Session Creation Policy

Last updated : 2020-07-21 15:21:53

API Name

UpdatePlayerSessionCreationPolicy

API Description

This API is used for the game process to update the policy of creating player sessions in the current game server session.

Request Message

```
message UpdatePlayerSessionCreationPolicyRequest {
string gameServerSessionId = 1;
string newPlayerSessionCreationPolicy = 2;
}
```

Response Message

```
message GseResponse
```

Field Description

UpdatePlayerSessionCreationPolicyRequest

Field Name	Type	Description
gameServerSessionId	string	`GameServerSessionId` which uniquely identifies a `GameServerSession`

Field Name	Type	Description
newPlayerSessionCreationPolicy	string	Updated policy. Valid values: • **ACCEPT_ALL** (accepts all new player sessions) • **DENY_ALL** (denies all new player sessions)

Sample

```
func (r *rpcClient) UpdatePlayerSessionCreationPolicy(gameServerSessionId, newpolicy string) (*gr
pcsdk.GseResponse, error) {
conn, _ := grpc.DialContext(context.Background(), LOCAL_ADDRESS, grpc.WithInsecure())
defer conn.Close()

req := &grpcsdk.UpdatePlayerSessionCreationPolicyRequest{
GameServerSessionId: gameServerSessionId,
NewPlayerSessionCreationPolicy: newpolicy,
}

client := grpcsdk.NewGseGrpcSdkServiceClient(conn)
return client.UpdatePlayerSessionCreationPolicy(getContext(), req)
}
```

Ending Game Server Session

Last updated : 2020-07-21 15:21:53

API Name

TerminateGameServerSession

API Description

This API is used for the game process to notify GSE that a `GameServerSession` sustained by it has ended, and GSE can subsequently reassign another `GameServerSession` to the process.

Request Message

```
message TerminateGameServerSessionRequest {
string gameServerSessionId = 1;
}
```

Response Message

```
message GseResponse
```

Field Description

TerminateGameServerSessionRequest

Field Name	Type	Description
gameServerSessionId	string	`GameServerSessionId` which uniquely identifies a `GameServerSession`

Sample

```
func (r *rpcClient) TerminateGameServerSession(gameServerSessionId string) (*grpcsdk.GseResponse,
error) {
conn, _ := grpc.DialContext(context.Background(), LOCAL_ADDRESS, grpc.WithInsecure())
defer conn.Close()
req := &grpcsdk.TerminateGameServerSessionRequest{
GameServerSessionId: gameServerSessionId,
}

client := grpcsdk.NewGseGrpcSdkServiceClient(conn)
return client.TerminateGameServerSession(g.getContext(), req)
}
```

Ending Process

Last updated : 2020-07-23 14:46:33

API Name

ProcessEnding

API Description

This API is used to notify GSE that the game process is ending.

Request Structure

```
message ProcessEndingRequest {
}
```

Response Message

```
message GseResponse
```

Field Description

N/A

Sample

```
func (r *rpcClient) ProcessEnding() (*grpcsdk.GseResponse, error) {
conn, _ := grpc.DialContext(context.Background(), LOCAL_ADDRESS, grpc.WithInsecure())
defer conn.Close()
req := &grpcsdk.ProcessEndingRequest{
}
```

```
client := grpcsdk.NewGseGrpcSdkServiceClient(conn)
return client.ProcessEnding(g.getContext(), req)
}
```

Reporting Custom Data

Last updated : 2020-07-21 15:21:54

API Name

ReportCustomData

API Description

This API is used for the game process to inform GSE of custom data.

Request Message

```
message ReportCustomDataRequest {
int32 currentCustomCount = 1 ;
int32 maxCustomCount = 2;
}
```

Response Message

```
message GseResponse
```

Field Description

ReportCustomDataRequest

Field Name	Type	Description
currentCustomCount	int32	Current custom value
maxCustomCount	int32	Maximum custom value

Sample

```
func (r *rpcClient) ReportCustomData(currentCustomCount, maxCustomCount int32) (*grpcsdk.GseRespo
nse, error) {
conn, _ := grpc.DialContext(context.Background(), LOCAL_ADDRESS, grpc.WithInsecure())
defer conn.Close()
req := &grpcsdk.ReportCustomDataRequest{
CurrentCustomCount: currentCustomCount,
MaxCustomCount: maxCustomCount,
}

client := grpcsdk.NewGseGrpcSdkServiceClient(conn)
return client.ReportCustomData(getContext(), req)
}
```

General API

Last updated : 2021-01-21 11:05:36

Overview

The game process communicates with GSE through gRPC. For gRPC protocol files, please see `GameServerGrpcSdkService.proto` and `GseGrpcSdkService.proto`.

`GameServerGrpcSdkService.proto` defines three service APIs, which should be called by GSE in the game process.
The service APIs defined by `GseGrpcSdkService.proto` should be called by the game server, and the corresponding APIs should be called in the game process at the right timing. GSE APIs listen on the gRPC port 5758. You can generate protocol files in different programming languages as needed.

Note :

- The Chinese edition and English edition of the GProxy user guide are for your reference.
- The sample codes are in the Go programming language, and the protocol package is named as `grpcsdk`. For the common function `getContext`, constant `LOCAL_ADDRESS`, and message structure `GseResponse`, please see Others.

Game Process Integration Process

After the game process is started, the gRPC server and the three APIs defined by `GameServerGrpcSdkService.proto` should be implemented.

OnHealthCheck

API description

This API is used for GSE to get the health status of the current game process every minute, and for the current game process to return its health status.

Request message

```
message HealthCheckRequest {
}
```

Response message

```
message HealthCheckResponse {
bool healthStatus = 1;
}
```

Field description

HealthCheckResponse

Field Name	Type	Description
healthStatus	Bool	`true` will be returned if the process is healthy; otherwise, `false` will be returned.

Sample code

```
func (s *rpcService) OnHealthCheck(ctx context.Context, req *grpcsdk.HealthCheckRequest) (*grpcsdk.HealthCheckResponse, error) {
resp := &grpcsdk.HealthCheckResponse{
HealthStatus: s.healthStatus, // Identify the health status of the current process
}

return resp, nil
}
```

OnStartGameServerSession

API description

This API is used for GSE to return the `GameServerSession` information to the game process after you called Tencent Cloud APIs such as `CreateGameServerSession` to generate a `GameServerSession`. After receiving the request, the game process should retain the `GameServerSession` information and call the GSE API ActivateGameServerSession in its implementation.

Request message

```
// game server session
message GameServerSession {
string gameServerSessionId = 1;
string fleetId = 2;
string name = 3;
int32 maxPlayers = 4;
bool joinable = 5;
repeated GameProperty gameProperties = 6;
```

```
int32 port = 7;
string ipAddress = 8;
string gameServerSessionData = 9;
string matchmakerData = 10;
string dnsName = 11;
}

// Assign `gameserversession` to the game process
message StartGameServerSessionRequest {
GameServerSession gameServerSession = 1;
}
```

Response message

```
message GseResponse {
enum Status {
OK = 0;
ERROR_400 = 1;
ERROR_500 = 2;
}
Status status = 1;
string responseData = 2;
string errorMessage = 3;
}
```

Field description

GameServerSession

For more information on game session, please see GameServerSession.

GseResponse

Field Name	Type	Description
status	Enumeration	Return code. 0: success; 400: request error; 500: internal error
responseData	String	Returned data
errorMessage	String	Message indicating success or failure

Sample code

```
func (s *rpcService) OnStartGameServerSession(ctx context.Context, req *grpcsdk.StartGameServerSe
ssionRequest) (*grpcsdk.GseResponse, error) {
s.GameServerSession = req.GameServerSession // Save `GameServerSession`
conn, _ := grpc.DialContext(context.Background(), LOCAL_ADDRESS, grpc.WithInsecure())
```

```
defer conn.Close()
cli := grpcsdk.NewGseGrpcSdkServiceClient(conn)

request := &grpcsdk.ActivateGameServerSessionRequest{ //Call this API to activate the game sessio
n
GameServerSessionId: req.GameServerSession.GameServerSessionId,
MaxPlayers: req.GameServerSession.MaxPlayers,
}

return cli.ActivateGameServerSession(getContext(), request)
}
```

OnProcessTerminate

API description

This API is used for GSE to tell the game process to end itself during reduction or if health check keeps failing. After the game process receives the request, it needs to end the game session which it sustains on OnStartGameServerSession, and call the two GSE APIs TerminateGameServerSession and ProcessEnding to tell GSE to end the game session and the process.

Request message

```
// End the game process
message ProcessTerminateRequest {
int64 terminationTime = 1;
}
```

Response message

```
message GseResponse
```

Field description

ProcessTerminateRequest

Field Name	Type	Description
terminationTime	Int64	Time (timestamp) after which GSE will end the process. • If the server fleet is under full protection, the time will be ignored. • If the server fleet is under time-period protection, the time will be the protection period. • If the server fleet is under no protection, the time will be 5 minutes by default.

Sample code

```
func (s *rpcService) OnProcessTerminates(ctx context.Context, req *grpcsdk.ProcessTerminateReques
t) (*grpcsdk.GseResponse, error) {
s.TerminationTime = req.TerminationTime
conn, _ := grpc.DialContext(context.Background(), LOCAL_ADDRESS, grpc.WithInsecure())
defer conn.Close()
cli := grpcsdk.NewGseGrpcSdkServiceClient(conn)

request := &grpcsdk.ProcessEndingRequest{ // Inform GSE that the process is being ended
}

return cli.ProcessEnding(getContext(), request)
}
```

GSE APIs

ProcessReady

API description

This API is used to notify GSE that a game process is started and ready to sustain a game session. Then, GSE can call the OnStartGameServerSession API to assign the game session to the process.

Request message

```
message ProcessReadyRequest {
repeated string logPathsToUpload = 1;
int32 clientPort = 2;
int32 grpcPort = 3;
}
```

Response message

```
message GseResponse
```

Field description

ProcessReadyRequest

Field Name	Type	Description

logPathsToUpload	String array	Path of the game process logs to be uploaded to Tencent Cloud. GSE will upload logs in the specified paths to Tencent Cloud for you to download.
clientPort	Int32	Port to be connected to by the game client.
grpcPort	Int32	Port used by GSE to call the service APIs defined by `GameServerGrpcSdkService.proto` in the game process.

Sample request

```
func (r *rpcClient) ProcessReady(logPath []string, clientPort int32, grpcPort int32) (*grpcsdk.GseResponse, error) {
conn, _ := grpc.DialContext(context.Background(), LOCAL_ADDRESS, grpc.WithInsecure())
defer conn.Close()
req := &grpcsdk.ProcessReadyRequest{
LogPathsToUpload: logPath,
ClientPort: clientPort,
GrpcPort: grpcPort,
}

client := grpcsdk.NewGseGrpcSdkServiceClient(conn)
return client.ProcessReady(getContext(), req)
}
```

ActivateGameServerSession

API description

This API is used for a game process to tell GSE to activate the corresponding `GameServerSession` after receiving the callback from GSE through the OnStartGameServerSession API.

Request message

```
message ActivateGameServerSessionRequest{
string gameServerSessionId = 1;
int32 maxPlayers = 2;
}
```

Response message

```
message GseResponse
```

Field description

ActivateGameServerSessionRequest

Field Name	Type	Description
gameServerSessionId	String	`GameServerSessionId` which uniquely identifies a `GameServerSession`.
maxPlayers	Int32	Maximum number of players allowed by a game session.

Sample code

For the sample code, please see the OnStartGameServerSession API.

AcceptPlayerSession

API description

This API is used for the game process to notify GSE that a new player has joined. GSE will then authenticate the player using the `gameServerSessionId` and `playerSessionId` parameters passed in through this API.

Request structure

```
message AcceptPlayerSessionRequest {
string gameServerSessionId = 1;
string playerSessionId = 2;
}
```

Response structure

```
message GseResponse
```

Field description

AcceptPlayerSessionRequest

Field Name	Type	Description
gameServerSessionId	String	`GameServerSessionId` which uniquely identifies a `GameServerSession`.
playerSessionId	String	Unique ID of a player in the game session returned through the API call of `JoinGameServerSession`.

Sample code

```
func (r *rpcClient) AcceptPlayerSession(gameServerSessionId, playerSessionId string) (*grpcsdk.Gs
eResponse, error) {
conn, _ := grpc.DialContext(context.Background(), LOCAL_ADDRESS, grpc.WithInsecure())
defer conn.Close()

req := &grpcsdk.AcceptPlayerSessionRequest{
GameServerSessionId: gameServerSessionId,
PlayerSessionId: playerSessionId,
}

client := grpcsdk.NewGseGrpcSdkServiceClient(conn)
return client.AcceptPlayerSession(getContext(), req)
}
```

RemovePlayerSession

API description

This API is used for a game process to notify GSE that a player has quit. After GSE receives the request, it will update the current number of players in the corresponding game session to allow other players to join.

Request message

```
message RemovePlayerSessionRequest {
string gameServerSessionId = 1;
string playerSessionId = 2;
}
```

Response message

```
message GseResponse
```

Field description

For the sample code, please see the AcceptPlayerSession API.

Sample code

```
func (r *rpcClient) RemovePlayerSession(gameServerSessionId, playerSessionId string) (*grpcsdk.Gs
eResponse, error) {
conn, _ := grpc.DialContext(context.Background(), LOCAL_ADDRESS, grpc.WithInsecure())
defer conn.Close()
req := &grpcsdk.RemovePlayerSessionRequest{
```

```
GameServerSessionId: gameServerSessionId,
PlayerSessionId: playerSessionId,
}

client := grpcsdk.NewGseGrpcSdkServiceClient(conn)
return client.RemovePlayerSession(getContext(), req)
}
```

DescribePlayerSessions

API description

This API is used for a game process to obtain the information list of the current players in its game session.

Request message

```
message DescribePlayerSessionsRequest {
string gameServerSessionId = 1;
string playerId = 2;
string playerSessionId = 3;
string playerSessionStatusFilter = 4;
string nextToken = 5;
int32 limit = 6 ;
}
```

Response message

```
message PlayerSession {
string playerSessionId = 1;
string playerId = 2;
string gameServerSessionId = 3;
string fleetId = 4;
string ipAddress = 5;
string status = 6;
int64 creationTime = 7;
int64 terminationTime = 8;
int32 port = 9;
string playerData = 10;
string dnsName = 11;
}

message DescribePlayerSessionsResponse {
string nextToken = 1;
repeated PlayerSession playerSessions = 2;
}
```

Field description

DescribePlayerSessionsRequest

For field definitions, please see the **Input Parameters** paragraph in DescribePlayerSessions.

DescribePlayerSessionsResponse

For field definitions, please see PlayerSession.

Sample code

```
func (r *rpcClient) DescribePlayerSessions(gameServerSessionId, playerId, playerSessionId, player
SessionStatusFilter, nextToken string,limit int32) (*grpcsdk.DescribePlayerSessionsResponse, erro
r) {
conn, _ := grpc.DialContext(context.Background(), LOCAL_ADDRESS, grpc.WithInsecure())
defer conn.Close()

req := &grpcsdk.DescribePlayerSessionsRequest{
GameServerSessionId: gameServerSessionId,
PlayerId: playerId,
PlayerSessionId: playerSessionId,
PlayerSessionStatusFilter: playerSessionStatusFilter,
NextToken: nextToken,
Limit: limit,
}

client := grpcsdk.NewGseGrpcSdkServiceClient(conn)
return client.DescribePlayerSessions(getContext(), req)
}
```

UpdatePlayerSessionCreationPolicy

API description

This API is used for a game process to update the policy for players to join the current game session.

Request message

```
message UpdatePlayerSessionCreationPolicyRequest {
string gameServerSessionId = 1;
string newPlayerSessionCreationPolicy = 2;
}
```

Response message

```
message GseResponse
```

Field description

UpdatePlayerSessionCreationPolicyRequest

Field Name	Type	Description
gameServerSessionId	String	`GameServerSessionId` which uniquely identifies a `GameServerSession`.
newPlayerSessionCreationPolicy	String	Updated policy. Valid values: • **ACCEPT_ALL** (accepts all new player sessions) • **DENY_ALL** (denies all new player sessions)

Sample code

```
func (r *rpcClient) UpdatePlayerSessionCreationPolicy(gameServerSessionId, newpolicy string) (*gr
pcsdk.GseResponse, error) {
conn, _ := grpc.DialContext(context.Background(), LOCAL_ADDRESS, grpc.WithInsecure())
defer conn.Close()

req := &grpcsdk.UpdatePlayerSessionCreationPolicyRequest{
GameServerSessionId: gameServerSessionId,
NewPlayerSessionCreationPolicy: newpolicy,
}

client := grpcsdk.NewGseGrpcSdkServiceClient(conn)
return client.UpdatePlayerSessionCreationPolicy(getContext(), req)
}
```

TerminateGameServerSession

API description

This API is used for a game process to notify GSE that the game session sustained on the game process has ended. GSE can then assign other game sessions to this process.

Request message

```
message TerminateGameServerSessionRequest {
string gameServerSessionId = 1;
}
```

Response message

```
message GseResponse
```

Field description

TerminateGameServerSessionRequest

Field Name	Type	Description
gameServerSessionId	String	`GameServerSessionId` which uniquely identifies a `GameServerSession`.

Sample code

```
func (r *rpcClient) TerminateGameServerSession(gameServerSessionId string) (*grpcsdk.GseResponse,
error) {
conn, _ := grpc.DialContext(context.Background(), LOCAL_ADDRESS, grpc.WithInsecure())
defer conn.Close()
req := &grpcsdk.TerminateGameServerSessionRequest{
GameServerSessionId: gameServerSessionId,
}

client := grpcsdk.NewGseGrpcSdkServiceClient(conn)
return client.TerminateGameServerSession(g.getContext(), req)
}
```

ProcessEnding

API description

This API is used for a game process to notify GSE that the game process is closing.

Request structure

```
message ProcessEndingRequest {
}
```

Response message

```
message GseResponse
```

Field description

N/A

Sample code

```
func (r *rpcClient) ProcessEnding() (*grpcsdk.GseResponse, error) {
conn, _ := grpc.DialContext(context.Background(), LOCAL_ADDRESS, grpc.WithInsecure())
defer conn.Close()
req := &grpcsdk.ProcessEndingRequest{
}
client := grpcsdk.NewGseGrpcSdkServiceClient(conn)
return client.ProcessEnding(g.getContext(), req)
}
```

ReportCustomData

API description

This API is used for a game process to inform GSE of custom data.

Request message

```
message ReportCustomDataRequest {
int32 currentCustomCount = 1 ;
int32 maxCustomCount = 2;
}
```

Response message

```
message GseResponse
```

Field description

ReportCustomDataRequest

Field Name	Type	Description
currentCustomCount	Int32	Current custom value
maxCustomCount	Int32	Maximum custom value

Sample code

```
func (r *rpcClient) ReportCustomData(currentCustomCount, maxCustomCount int32) (*grpcsdk.GseRespo
nse, error) {
conn, _ := grpc.DialContext(context.Background(), LOCAL_ADDRESS, grpc.WithInsecure())
```

```
defer conn.Close()
req := &grpcsdk.ReportCustomDataRequest{
CurrentCustomCount: currentCustomCount,
MaxCustomCount: maxCustomCount,
}

client := grpcsdk.NewGseGrpcSdkServiceClient(conn)
return client.ReportCustomData(getContext(), req)
}
```

Others

Request `meta`

When the game process uses gRPC to call GSE APIs, you need to add two fields to `meta` of the gRPC request.

Field	Description	Type
pid	`pid` of the current game process	String
requestId	`requestId` of the current request, which is used to uniquely identify a request.	String

Sample common code

```
const (
LOCAL_ADDRESS = "127.0.0.1:5758"
)

var (
pid = strconv.Itoa(os.Getpid())
)

func getContext() context.Context {
requestId := uuid.NewV4().String() //The process generates its own `requestId`.
ctx := metadata.AppendToOutgoingContext(context.Background(), "pid", pid)
return metadata.AppendToOutgoingContext(ctx, "requestId", requestId)
}

message GseResponse {
enum Status {
OK = 0;
ERROR_400 = 1;
```

```
ERROR_500 = 2;
}
Status status = 1;
string responseData = 2;
string errorMessage = 3;
}
```