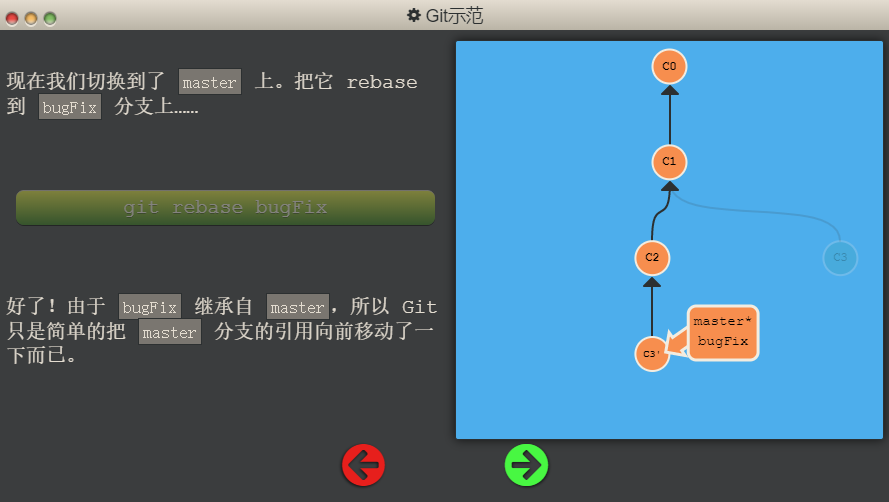
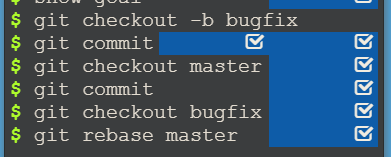
### git

git rebase



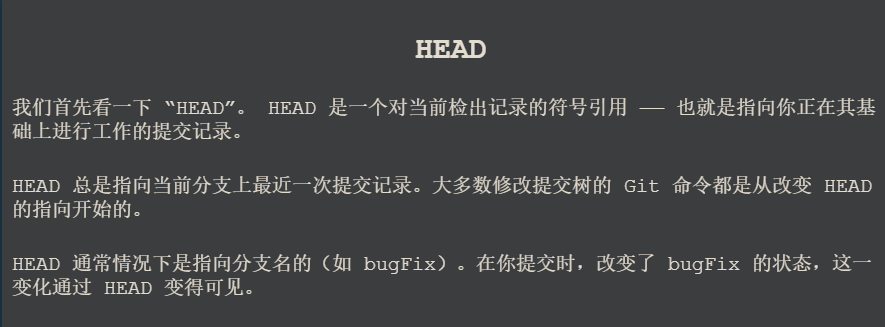








head



### User项目练习

#### 数据库设计



#### 接口设计

* + - 1. 登陆后的基本信息数据结构

struct BasicInfo

    {

        0  *optional*  long        userId;                // 用户id

        1  *optional*  string      token;                 // token

        2  *optional*  string      username;              // 用户名

    };

* + - 1. 数据响应的基本状态数据结构

  struct BasicRsp

    {

        0  *optional*  int                     iRet;                  // 返回码

        1  *optional*  string                  message;               // 返回信息

    };

* + - 1. 登录请求的数据结构

   struct LoginReq

    {

        0  *require*  string                  username;                // 用户名

        1  *require*  string                  password;                // 用户密码

    };

* + - 1. 登录响应的数据结构

    struct LoginRsp

    {

        0  *optional*  int                     iRet;                  // 返回码

        1  *optional*  string                  message;               // 返回信息

    };

* + - 1. 分页请求的数据结构

   struct PageInfoReq

    {

        0  *optional*  BasicInfo               basicInfo;             // 基本信息

        1  *optional*  long                    uid;                    // 查询ID

        2  *optional*  int                     page = 0;              // 当前页数 默认:0

        3  *optional*  int                     pageNum = 10;          // 每页显示条数 默认:10

    };

* + - 1. 根据id查询信息的数据结构

  struct QueryIdReq

    {

        0  *optional*  BasicInfo               basicInfo;             // 基本信息

        1  *require*  long                     uid;                   // 用户id

    };

* + - 1. 用户基本信息数据结构

    struct UserInfo

    {

        0  *optional*  long                    uid;                   // 用户id

        1  *optional*  string                  username;               // 用户姓名

        2  *optional*  string                  password;               // 用户密码

        3  *optional*  string                  gender;                // 用户性别

        4  *optional*  string                  tel;                   // 用户电话

        5  *optional*  int                   age;                   // 用户年龄

    };

* + - 1. 用户详细信息响应数据结构

    struct UserInfoRsp

    {

        0  *optional*  int                     iRet;                  // 返回码

        1  *optional*  string                  message;               // 返回信息

        2  *optional*  UserInfo                userinfo;               // 用户信息

    };

* + - 1. 用户列表数据结构

  struct UserInfoList

    {

        0  *optional*  int                     total;                 // 总条数

        1  *optional*  vector<UserInfo>        list;                  // 用户列表

    };

* + - 1. 用户列表响应数据结构

    struct UserListRsp

    {

        0  *optional*  int                     iRet;                  // 返回码

        1  *optional*  string                  message;               // 返回信息

        2  *optional*  UserInfoList            data;                  // 用户信息列表

    };

* + - 1. 保存用户请求数据结构

  struct SaveUserInfoReq

    {

        0   *optional*  BasicInfo              basicInfo;              // 基本信息

        1   *require*  UserInfo               userinfo;               // 用户信息

    };

* + - 1. 保持用户响应数据结构

 struct SaveUserInfoRsp

    {

        0  *optional*  int                     iRet;                  // 返回码

        1  *optional*  string                  message;               // 返回信息

        2  *optional*  int                     uid;                    // 返回id

    };

* + - 1. 接口方法

 // h5 用户登录

        int *login*(LoginReq stReq, out LoginRsp stRsp);

        // h5 获取用户列表

        int *getUserList*(PageInfoReq stReq, out UserListRsp stRsp);

        // h5 获取用户详细信息

        int *getUserDetail*(QueryIdReq stReq, out UserInfoRsp stRsp);

           // h5 用户登录

        int *loginForAd*(LoginReq stReq, out LoginRsp stRsp);

        // admin 获取用户列表

        int *getUserListForAd*(PageInfoReq stReq, out UserListRsp stRsp);

        // admin 删除用户

        int *delUser*(QueryIdReq stReq, out BasicRsp stRsp);

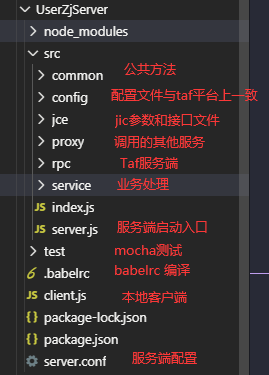
        // admin 添加/保存用户

        int *saveUser*(SaveUserInfoReq stReq, out SaveUserInfoRsp stRsp);

        // h5 获取用户详细信息

        int *getUserDetailForAd*(QueryIdReq stReq, out UserInfoRsp stRsp);

#### 项目结构搭建



#### 接口的实现（举一个例子，其他参照这个格式）

import { Test } from './User';

import UserService from '../service/User';

import { reflectRun } from '../common';

Test.UserImp.prototype.getUserDetailForAd = async function (*current*, *stReq*, *stRsp*) {

    await reflectRun(current, stReq, stRsp, UserService.getUserDetailForAd);

}

#### 处理接口里的业务逻辑

  getUserDetailForAd: async *req* => {

    const {uid} = req;

    if(!uid) throw error.INPUT\_ERROR();

    let getDetailSql = `select \* from user\_info where uid='${uid}';`;

*console*.log(getDetailSql);

    const [results, fields] = await conn.query(getDetailSql);

    const { username, password, gender, tel, age} = results[0];

          return {userinfo: {

            uid,

            username,

            password,

            gender,

            tel,

            age

          }};

  }

};

#### 数据库连接

// 引入

const mysql = require('mysql2');

// 创建数据库连接

// 使用异步

const conn  = mysql.createConnection({

    host: '192.168.6.170',

    user: 'root',

    password:'root',

    database: 'test\_zzj',

    charset:'utf8',

  });

export default conn.promise();

#### taf返回数据的处理

const OK\_RET = 0;

const OK\_RSP = { iRet: OK\_RET, message: 'ok' };

const returnRet = (*current*, *stRsp*, *ret*) => {

*console*.log(ret);

  stRsp.readFromObject(\_.assign({}, OK\_RSP, ret));

*console*.log(stRsp);

  current.sendResponse(OK\_RET, stRsp);

};

const returnErr = (*current*, *stRsp*, *err*) => {

*console*.log('xxxxxxxxxxxxxxxx');

  logger.exception.error(err);

  stRsp.readFromObject(err.err);

  current.sendResponse(OK\_RET, stRsp);

};

const reflectRun = async (*current*, *stReq*, *stRsp*, *fn*) => {

  try {

    const req = stReq.toObject();

    const ret = await fn(req);

    returnRet(current, stRsp, ret);

  } catch (err) {

    returnErr(current, stRsp, err);

  }

};

#### 配置taf服务

<taf>

    <application>

        <server>

            app=Test

            server=UserZjServer

            <Test.UserZjServer.UserZjObjAdapter>

                allow

                endpoint=tcp -h 127.0.0.1 -p 14001 -t 60000

                protocol=taf

                servant=Test.UserZjServer.UserZjObj

            </Test.UserZjServer.UserZjObjAdapter>

        </server>

        <client>

            modulename=Test.UserZjServer

        </client>

    </application>

</taf>

#### taf服务启动文件

const svr = **new** *Taf*.*server*();

const startSer = () => {

  svr.initialize(process.env.TAF\_CONFIG || './server.conf', function(*server*) {

*console*.log('--------------------------------------------------------');

    const servantName = `${server.Application}.${server.ServerName}.UserZjObj`;

    server.addServant(Test.UserImp, servantName);

    logger.data.info(`start servant: ${servantName}, ${process.pid}`);

  });

  svr.start();

};

const startup = async () => {

  try {

    await startSer();

  } catch (error) {

    process.exit(-1);

  }

};

process.on('uncaughtException', *err* => {

  logger.exception.error('uncaughtException', err);

});

process.on('unhandledRejection', (*reason*, *p*) => {

  logger.exception.error('unhandledRejection', reason, p);

});

startup();

#### 编写客户端

let servant = 'Test.UserZjServer.UserZjObj';

if (!process.env.TAF\_CONFIG) {

    servant += '@tcp -h 127.0.0.1 -p 14001 -t 60000';

}

const prx = TAF.stringToProxy(Test.UserProxy, servant);

const stReq = **new** *Test*.*QueryIdReq*();

stReq.readFromObject(

    {

        uid: 1,

    }

);

prx.getUserDetailForAd(stReq).then(function (*ret*) {

*console*.log('### helloWorld ok ###', ret.response.arguments.stRsp.toObject());

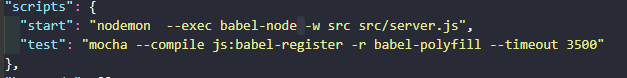
}, function (*ret*) {

*console*.log('### helloWorld error ###', ret.response);

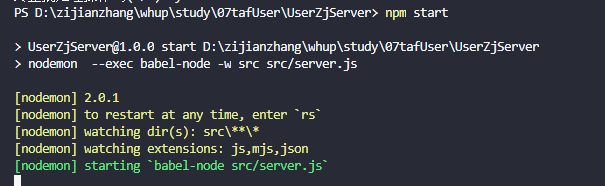
});

#### 启动服务端和客户端获取数据

1. 配置 package.json



1. 启动服务端



1. 启动客户端



