分布式计算 通信技术 第二次作业

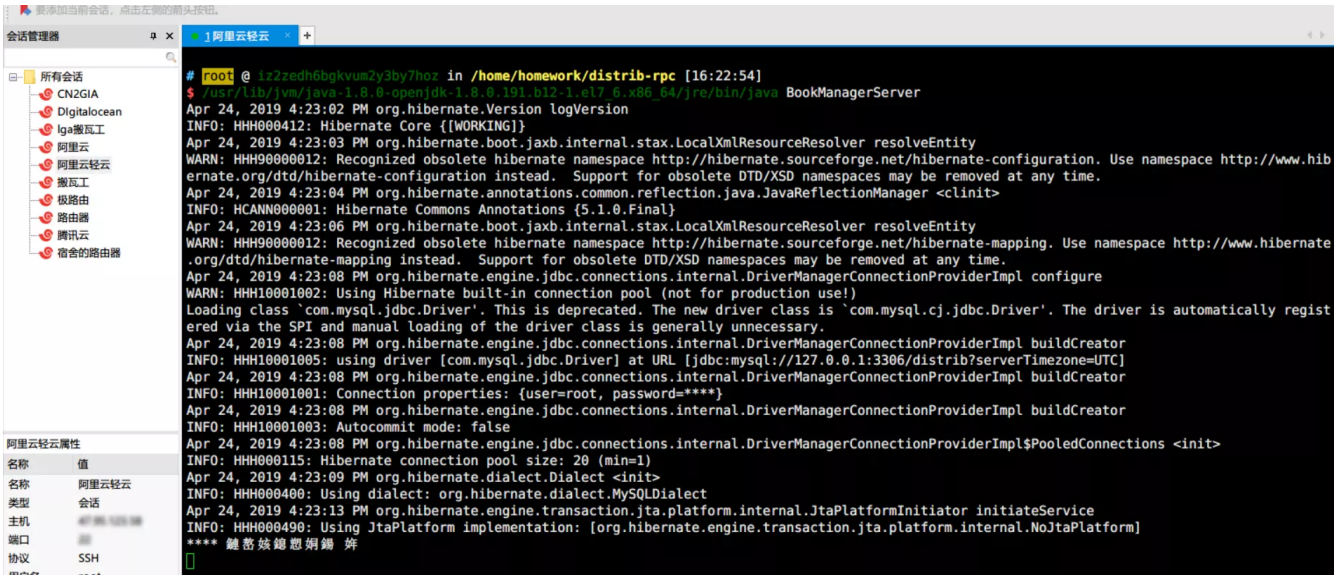
> **题目**
>
> 利用RPC/RMI技术实现一个书籍信息管理系统，具体要求：
>
> 1.客户端实现用户交互，服务器端实现书籍信息存储和管理。客户端与服务器端利用RPC/RMI机制进行协作。中间件任选。
>
> 2.服务器端至少暴露如下RPC/RMI接口：
>
> l bool add(Book b) 添加一个书籍对象。
>
> l Book queryByID(int bookID) 查询指定ID号的书籍对象。
>
> l BookList queryByName(String name) 按书名查询符合条件的书籍对象列表，支持模糊查询。
>
> l bool delete((int bookID) 删除指定ID号的书籍对象。

结果

服务器的部署

服务端 Java 程序编写完成之后，进行了打包操作，打包后的 Jar 文件约为 20M， 可以直接通过 `java -jar` 方式运行。

将 jar 包上传至阿里云服务器，运行，服务默认运行在 50051 端口



客户端生成

使用 electron 打包工具，可以生成跨平台的前端界面


ELECTRON RPC LIBRARY
书籍查询
通过 书籍ID 查询图书信息
通过 书籍名称 模糊检索图书信息
书籍管理
书籍管理后台
About
with by GitHub
GRPC 远程调用图书馆——书籍管理后台
Electron（原名为Atom Shell）是GitHub开发的一个开源框架。它允许使用Node.js（作为后端）和Chromium（作为前端）完成桌面GUI应用程序的开发。Electron现已被多个开源Web应用程序用于前端与后端的开发，著名项目包括GitHub的Atom和微软的Visual Studio Code
查询所有书籍
删除书籍
ID 书名 作者 简介
1 平凡的世界 路遥
2 Google gRPC 编程实例 诚夏
添加书籍
添加一本新书


ELECTRON RPC LIBRARY
书籍查询
通过 书籍ID 查询图书信息
通过 书籍名称 模糊检索图书信息
书籍管理
书籍管理后台
About
with by GitHub
GRPC 远程调用图书馆——书籍 ID 检索
gRPC 是一个高性能、开源和通用的 RPC 框架，面向移动和 HTTP/2 设计。目前提供 C、Java 和 Go 语言版本，分别是：grpc, grpc-java, grpc-go. 其中 C 版本支持 C, C++, Node.js, Python, Ruby, Objective-C, PHP 和 C# 支持.
gRPC 基于 HTTP/2 标准设计，带来诸如双向流、流控、头部压缩、单 TCP 连接上的多复用请求等特。这些特性使得其在移动设备上表现更好，更省电和节省空间占用。
请输入查询 ID （整数）
2
查询
ID 书名 作者 简介
2 Google gRPC 编程实例 诚夏

界面的动态演示可以点击： http://media.sumblog.cn/img/20190424162851.gif-min_pic 查看

实现过程

本系统采用 gRPC 作为 RPC 中间件，客户端通过远程过程调用，实现与服务器端的通信

服务器端实现

服务器端采用 Java 作为开发语言，gPRC 作为远程过程调用中间件，使用 hibernate 将 Java 类映射到数据库表中，后端数据库采用 Mysql。采用 npm 包管理器，管理项目所依赖的组件。

gRPC 配置

- 编写 proto 文件

  创建 BookID、BookName、Book、BookList、UserID 这几种 message 类型

  创建 BookManager 类，完整的 proto 配置如下：

```
syntax = "proto3";

option java_multiple_files = true;
option java_package = "SincereXIA.distribrpc.book";
option java_outer_classname = "BookProto";

package library;

message BookID{
    int32 id = 1;
```

```
}

message BookName{
    string name = 1;
}

message Book{
    BookID bookID = 1;
    BookName bookName = 2;
    string bookAuthor = 3;
    string bookInfo = 4;
}

message BookList{
    repeated Book book = 1;
}

message UserID{
    string userName = 1;
    string userPwd = 2;
}

message Request{
    BookID bookID = 1;
    BookName bookName = 2;
    Book book = 3;
    UserID userID = 4;
}

message Reply{
    int32 status = 1;
    string msg = 2;
}

service BookManager{
    rpc add(Request) returns (Reply){};
    rpc queryByID(Request) returns (Book){};
    rpc queryByName(Request) returns (BookList){};
    rpc delete(Request) returns (Reply){};
    rpc login(UserID) returns (Reply){};
}
```

- 数据库配置

  在数据库中建立表 `book`

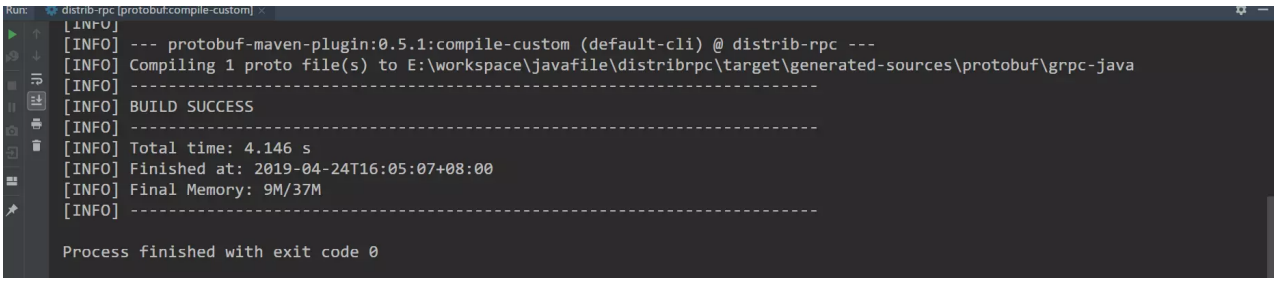
  为 Hibernate 编写数据库映射文件

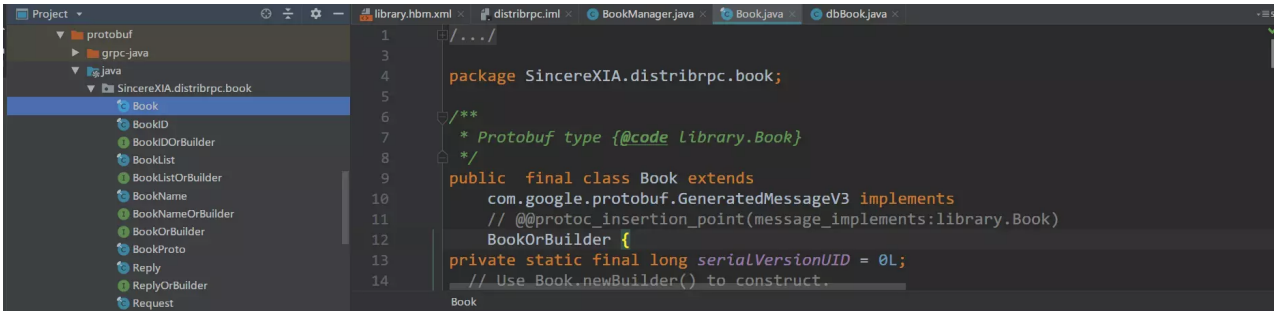
```xml
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE hibernate-mapping PUBLIC "-//Hibernate/Hibernate Mapping DTD 3.0//EN"
        "http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd">

<hibernate-mapping>
    <class name="GrpcProto.dbBook" table="books" catalog="library">
        <id name="id" type="java.lang.Integer">
            <column name="id" />
            <generator class="identity" />
        </id>
        <property name="bookName" type="string">
            <column name="bookname" length="200" not-null="true" unique="true" />
        </property>
        <property name="bookAuthor" type="string">
            <column name="bookauthor" length="200" not-null="true" unique="true"
/>
        </property>
        <property name="bookInfo" type="string">
            <column name="bookinfo" length="200" not-null="true" unique="true" />
        </property>
    </class>
</hibernate-mapping>
```

创建 dbBook 类，实现从 java 类到数据库记录之间的映射，至此，对数据库的操作就可以转为对 dbBook 对象的操作。

- 生成 `server stub`

  运行 `protobuf:compile-custom` 、`protobuf:compile` 命令，编译生成对应的 server stub 类





- 编写 BookManager 类，对定义的接口进行实现

  protobuf 编译之后，在生成的 `BookManagerGrpc.BookManagerImplBase` 类中，定义了需要实现的接口，编写 `BookManager` 类继承之，并完成方法的编写

- 编写 gRPC 服务器端程序

  编写 `BookManagerServer` 类，启动 gRPC 服务，进行端口监听

```
    private void start() throws IOException {
        /* The port on which the server should run */
        int port = 50051;
        server = ServerBuilder.forPort(port).addService(new
BookManager()).build().start();
        System.out.println("**** 服务成功启动");
        Runtime.getRuntime().addShutdownHook(new Thread() {
            @Override
            public void run() {
                // Use stderr here since the logger may have been reset by its
                // JVM shutdown hook.
                System.err.println("*** shutting down gRPC server since JVM is
shutting down");
                BookManagerServer.this.stop();
                System.err.println("*** server shut down");
            }
        });
    }
```

至此 服务器端开发完毕

客户端实现

客户端采用 Nodejs + election 实现前端界面的开发，gRPC 也提供了 Nodejs 的中间件，来完成客户端与服务器端的通信

- proto 文件引入

  客户端代码中引入和服务器端相同的 proto 文件，定义接口

- 编写调用函数，使用 Nodejs 回调函数的方法，实现客户端与服务器端的异步通信

```
var grpc = require('grpc');
var PROTO_PATH = __dirname + '\\protos\\books.proto';
var protoLoader = require('@grpc/proto-loader');
var packageDefinition = protoLoader.loadSync(
    PROTO_PATH,
    {keepCase: true,
        longs: String,
        enums: String,
        defaults: true,
        oneofs: true
    });
const library = grpc.loadPackageDefinition(packageDefinition).library;
var bookManager = new library.BookManager('disk.sumblog.cn:50051',
    grpc.credentials.createInsecure());

/**
 * 通过 ID 值来获取书籍
 * @param id 书的编号 一个整数
 * @param callback
 */
function queryByID(id, callback) {
    function _(err, book) {
```

```
        if (err) {
            console.log(err);
        } else {
            callback(book);
        }
    }
    return bookManager.queryByID({"bookID": {'id': id}},  _);
}

function queryByName(bookname, callback) {
    return bookManager.queryByName({"bookName":{'name':bookname}}, function (err, books) {
        if (err){
            console.log(err);
        } else {
            callback(books.book)
        }
    })
}

function add(request, callback) {
    return bookManager.add(request, function (err, msg) {
        if (err){
            console.log(err);
        }else {
            callback(msg);
        }
    })
}

function del(request, callback){
    return bookManager.delete(request, function (err, msg) {
        if (err){
            console.log(err);
        }else {
            callback(msg);
        }
    })
}

module.exports = {
    queryByID,
    queryByName,
    add,
    del
}

```

- 编写前端界面代码

至此，客户端开发工作完成