

# CHAT SERVER

---

1. Installation
2. Design
3. Code Analysis

# Index



## Installation

- TypeScript & NodeJS
- TypeScript-STL & Samchon Framework
- Examples; chat-server & chat-application



## Design

- Service
- Chat-Server
- Chat-Application



## Code Analysis

- Chat-Server
- Chat-Application

# INSTALLATION

---

1. TypeScript & NodeJS
2. TypeScript-STL & Samchon-Framework
3. Examples; chat-server & chat-application

# 1. TypeScript & NodeJS

## •이미 설치하셨는지?

- TypeScript 설치
  - `npm install -g typescript`
- NodeJS 설치
  - <https://nodejs.org/>

## 2. Packages

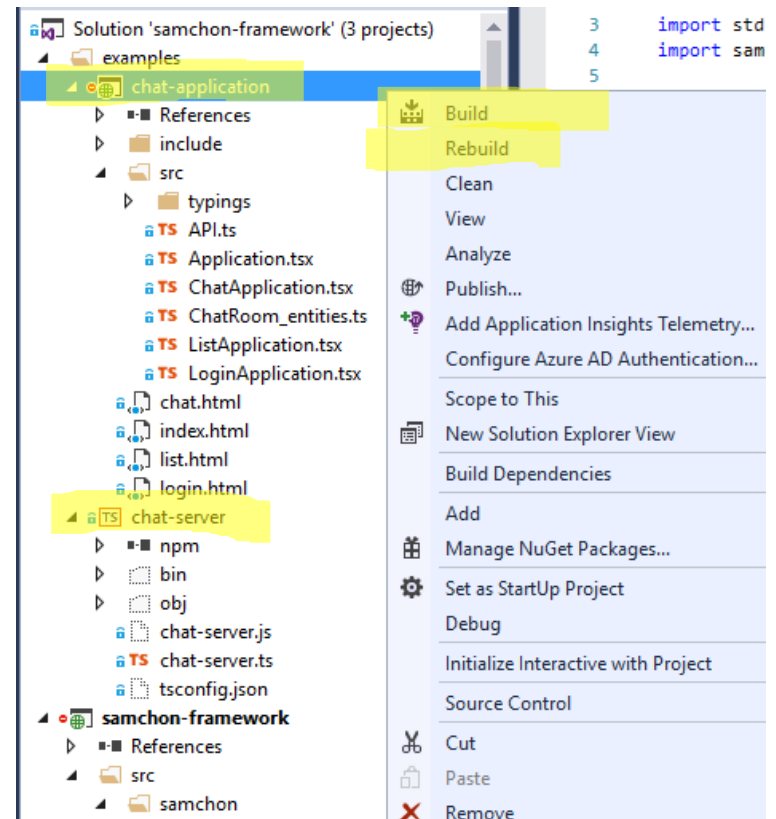
- NPM Package 설치
- `npm install -g typescript-stl`
- `npm install -g samchon-framework`

# 3. Examples

- Samchon Framework 소스 파일
- <https://github.com/samchon/framework/archive/master.zip>
- 오늘 우리가 볼 채팅 서비스는
  - ts\examples\chat-server
  - ts\examples\chat-application

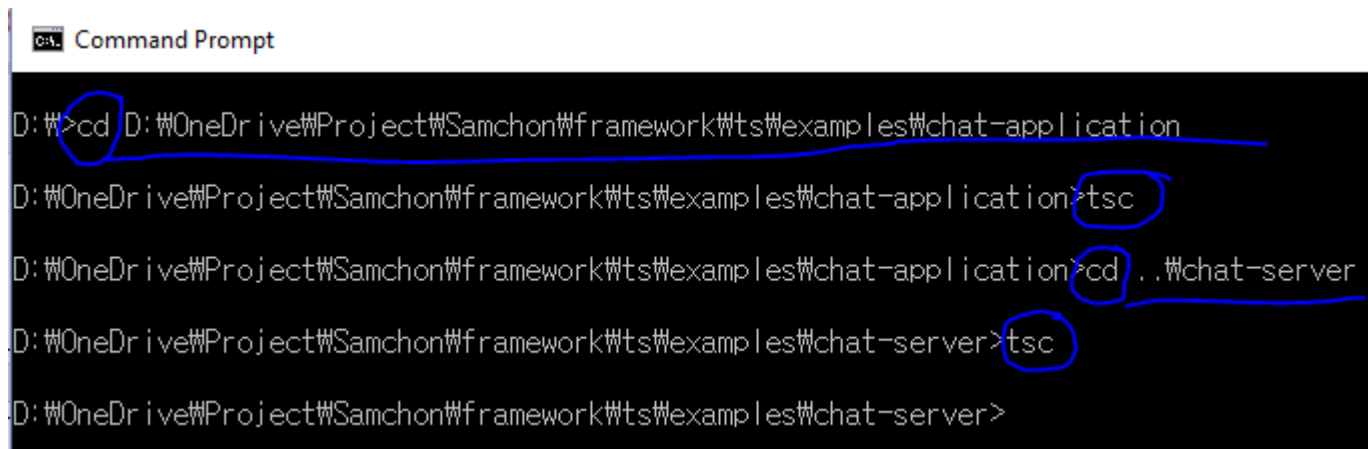
# 3. Examples

- 컴파일 – Visual Studio
- chat-application
- chat-server
- 각 프로젝트에 오른쪽 버튼 클릭
- Build 클릭
- F5를 눌러 각각 실행하면 된다



### 3. Examples

- 컴파일 – Command Line
- 각 폴더로 이동
- tsc 명령어 입력 후 엔터



```
CA: Command Prompt
D: >cd D:\OneDrive\Project\Samchon\Framework\ts\examples\chat-application
D:\OneDrive\Project\Samchon\Framework\ts\examples\chat-application>tsc
D:\OneDrive\Project\Samchon\Framework\ts\examples\chat-application>cd ..\chat-server
D:\OneDrive\Project\Samchon\Framework\ts\examples\chat-server>tsc
D:\OneDrive\Project\Samchon\Framework\ts\examples\chat-server>
```

The screenshot shows a Windows Command Prompt window with a black background and white text. The title bar reads "CA: Command Prompt". The command history is as follows: 1. The prompt is at "D:". The user enters "cd D:\OneDrive\Project\Samchon\Framework\ts\examples\chat-application". The "cd" command is circled in blue. 2. The prompt moves to "D:\OneDrive\Project\Samchon\Framework\ts\examples\chat-application". The user enters "tsc". The "tsc" command is circled in blue. 3. The prompt moves to "D:\OneDrive\Project\Samchon\Framework\ts\examples\chat-application". The user enters "cd ..\chat-server". The "cd" command is circled in blue. 4. The prompt moves to "D:\OneDrive\Project\Samchon\Framework\ts\examples\chat-server". The user enters "tsc". The "tsc" command is circled in blue. 5. The prompt moves to "D:\OneDrive\Project\Samchon\Framework\ts\examples\chat-server" and remains idle.



# 3. Examples

- 실행 – Command Line
- **서버 실행**
  - cd (소스 폴더)\ts\examples\chat-server
  - node chat-server
- **클라이언트 실행**
  - chat-client 폴더로 이동하여
  - index.html 실행

# DESIGN

---

1. Service
2. Chat Server
3. Chat Application

## Service

### Server

extends WebServer  
implements IProtocol

-session\_map: HashMap<string, User>  
-account\_map: HashMap<String, User>

+Server()  
#createUser() -> User  
#addClient(WebClientDriver)  
-erase\_user(User)  
+sendData(Invoke)  
+replyData(Invoke)

### User

extends HashMap<size\_t, Client>  
implements IProtocol

#server: Server  
-session\_id: string  
-account\_id: string  
-authority: number  
+User(Server)  
#createClient(WebClientDriver) -> Client  
-handle\_erase(CollectionEvent)  
+sendData(Invoke)  
+replyData(Invoke)  
#setAccount(string, number)

### Client

implements IProtocol

#user: User  
#service: Service  
#driver: WebClientDriver  
-no: size\_t  
+Client(User, WebClientDriver)  
#createService(string) -> Service  
+sendData(Invoke)  
+replyData(Invoke)

### Service

implements IProtocol

#client: Client  
-path: string  
+constructor(Client, string)  
+destructor()  
+sendData(Invoke)  
+replyData(Invoke)

### service::Server

Service-Server is very good for development of cloud server. You can use web or flex. I provide the libraries for implementing the cloud in the client side.

The usage is very simple. In the class Server, what you need to do is defining port number and factory method

### service::Client

It deals the network communication with client side. Just define the factory method and network I/O chain.

### service::User

ServerUser does not have any network I/O and its own special work something to do. It's a container for grouping clients by their ip and session id.

Thus, the service::User corresponds with a User (Computer) and service::Client corresponds with a Client(A browser window)

### service::Service

Most of functinos are be done in here. This Service is correspondent with a 'web browser window'.

For a cloud server, there can be enormous Service classes. Create Services for each functions and Define the functions detail in here

# 1. Service

- 소스 코드

- <https://github.com/samchon/framework/tree/master/ts/src/samchon/protocol/service>

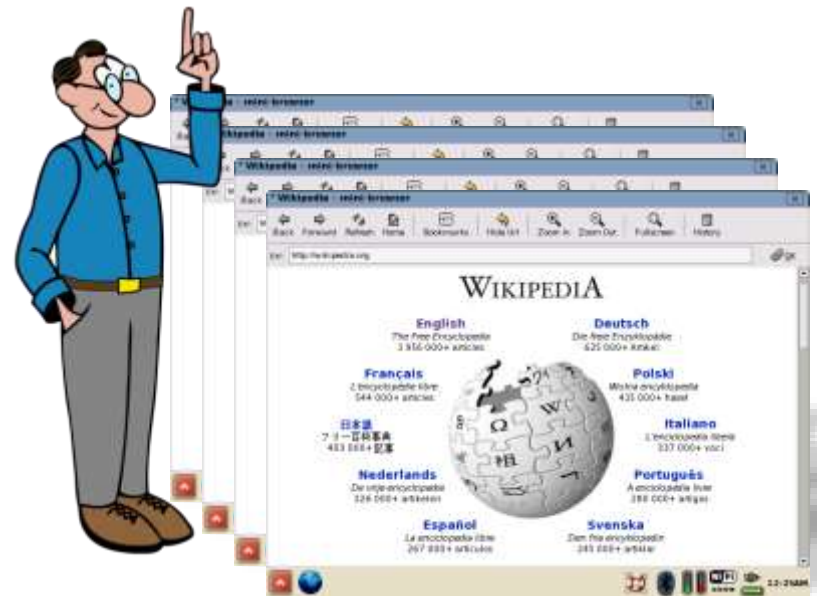
- 역할

- Server

- 문자 그대로의 서버
    - 서버를 열어 클라이언트를 받는다

- User

- 세션을 관리하는 객체
    - 여럿의 클라이언트를 가짐
      - 사용자: User
      - 한 사용자가 킨 브라우저들: Client(s)



# 1. Service

- **Client**

- 클라이언트(브라우저)와의 통신을 담당
- 하나의 Client는 하나의 Service를 가짐

- **Service**

- 유저(User)가 브라우저(Client)를 열면서
- 열린 웹 페이지 (다른 서비스)

- 클라우드 서비스에서 목표했던
- 실질적인 기능을 수행함



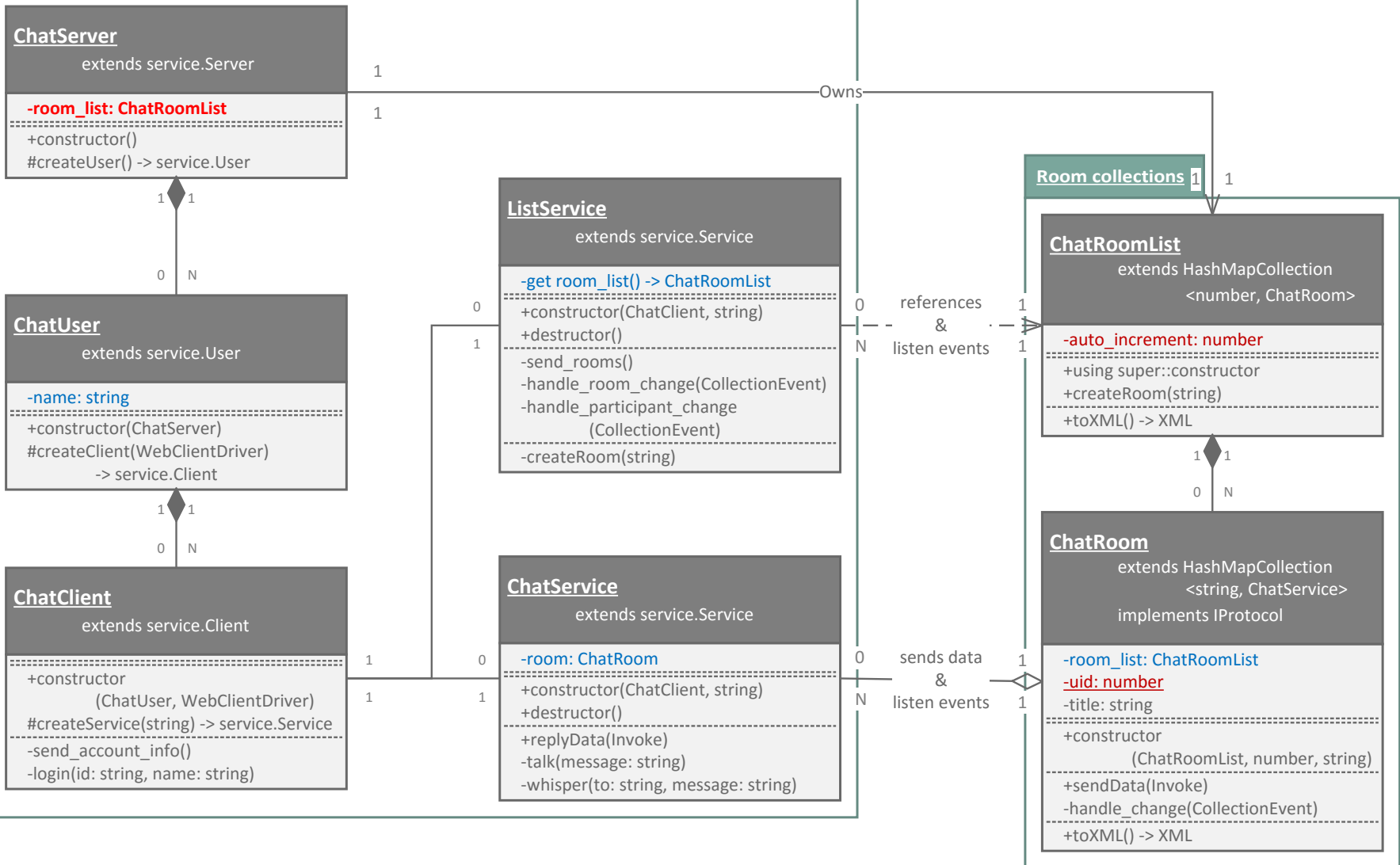
Service A



Service B

## Chat Server

### Service objects



## 2. Chat-Server; Server & User

- **ChatServer**

- `private room_list: RoomList;`
  - 채팅방 목록 클래스
- `protected createUser(): ChatUser;`
  - User를 생성하는 Factory method

- **ChatUser**

- `private name: string;`
  - 채팅 참여자의 이름 (닉네임)
- `protected createClient(driver: protocol.WebClientDriver): service.Client;`
  - Client를 생성하는 Factory method

## 2. Chat-Server; Client

### • ChatClient

- `protected createServer(path: string): service.Service;`

- Service를 생성하는 factory method
- Path는 클라이언트가 요청한 경로, 서비스 식별자
  - "list" -> `ListService`
  - "chat/3" -> `ChatServer`, 방번호: 3

- `private login(id: string, name: string): void;`

- 로그인 수행 함수
- `replyData`를 통해 호출됨
  - 즉, 클라이언트가 보낸 요청 (Invoke 메시지에 대한 응답 함수임)

```
<invoke listener="login">  
  <parameter type="string">samchon</parameter>  
  <parameter type="string">Jeongho Nam</parameter>  
</invoke>
```



## 2. Chat-Server; ListService

- **ListService**

- `private get room_list(): ChatRoomList;`
  - ChatServer.room\_list 의 shortcut
- `public destructor(): void;`
  - ListService가 소멸될 때 (클라이언트의 접속이 끊어졌을 때)
  - 수행하게 되는 작업
- `private send_rooms(): void;`
  - 현재의 채팅방 목록을 클라이언트에게 전송함
    - `this.sendData(new Invoke("setRoomList", this.room_list.toXML()));`
- `private createRoom(title: string): void;`
  - 클라이언트가 새 채팅방 개설 명령을 내렸을 때 실행되는 함수
  - `replyData()`에 의해 불림

## 2. Chat-Server; ListService

- **ListService**

- `private handle_room_change`  
(event: collection.CollectionEvent<std.Pair<number, ChatService>>);
  - ListService.room\_list (ChatRoomList)에 변동이 생겼을 때,
  - Elements I/O 이벤트가 발생했을 때
    - 새로운 방이 개설되었거나
    - 기존의 방이 없어졌거나
  - 이를 클라이언트에게 알려주기 위한 이벤트 리스너
- `private handle_participant_change`  
(event: collection.CollectionEvent<std.Pair<number, ChatService>>);
  - ChatRoom (ChatRoomList에 소속된 자식 엘리먼트)에 변동이 생겼을 때,
  - Elements I/O 이벤트가 발생했을 때,
    - 채팅방에 새로운 참여자가 생겼거나
    - 기존 참여자가 채팅방을 나갔을 때
  - 이를 클라이언트에게 알려주기 위한 이벤트 리스너

## 2. Chat-Server; ChatService

- **ChatService**

- 각 멤버변수와 메소드들이, 무슨 일을 하기 위함인지 맞춰봅시다.
- `private room: ChatRoom;`
- `public destructor(): void;`
- `private talk(message: string): void;`
- `private whisper(to: string, message: string): void;`

## 2. Chat-Server; ChatService

- 왜 ChatService에는 collection event listener 가 안 보일까?
  - `class ChatRoom extends collection.HashMapCollection<string, ChatService>;`
  - ChatRoom에서는 채팅 참여자의 서비스 객체 ChatService를 직접 보유하여
  - ChatRoom에서 collection event를 자체적으로 처리하고 있기 때문이다.
- <https://github.com/samchon/framework/blob/master/ts/examples/chat-server/chat-server.ts#L387-L401>

## Chat Application

### View - Applications

#### Application

extends React.Component  
implements IProtocol

```
#host: string
#id: string
#name: string
#communicator: WebServerConnector
+constructor()
#refresh()
+render() -> JSX.Element
#setAccount(id: string, name: string)
```

#### LoginApplication

extends Application

```
+constructor()
+render() -> JSX.Element
+static main()
-handle_login_click(MouseEvent)
-handle_connect()
-login()
#setAccount(id: string, name: string)
-handleLoginFailed(message: string)
```

#### ListApplication

extends React.Component

```
-room_list: ChatRoomList
+constructor(host: string)
+render() -> JSX.Element
#refresh()
+static main()
-create_room(MouseEvent)
-setRoomList(XML)
-setRoom(number, XML)
```

#### ChatApplication

extends React.Component

```
-room: ChatRoom
-messages: string
+constructor(host: string, uid: number)
+render() -> JSX.Element
#refresh()
+static main()
-send_message(MouseEvent)
-setRoom(XML)
-printTalk(sender: string, string)
-printWhisper
    (from: string, to:string, string)
```

### Model - Entities

#### ChatRoomList

extends EntityArray<ChatRoom>

```
+constructor()
#createChild(XML) -> ChatRoom
+TAG() -> string := "roomList"
```

1 ♦ 1  
contains

0 N

#### ChatRoom

extends EntityArray<Participant>

```
-uid: number
-title: string
+constructor()
#createChild(XML) -> Participant
+TAG() -> string := "room"
```

1 ♦ 1  
1 N

#### Participant

extends Entity

```
-id: string
-name: string
+constructor()
+TAG() -> string := "participant"
```

refers

refers

# CODE ANALYSIS

---

1. Chat-Server
2. Chat-Application

# 1. Chat-Server

- <https://github.com/samchon/framework/tree/master/ts/examples/chat-server>
- Collection에서 발생시키는
  - Elements I/O Event에 주의하여 읽을 것
- Element I/O를 발생시키는 객체 (Collection)
  - ChatRoomList
  - ChatRoom
  - ChatUser

## 2. Chat-Application

- <https://github.com/samchon/framework/tree/master/ts/examples/chat-application>
- Chat-Server에 비해 중요도는 떨어짐
  - 우리는 SDN Framework를 만드는 것이지
  - UI를 만드려는 게 아님
- 하지만, HTML 및 UI에 관심이 많다면,
  - ReactJS를 사용하니
  - HTML DOM Elements와의 상호작용에 유의하며 읽을 것



# 3. Plans

- 다음주: interaction

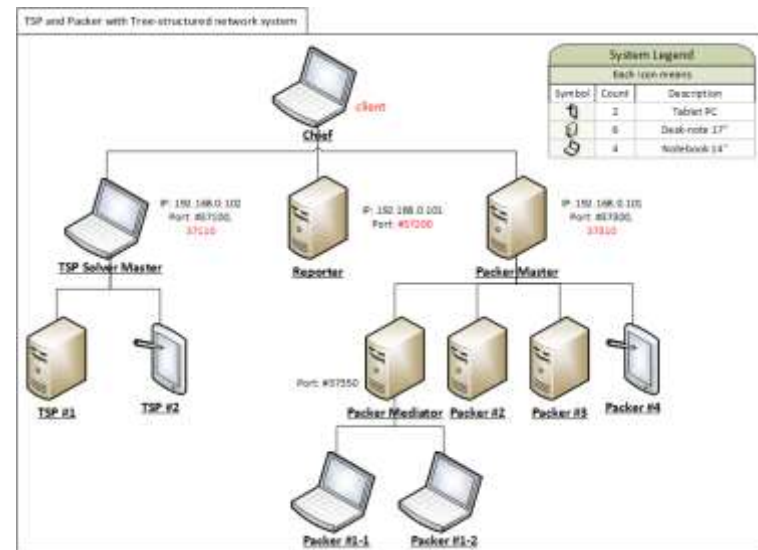
- 트리 구조의 분산-병렬처리시스템
- 오늘과 같이 설계 및 코드 분석

- 예제 코드 제작

- 이와 같이 간단한 활용예제를 제작
  - STL and Collection
  - XML
  - Entity
- 다같이 설계와 코드를 분석할 것

- 프레임워크 코드 분석

- 틸틈히 프레임워크 코드 분석
- 예제가 아닌
- 프레임워크 코드를 수정할 정도에 이르는 게 목표



# Q & A

---

2016-07-17

Samchon Framework

남 정호