National
College of
Ireland

SMART OFFICE

Distributed Systems
Baozhijie
X21186189

Contents

1 Introduction

With the developmentof technology smart office is a modern office environment that is designed to enhance productivity and collaboration through the integration of technology and automation. One of the key features of a smart office is the use of different services that are powered by Remote Procedure Calls (RPCs).

The Employee Directory Service allows employees to easily access information about their colleagues, including contact information and job title, through the use of RPCs such as GetEmployeeSystem and ListEmployeesSystem. This service makes it easier for employees to connect and collaborate with each other, ultimately increasing efficiency and productivity.

The Printer Service is another important component of a smart office, providing a convenient way for employees to print documents through the use of RPCs like PrintDocumentSystem and ListPrintJobsSystem. This service helps to streamline printing processes and reduce the amount of time employees spend managing print jobs.

Lighting services are another important feature of smart offices, allowing employees to remotely control any number of lights in any office through RPC.

Employees can not only remotely switch on and off any number of lights in any office, but also check the status of any light in any office, which is in line with the concept of energy conservation and environmental protection in future offices.

By implementing these RPC driven services, smart offices can provide employees with a more streamlined and efficient work environment, ultimately improving workplace productivity and collaboration capabilities

2 Employee Service

2.1 EmployeeService.proto

```
service EmployeeService {
   rpc AddEmployee (AddEmployeeRequest) returns (AddEmployeeResponse) {}
   rpc GetEmployee (GetEmployeeRequest) returns (GetEmployeeResponse) {}
   rpc UpdateEmployee (stream UpdateEmployeeRequest) returns (stream
UpdateEmployeeResponse) {}
}

message AddEmployeeRequest {
   string name = 1;
   int32 age = 2;
   string department = 3;
}

message AddEmployeeResponse {
   int32 id = 1;
}

message GetEmployeeRequest {
   int32 id = 1;
}

message GetEmployeeResponse {
   string name = 1;
   int32 age = 2;
   string department = 3;
}

message UpdateEmployeeRequest {
   int32 id = 1;
   Employee employee = 2;
}

message UpdateEmployeeResponse {
   bool success = 1;
}

message Employee {
   string name = 1;
   int32 age = 2;
   string department = 3;
}
```

2.2 EmployeeService.proto Code Analysis

The EmployeeService.proto file defines a gRPC service and some related messages. Here is an analysis of the rpc services and parameters in it:

AddEmployee: This is an rpc service to add a new employee, which takes an AddEmployeeRequest message as input parameter and returns an AddEmployeeResponse message. The AddEmployeeRequest message contains the employee's id, name, age, and department information, and the AddEmployeeResponse message only contains the employee's id.

GetEmployee: This is an rpc service to retrieve employee information, which takes a GetEmployeeRequest message as input parameter and returns a GetEmployeeResponse message. The GetEmployeeRequest message only contains the id of the employee to be queried, and the GetEmployeeResponse message contains the employee's id, name, age, and department information.

UpdateEmployee: This is an rpc service to update employee information, which uses streaming. It takes one or more UpdateEmployeeRequest messages as input stream and returns one or more UpdateEmployeeResponse messages as output stream. The UpdateEmployeeRequest message contains the employee's id and an Employee message, which contains all the information of the employee to be updated. The UpdateEmployeeResponse message only contains a boolean value success, indicating whether the update was successful.

2.3 EmployeeService.Server Code Analysis

using gRPC. The following are analyses of jmdns and Remote Error Handling & Advanced gRPC:

jmdns: In the start method, a JmDNS instance is created using JmDNS.create, and the service information is registered so that the client can call the service through service discovery. In the stop method, the service information is unregistered and the JmDNS instance is closed.

Remote Error Handling: In the getEmployee and updateEmployee methods in the EmployeeServiceImpl class, the responseObserver.onError method is used to handle exceptional cases and return a status message with an error description to the client. In the updateEmployee method, the streaming method is used to handle exceptional cases in the client stream.

Cancelling of messages: In the updateEmployee method, the responseObserver.onError method is used to handle the case where the client cancels a message, preventing the server from being blocked while waiting for requests.

server: The gRPC server object is created using ServerBuilder. The EmployeeServiceImpl instance is added to the service using the addService method. The start method starts the service.

jmdns: The JmDNS object is used for service registration and discovery. In the start method, a JmDNS object is created and the service information is registered. In the stop method, the service information is unregistered and the JmDNS object is closed.

employeeMap: The ConcurrentMap object is used to store employee information.

prop: The Properties object is used to store service configuration information. It is obtained using the getProperties method.

main method: The entry point of the program. The service is started using the EmployeeServiceServer instance, and the service is stopped when it is closed.

start method: Starts the service. Gets the service configuration information, creates and starts the gRPC service, and registers the service information. The service is closed using the addShutdownHook method when it is stopped.

stop method: Closes the service and the JmDNS object.

blockUntilShutdown method: Blocks the main thread and waits for the service to close.

getProperties method: Gets the service configuration information from the configuration file.

EmployeeServiceImpl class: Implements the EmployeeServiceGrpc.EmployeeServiceImplBase class and overrides the addEmployee, getEmployee, and updateEmployee methods to handle client requests.

addEmployee method: Adds a new employee. Stores the employee information in the employeeMap object and returns the employee's id.

getEmployee method: Gets employee information. Searches for the corresponding employee information in the employeeMap object using the employee's id. If found, returns all the employee information; otherwise, returns an error status message.

updateEmployee method: Updates employee information. Uses streaming to accept one or more UpdateEmployeeRequest messages as input streams. Handles client requests in the input stream and updates employee information. If the update is successful, returns a boolean success value; if the update fails, returns an error status message.

Overall, this code implements an Employee service server based on gRPC, which provides three rpc services for adding, retrieving, and updating employee information. The service registration and discovery is implemented through JmDNS. In addition, appropriate error handling mechanisms and streaming methods are used to ensure service reliability and performance.

2.4 EmployeeService ClientGUI Code Analysis

This code is an implementation of a client for an employee information management system based on the gRPC protocol. It allows users to add, update, and view employee information. Here is an analysis of the code:

Firstly, in the EmployeeServiceClientGUI class, the main() method starts the client by creating an EmployeeServiceClientGUI instance and calling the discoverService() method and createAndShowGUI() method.

The discoverService() method implements service discovery using the JmDNS library and obtains the service host and port from the service information.

The createAndShowGUI() method creates a simple Swing user interface where the user can enter employee information and add it to the system by clicking the "Add Employee" button. The user can also view the information of an employee by entering their ID and clicking the "Get Employee BY ID" button. The user can also update employee information by clicking the "Update Employee" button.

In the connectToServer() method, the client creates a gRPC channel and uses it to construct synchronous stub (blockingStub) and asynchronous stub (asyncStub).

In the disconnectFromServer() method, the client closes the gRPC channel to disconnect from the server.

In the addEmployee() method, the client uses the stub to send an add employee request to the server and returns the server response.

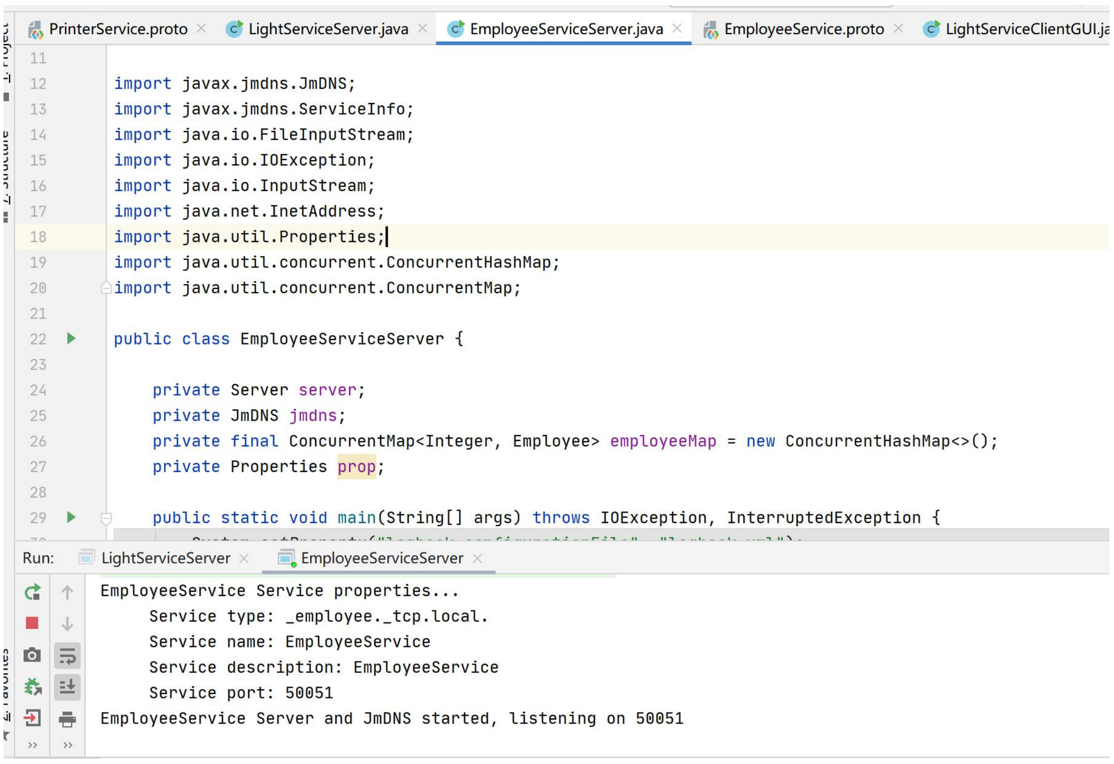
In the getEmployee() method, the client uses the stub to send a request to retrieve employee information from the server and returns the server response. If the request fails, null is returned.

In the updateEmployee() method, the client uses the asynchronous stub to send an update employee request to the server and uses CountDownLatch to implement asynchronous notification of waiting for a response. If the update is successful, true is returned, otherwise false is returned.

Overall, this code implements a client for an employee information management system based on the gRPC protocol. It uses the JmDNS library to implement service discovery, uses gRPC stubs to handle communication with the server, and uses a Swing user interface to interact with the user.

2.5 Screenshot of running results of EmployeeService

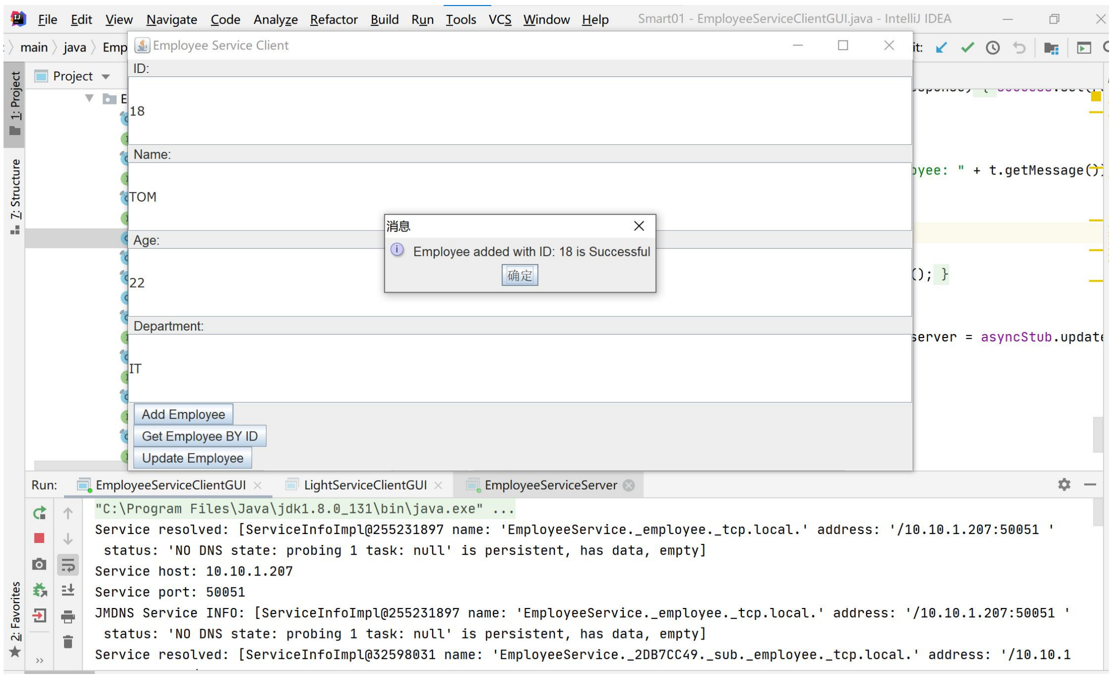
2.5.1 EmployeeServiceServer



```
import javax.jmdns.JmDNS;
import javax.jmdns.ServiceInfo;
import java.io.FileInputStream;
import java.io.IOException;
import java.io.InputStream;
import java.net.InetAddress;
import java.util.Properties;
import java.util.concurrent.ConcurrentHashMap;
import java.util.concurrent.ConcurrentMap;

public class EmployeeServiceServer {

    private Server server;
    private JmDNS jmdns;
    private final ConcurrentMap<Integer, Employee> employeeMap = new ConcurrentHashMap<>();
    private Properties prop;

    public static void main(String[] args) throws IOException, InterruptedException {
```

```
EmployeeService Service properties...
     Service type: _employee._tcp.local.
     Service name: EmployeeService
     Service description: EmployeeService
     Service port: 50051
EmployeeService Server and JmDNS started, listening on 50051
```

2.5.2 EmployeeService ClientGUI



ID:
18

Name:
TOM

Age:
22

Department:
IT

Employee added with ID: 18 is Successful

```
"C:\Program Files\Java\jdk1.8.0_131\bin\java.exe" ...
Service resolved: [ServiceInfoImpl@255231897 name: 'EmployeeService._employee._tcp.local.' address: '/10.10.1.207:50051 '
 status: 'NO DNS state: probing 1 task: null' is persistent, has data, empty]
Service host: 10.10.1.207
Service port: 50051
JMDNS Service INFO: [ServiceInfoImpl@255231897 name: 'EmployeeService._employee._tcp.local.' address: '/10.10.1.207:50051 '
 status: 'NO DNS state: probing 1 task: null' is persistent, has data, empty]
Service resolved: [ServiceInfoImpl@32598031 name: 'EmployeeService._2DB7CC49._sub._employee._tcp.local.' address: '/10.10.1
```

3 PrinterService

3.1 PrinterService.proto

```
service PrinterService {
rpc PrintDocument (PrintDocumentRequest) returns (PrintDocumentResponse);
//Add a URL for printing files
rpc ListPrintJobs (ListPrintJobsRequest) returns (stream ListPrintJobsResponse);
///Add a URL for printing files
rpc CancelPrintJob (CancelPrintJobRequest) returns (CancelPrintJobResponse);
}
message PrintDocumentRequest {
string document_ url = 1;
}
message PrintDocumentResponse {
bool success = 1;
}
message ListPrintJobsRequest {}
message ListPrintJobsResponse {
string document_ url = 1;
int32 pages = 2;
}
message CancelPrintJobRequest {
string document_ url = 1;
}
message CancelPrintJobResponse {
bool success = 1;
}
```

The code defines a gRPC service named PrinterService with three methods:

PrintDocument: adds a printing file and receives a PrintDocumentRequest that contains the URL of the file to be printed, and returns a PrintDocumentResponse indicating whether the file was added successfully.

ListPrintJobs: lists all the files currently being printed, receives a ListPrintJobsRequest with no parameters, and returns a stream of ListPrintJobsResponse, with each response containing the URL and number of pages of a printing job.

CancelPrintJob: cancels a printing job, receives a CancelPrintJobRequest that includes the URL of the file to be canceled, and returns a CancelPrintJobResponse indicating whether the cancellation was successful.

Each message includes several fields:

PrintDocumentRequest contains a document_url field, indicating the URL of the file to be added for printing.

PrintDocumentResponse contains a success field, indicating whether the printing was successful.

ListPrintJobsResponse contains a document_url field, indicating the URL of a printing job, and a pages field, indicating the number of pages of the job.

CancelPrintJobRequest contains a document_url field, indicating the URL of the file to be canceled.

CancelPrintJobResponse contains a success field, indicating whether the cancellation was successful.

3.2 PrinterService server:

This server-side code implements the three RPC methods defined in PrinterService. Specifically, the PrintDocument method adds the URL of the printing file to a list, the ListPrintJobs method returns a list of print jobs, and the CancelPrintJob method cancels a print job. Additionally, this server-side code registers the service using JmDNS and starts the gRPC server. An AuthInterceptor class is also defined in the server code to authenticate gRPC requests.

If the authentication fails, the server will reject the request. In this implementation, the authentication logic is based on a simple string match, meaning that the request is only allowed to pass through if the authentication token in the request matches the valid authentication token specified by the server.

3.3PrinterServiceGUI :

The PrinterServiceGUI client code implements a simple graphical user interface to interact with the gRPC service. It includes the following components:

PrinterServiceGUI class: used to start the application, discover the service, and create the GUI interface.

discoverService method: uses the JmDNS library to discover the service and obtain the service host and port.

createAndShowGUI method: creates the GUI interface, including three buttons for adding a print document URL, listing print jobs, and canceling a print job.

connectToServer method: creates a gRPC channel using the host and port, and adds authentication metadata.

disconnectFromServer method: closes the gRPC channel.

printDocument method: adds a print document URL, sends a gRPC request, and receives a response.

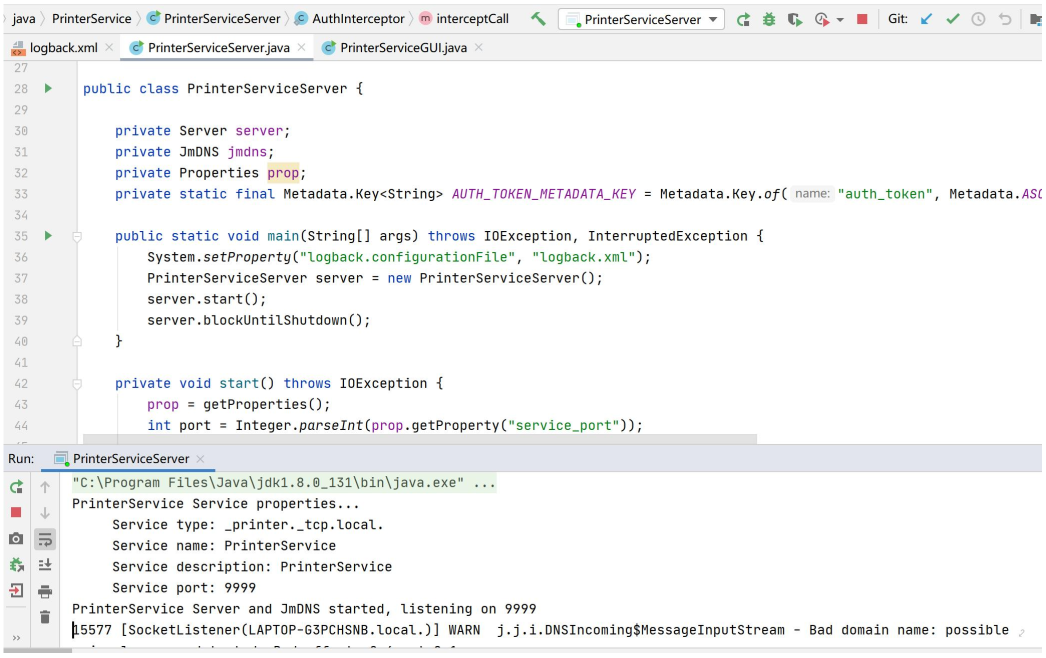
listPrintJobs method: lists print jobs, sends a gRPC request, and receives a response.

cancelPrintJob method: cancels a print job, sends a gRPC request, and receives a response.

The GUI client code interacts with the gRPC service by creating the GUI interface and listening to button click events. The connectToServer method creates a gRPC channel using the host and port, and adds authentication metadata, while the disconnectFromServer method closes the gRPC channel. The printDocument, listPrintJobs, and cancelPrintJob methods send gRPC requests and receive responses, which respectively add a print document URL, list print jobs, and cancel a print job.

3.4 Screenshot of running results of PrinterService

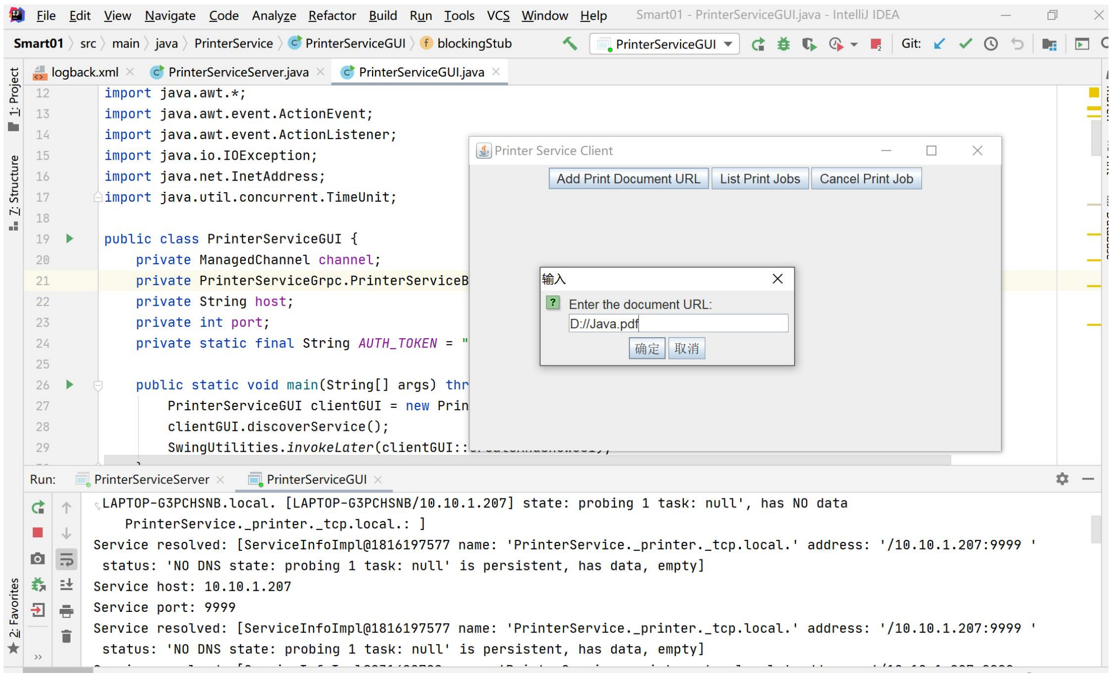
3.4.1 PrinterServiceserver:



```
public class PrinterServiceServer {

    private Server server;
    private JmDNS jmdns;
    private Properties prop;
    private static final Metadata.Key<String> AUTH_TOKEN_METADATA_KEY = Metadata.Key.of( name: "auth_token", Metadata.ASC

    public static void main(String[] args) throws IOException, InterruptedException {
        System.setProperty("logback.configurationFile", "logback.xml");
        PrinterServiceServer server = new PrinterServiceServer();
        server.start();
        server.blockUntilShutdown();
    }

    private void start() throws IOException {
        prop = getProperties();
        int port = Integer.parseInt(prop.getProperty("service_port"));
```

```
"C:\Program Files\Java\jdk1.8.0_131\bin\java.exe" ...
PrinterService Service properties...
     Service type: _printer._tcp.local.
     Service name: PrinterService
     Service description: PrinterService
     Service port: 9999
PrinterService Server and JmDNS started, listening on 9999
15577 [SocketListener(LAPTOP-G3PCHSNB.local.)] WARN  j.j.i.DNSIncoming$MessageInputStream - Bad domain name: possible
```

3.4.2PrinterServiceGUI :



```
import java.awt.*;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.io.IOException;
import java.net.InetAddress;
import java.util.concurrent.TimeUnit;

public class PrinterServiceGUI {
    private ManagedChannel channel;
    private PrinterServiceGrpc.PrinterServiceB
    private String host;
    private int port;
    private static final String AUTH_TOKEN = "

    public static void main(String[] args) thr
        PrinterServiceGUI clientGUI = new Prin
        clientGUI.discoverService();
        SwingUtilities.invokeLater(clientGUI::
```

Printer Service Client

输入

Enter the document URL:

D://Java.pdf

```
LAPTOP-G3PCHSNB.local. [LAPTOP-G3PCHSNB/10.10.1.207] state: probing 1 task: null', has NO data
    PrinterService._printer._tcp.local.: ]
Service resolved: [ServiceInfoImpl@1816197577 name: 'PrinterService._printer._tcp.local.' address: '/10.10.1.207:9999 '
 status: 'NO DNS state: probing 1 task: null' is persistent, has data, empty]
Service host: 10.10.1.207
Service port: 9999
Service resolved: [ServiceInfoImpl@1816197577 name: 'PrinterService._printer._tcp.local.' address: '/10.10.1.207:9999 '
 status: 'NO DNS state: probing 1 task: null' is persistent, has data, empty]
```

4 LightService

4.1 LightService.proto

```
service LightService {
//Control the on/off status of a light
rpc ControlLight(ControlLightRequest) returns (ControlLightResponse);
//Obtain the status of all lights in a room
rpc GetRoomLights(GetRoomLightsRequest) returns (GetRoomLightsResponse);
//Control the on/off status of multiple lights
rpc ControlMultipleLights(stream ControlMultipleLightsRequest) returns
(ControlMultipleLightsResponse);
}
message ControlLightRequest {
string light_ id = 1;
bool turn_ on = 2;
}
message ControlLightResponse {
bool success = 1;
}
message GetRoomLightsRequest {
string room_ id = 1;
}
message Light {
string id = 1;
bool is_ on = 2;
}
message GetRoomLightsResponse {
repeated Light lights = 1;
}
message ControlMultipleLightsRequest {
string light_ id = 1;
bool turn_ on = 2;
}
message ControlMultipleLightsResponse {
bool success = 1;
}
```

The LightServicecode defines a gRPC service , which includes the following three RPC methods:

ControlLight: Controls the on/off status of a light. Its input parameter is ControlLightRequest, which contains the ID and on/off status of a light, and its return value is ControlLightResponse, indicating whether the operation is successful.

GetRoomLights: Retrieves the status of all lights in a room. Its input parameter is

GetRoomLightsRequest, which contains the ID of a room, and its return value is GetRoomLightsResponse, where the lights field is a repeated list of Light objects, representing all the lights in the room and their statuses.

ControlMultipleLights: Controls the on/off status of multiple lights. It uses stream semantics, indicating that information on the lights to be controlled can be sent one by one through a stream. Its input parameter is ControlMultipleLightsRequest, which contains the ID and on/off status of a light, and its return value is ControlMultipleLightsResponse, indicating whether the operation is successful.

Each RPC method takes one or more protocol buffer messages as input and output, corresponding to the ControlLightRequest/ControlLightResponse, GetRoomLightsRequest/GetRoomLightsResponse, and ControlMultipleLightsRequest/ControlMultipleLightsResponse messages in the code above. The Light message represents a single light, containing its ID and on/off status.

4.2 LightServiceServer

LightServiceServer implements a gRPC server that uses Protocol Buffers for data transfer. In the code, LightService defines three RPC methods for controlling the on/off state of a single light, getting the status of all lights in a room, and controlling the on/off state of multiple lights. Each method has corresponding request and response message formats.

The LightServiceServer class implements the server's startup and shutdown, including creating the server and listening on a specified port, registering the service with a local mDNS server, and blocking the thread to wait for termination signals. Additionally, the LightServiceImpl class inherits the generated gRPC code and implements the specific logic for the three defined RPC methods. Controlling the on/off state of a single light uses ConcurrentHashMap to store the light's state; getting the status of all lights in a room directly traverses ConcurrentHashMap to find the matching lights; controlling the on/off state of multiple lights uses stream RPC and records the number of successful and total requests before returning the number of successful requests.

The entire code implements a server based on gRPC and Protocol Buffers, which can remotely control the state of lights. The server automatically registers with a local JmDNS server upon startup, and clients can discover the service and call RPC methods through mDNS without knowing the specific IP and port of the server.

4.3 LightServiceClientGUI

The LightServiceClientGUI code implements a simple user interface that allows users to send gRPC requests to the server by filling in text boxes and selecting checkboxes.

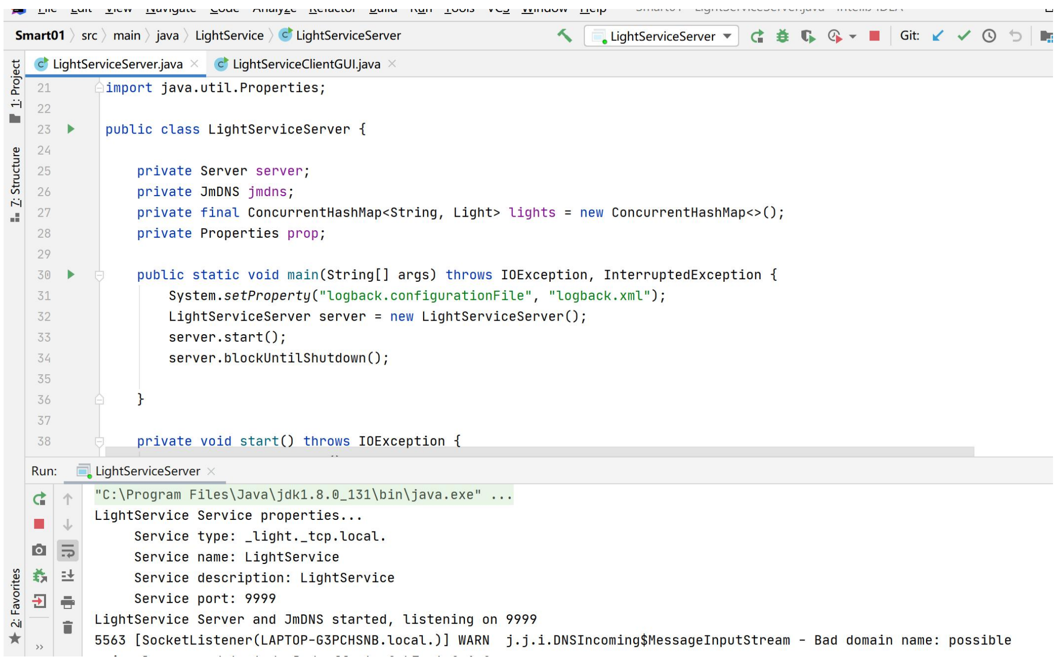
At program startup, the client initializes a JmDNS instance to listen for service discovery events and establishes a connection with the server when a service is discovered. Then, the client creates a JFrame object to display the user interface, where users can control the on/off state of lights and get the status of all lights in the room.

The client program uses blocking/async stubs to call the server's RPC methods. Depending on the RPC method called, it sends ControlLightRequest, GetRoomLightsRequest, or ControlMultipleLightsRequest to the server and displays the response in a dialog box to the user after receiving it.

It is important to note that the client program is implemented using Java Swing

4.4 Screenshot of running results of LightService

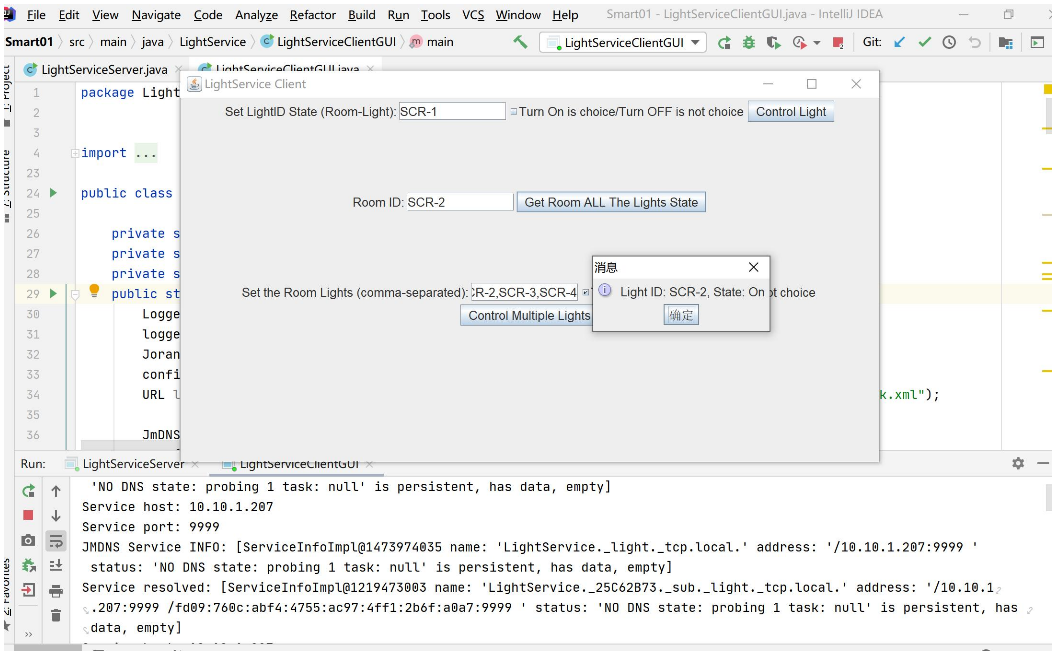
4.1LightServiceServer


Smart01 〉 src 〉 main 〉 java 〉 LightService 〉 LightServiceServer
LightServiceServer.java
LightServiceClientGUI.java
Git:
1: Project
Z: Structure
2: Favorites
Run:
LightServiceServer

```
import java.util.Properties;

public class LightServiceServer {

    private Server server;
    private JmDNS jmdns;
    private final ConcurrentHashMap<String, Light> lights = new ConcurrentHashMap<>();
    private Properties prop;

    public static void main(String[] args) throws IOException, InterruptedException {
        System.setProperty("logback.configurationFile", "logback.xml");
        LightServiceServer server = new LightServiceServer();
        server.start();
        server.blockUntilShutdown();

    }

    private void start() throws IOException {
```

```
"C:\Program Files\Java\jdk1.8.0_131\bin\java.exe" ...
LightService Service properties...
     Service type: _light._tcp.local.
     Service name: LightService
     Service description: LightService
     Service port: 9999
LightService Server and JmDNS started, listening on 9999
5563 [SocketListener(LAPTOP-G3PCHSNB.local.)] WARN  j.j.i.DNSIncoming$MessageInputStream - Bad domain name: possible
```

4.2LightServiceClientGUI


File Edit View Navigate Code Analyze Refactor Build Run Tools VCS Window Help
Smart01 - LightServiceClientGUI.java - IntelliJ IDEA
Smart01 〉 src 〉 main 〉 java 〉 LightService 〉 LightServiceClientGUI 〉 main
LightServiceClientGUI
Git:
LightServiceServer.java
LightServiceClientGUI.java
LightService Client
消息
确定
Run:
LightServiceServer
LightServiceClientGUI

```
package Light

import ...

public class

    private s
    private s
    private s
    public st
        Logge
        logge
        Joran
        confi
        URL l                                                    k.xml");

        JmDNS
```

Set LightID State (Room-Light): SCR-1 ☐ Turn On is choice/Turn OFF is not choice [Control Light]

Room ID: SCR-2 [Get Room ALL The Lights State]

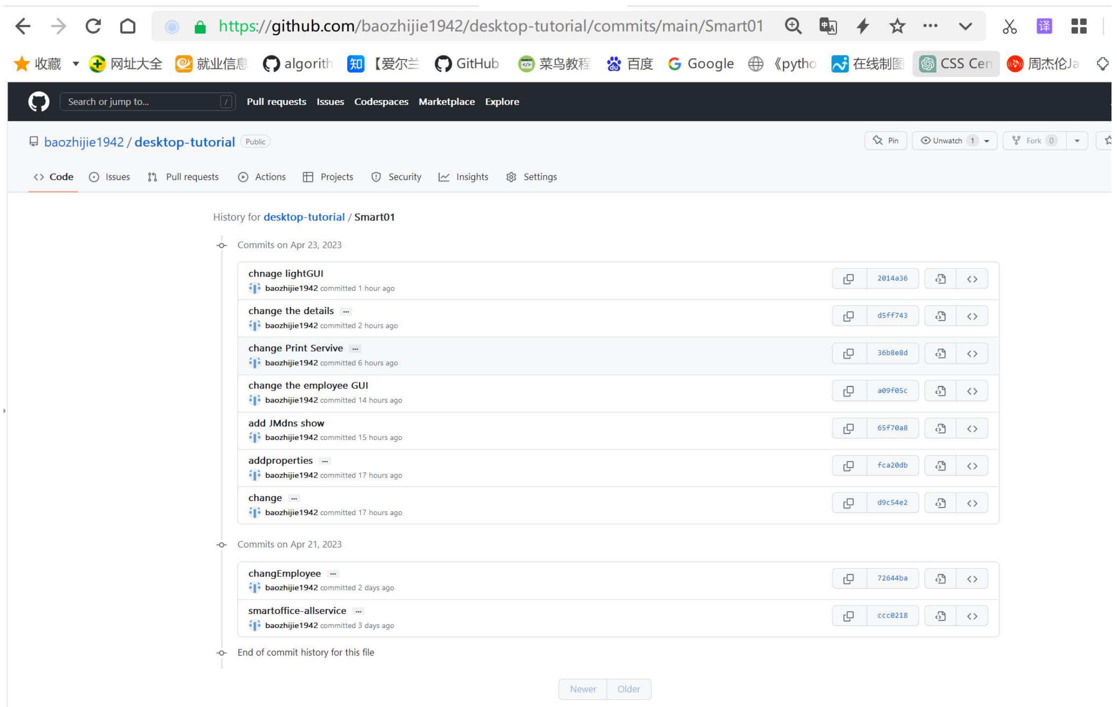
Set the Room Lights (comma-separated): ;R-2,SCR-3,SCR-4 ☑ ...t choice [Control Multiple Lights]

Light ID: SCR-2, State: On

```
 'NO DNS state: probing 1 task: null' is persistent, has data, empty]
Service host: 10.10.1.207
Service port: 9999
JMDNS Service INFO: [ServiceInfoImpl@1473974035 name: 'LightService._light._tcp.local.' address: '/10.10.1.207:9999 '
 status: 'NO DNS state: probing 1 task: null' is persistent, has data, empty]
Service resolved: [ServiceInfoImpl@1219473003 name: 'LightService._25C62B73._sub._light._tcp.local.' address: '/10.10.1
.207:9999 /fd09:760c:abf4:4755:ac97:4ff1:2b6f:a0a7:9999 ' status: 'NO DNS state: probing 1 task: null' is persistent, has
data, empty]
```

5 GitHub Record

5.1 My Github Record



https://github.com/baozhijie1942/desktop-tutorial/commits/main/Smart01

History for desktop-tutorial / Smart01

Commits on Apr 23, 2023

chnage lightGUI baozhijie1942 committed 1 hour ago	2014a36
change the details … baozhijie1942 committed 2 hours ago	d5ff743
change Print Servive … baozhijie1942 committed 6 hours ago	36b8e8d
change the employee GUI baozhijie1942 committed 14 hours ago	a09f05c
add JMdns show baozhijie1942 committed 15 hours ago	65f70a8
addproperties … baozhijie1942 committed 17 hours ago	fca20db
change … baozhijie1942 committed 17 hours ago	d9c54e2

Commits on Apr 21, 2023

changEmployee … baozhijie1942 committed 2 days ago	72644ba
smartoffice-allservice … baozhijie1942 committed 3 days ago	ccc0218

End of commit history for this file

5.2 My Github URL

https://github.com/baozhijie1942/desktop-tutorial/tree/main/Smart01