4 数据库设计

整个项目用到的数据格式比较统一，使用关系型数据库存储数据是一个不错的选择。根据需求分析，本项目使用四张表来存储所需要用到的数据：信息配置表（con），用户信息表（usr），跑步记录表（run），详细路线表（loc）。

4.1 信息配置表

信息配置表的表结构比较简单，因为它使用键值对的形式存储配置，其表结构如表 4.1 所示。

表 4.1 信息配置表结构

字段名	类型	是否为主键	可空	默认值	解释
k	text	是	否		配置项的键
v	text	否	否		配置项的值

在该表中键是固定的，而键对应的值需要根据实际场景进行配置，下面列出全部所需的键值对的配置规范：

miniv 系统最低支持的 iOS 软件版本号

minav 系统最低支持的安卓软件版本号

downi 最新版 iOS 软件下载地址，如：`http://example.com/d/i`

downa 最新版安卓软件下载地址，如：`http://example.com/d/a`

certfile HTTPS 证书存放位置，如：`/home/certfile.pem`

admin 管理员密码，如：`pa55w0rd`

continue 单次跑步允许中断的次数

count 每日记录有效成绩的跑步次数

mina GPS 定位精度下限（米）

maxa GPS 定位精度上限（米）

mins 瞬时速度下限（米/秒

maxs 瞬时速度上限（米/秒

mint 跑步时间下限（秒）

maxt 跑步时间上限（秒）

goal 本学期达标跑步次数

mdist 男生单次跑步路程要求（米）

wdist 女生单次跑步路程要求（米）

stime 跑步开始日期时间 UNIX 毫秒时间戳，例如 1550419200000 代表开始跑步的时间为 2019 年 2 月 18 日零点

etime 跑步结束日期时间 UNIX 毫秒时间戳，例如 1559491200000 代表结束跑步的时间为 2019 年 6 月 3 日零点

period 允许跑步的时间段，需要填入一个正整数，要解释这个正整数的含义，首先要将这个正整数转换成一个 48 位的二进制数，这 48 位从低位到高位依次代表了一天从零点开始到次日零点等分的四十八个时间段，如果这一位为 1 则代表该时间段可以跑步，为 0 则代表不能跑步。如 $281474976710655 = 2^{48} - 1$ 代表全天都可以跑步，而 0 则代表全天都不能跑步

region 允许的跑步场地，需要填入一个 JSON 数组，数组里的每一个对象代表的是一个场地，一个场地有名字（name）和形状（shape）两个属性，其中 shape 是一个坐标点数组，数组中坐标使用点对的形式表示。如：`[{"name":"三角","shape":[[0,0],[0,1],[1,0]]}]`

4.2 用户信息表

用户信息表用于存储学生基本信息，其表结构如表 4.2 所示。由于需要根据学号查询学生信息，所以将学号（sid）字段设置为主键即可。

表 4.2　用户信息表结构

字段名	类型	是否为主键	可空	默认值	解释
sid	text	是	否		学号
name	text	否	否		姓名
birth	text	否	否		八位数生日（用作密码）
sex	boolean	否	否		性别，0 代表女，1 代表男
cnt	integer	否	否	0	额外导入的成绩（次数）

4.3　跑步记录表

跑步记录表用于存储学生每次跑步的具体信息，其表结构如表 4.3 所示。由于有查询特定学生的跑步记录的需求，所以除了设置跑步 ID 为主键之外，还需要额外为学号（sid）字段建立索引。

表 4.3 中的状态（status）字段为一个整数，不同的整数代表着不同的含义：-1 表示的是，服务器已经接收到开始跑步的请求并为其生成跑步 ID，但跑步并没有结束；0 表示的是，跑步已经成功结束，并且是在规定时间内完成的，算作有效成绩；11 表示的是，跑步路程在规定时间内未达标，算作无效成绩；12 表示的是，跑完规定路程用时过短（小于规定时间的下限），有作弊嫌疑，算作无效成绩；13 表示的是，跑步已经成功结束，但超过了当日记录有效成绩的次数上限，算作无效成绩。

4.4　详细路线表

详细路线表记录了学生每次跑步的路线信息，他是以坐标点和时间戳的形式来存储数据的。跑步路线仅用于作为管理员判定某次跑步为作弊的辅助参考信息，普通用户无法直接获取。其表结构如表 4.4 所示。

表 4.3　跑步记录表结构

字段名	类型	是否为主键	可空	默认值	解释
rid	text	是	否		跑步 ID
sid	text	否	否		学号
date	text	否	否		yyyy-MM-dd 形式的跑步日期
time	bigint	否	否		跑步开始的 UNIX 毫秒时间戳
cost	integer	否	否	0	跑步用时（秒）
dist	integer	否	否	0	跑步路程（米）
step	integer	否	否	0	步数
status	integer	否	否	-1	跑步状态

表 4.4　详细路线表结构

字段名	类型	是否为主键	可空	默认值	解释
rid	text	否	否		跑步 ID
lng	numeric	否	否		坐标点经度
lat	numeric	否	否		坐标点纬度
rec_time	bigint	否	否		GPS 定位时间戳
ins_time	bigint	否	否		入库时间戳

5　网络接口设计

网络接口是客户端与服务端通讯的方式，本项目网络接口选择使用 HTTPS 作为通讯协议，使用 JSON 作为数据传输格式。为了满足需求分析中的需求，应当有如下六个接口：login（用户登录，同时获取用户信息）、info（获取跑步规则等信息）、start（开始跑步，请求生成跑步 ID）、end（结束跑步，上传成绩）、report（上报 GPS 定位点）、record（获取跑步记录）。

在详细说明各个接口之前，需要首先解决一个问题，那就是如何避免中间人攻击。在上文中已经提到，攻击者可以通过伪造证书的形式来实施中间人攻击，那么需要解决的问题其实是如何判断证书是否是被伪造的。本项目的解决方法是进行证书的双向认证：

1. 客户端先生成两个随机字符串 cs 和 cp，并将它们放入登录请求的头部。
2. 服务端接收到登录请求后，将证书指纹与 cs, cp 连接组成一个新字符串并计算其哈希值 vv，同时服务端根据学号哈希生成两个字符串 vp 和 vs，在处理完正常的登录请求之后，在返回的数据中带上 vp, vs, vv 三个字段。
3. 客户端拿到登录的返回结果同时也拿到了 vp, vs, vv 三个值，此时客户端将自己拿到的证书的指纹与 cs 和 cp 进行上一条与服务端同样的计算，并将计算结果与 vv 进行比对则可验证自己是否拿到了真正的证书，到这里客户端验证已经完成。同时客户端需要将自己拿到的证书的指纹与 vp, vs 连接组成一个新字符串并计算其哈希值 v，在后续所有的非登录请求中，v 需要放入请求头部以便服务端进行验证。
4. 服务端处理所有的非登录请求之前，都会先检查客户端计算的 v 是否与自己计算的相同，如果不同则很明显证书已经被攻击者伪造，如果相同则证明客户端拿到了自己的证书。至此，服务端的验证也已经完成。
5. 如果上述环节出现认证失败，服务端通过返回错误码来告知客户端，而客户端则通过界面提示来告知用户。

5.1 用户登录

用户登录接口主要用于验证用户的账号密码，同时也是进行证书双向验证过程中关键的接口。该接口的请求方式为 POST，请求相对地址为 `/api/login`。

5.1.1 请求参数格式

sid 必填字符串，代表学号

pwd 必填字符串，代表密码

5.1.2 返回数据格式

err 错误码，0 代表登陆成功，100 代表客户端版本因过低需要升级，101 代表用户不存在，102 代表密码错误，103 代表请求数据不完整

token 成功登陆之后，用于后续请求验证用户身份的 token key 字符串

vp 服务端根据学号生成的一个字符串，用于证书双向验证

vs 服务端根据学号生成的另一个字符串，用于证书双向验证

vv 服务端根据客户端生成的随机字符串计算出的哈希字符串，用于证书双向验证

```
{ // 用户登录接口返回数据示例
        "err": 0,
        "token": "user_valid_token",
        "vp": "server_prefix_string",
        "vs": "server_suffix_string",
        "vv": "validation_string",
}
```

5.2 获取信息

获取信息接口主要用于客户端获取跑步规则、场地限制等信息，是跑步开始之前必须请求的接口。该接口的请求方式为 POST，请求相对地址为 `/api/info`。

5.2.1 请求参数格式

token 必填字符串，验证用户身份的 token key

5.2.2 返回数据格式

err 错误码，0 代表成功，201 代表无法验证用户身份

name 学生姓名

dist 该学生单次需要跑的路程

其他数据 很多数据的含义在讲配置信息表的时候已经有解释，于是不再赘述，这些数据包括：continue, count, mina, maxa, mins, maxs, mint, maxt, goal, stime, etime, period, region

```
{ // 获取信息接口返回数据示例
        "err": 0, "name": "张三", "goal": 30, "count": 1,
        "dist": 2000, "continue": 1, "mina": 0, "mins": 0,
        "mint": 200, "maxa": 20, "maxs": 10, "maxt": 1200,
        "stime": 1550419200000, "etime": 1559491200000,
        "period": 281474976710655, "region": [ {
                "name": "三角", "shape": [
                        [0, 0], [0, 1], [1, 0]
        ] } ]
}
```

5.3 开始跑步

开始跑步接口用于告知服务器某个学生开始跑步，同时服务器生成一个跑步 ID 并返回。该接口的请求方式为 POST，请求相对地址为 `/api/start`。

5.3.1 请求参数格式

token 必填字符串，验证用户身份的 token key

5.3.2 返回数据格式

err 错误码，0 代表成功，300 代表不在规定的跑步时间，301 代表无法验证用户身份，302 代表后台系统出错

rid 成功生成跑步 ID 且跑步初始信息已经入库后，该数据表示生成的跑步 ID

5.4 结束跑步

结束跑步接口用于服务端接收客户端上传的跑步结果，判定跑步结果并返回。该接口的请求方式为 POST，请求相对地址为 `/api/end`。

5.4.1 请求参数格式

rid 开始跑步请求生成的跑步 ID

token 必填字符串，验证用户身份的 token key

dist 跑步路程

cost 跑步用时

step 步数统计

5.4.2 返回数据格式

err 错误码，0 代表结果成功上传，401 代表无法验证用户身份，402 代表数据无效（比如跑步 ID 不存在，数据不完整等情况），-1 代表更新数据库时出现错

误。如果错误码为 0, 11, 12, 13 则其含义与跑步信息表中 status 字段含义相同，不再赘述

5.5 位置上报

位置上报接口用于客户端上传 GPS 定位获取的坐标数据集，没有返回数据。该接口的请求方式为 POST，请求相对地址为 `/api/report`。

5.5.1 请求参数格式

rid 开始跑步请求生成的跑步 ID

token 必填字符串，验证用户身份的 token key

route JSON 格式的坐标数据集，为一个数组，数组中每一个对象有 x, y, t 三个属性代表经纬度和记录时间，如：`[{ "x": 1, "y": 2, "t": 1000 }]`

5.6 获取记录

获取记录接口用于客户端向服务端请求该学生的跑步记录。该接口的请求方式为 POST，请求相对地址为 `/api/record`。

5.6.1 请求参数格式

token 必填字符串，验证用户身份的 token key

5.6.2 返回数据格式

err 错误码，0 代表成功，501 代表无法验证用户身份

data 该学生的跑步记录数组，数组中每个对象拥有如下属性：rid, time, step, cost, dist, sid, status。其意义与跑步信息表字段意义相同，不再赘述

6 服务端设计

在本项目中，服务端是连接客户端与数据库的桥梁，所有客户端需要获取的数据库数据都需要通过服务端代理访问，所有客户端需要对数据库进行的数据变更操作也需要经由服务端代为变更，服务端相较于数据库有更灵活的数据处理能力以及鉴权能力。

使用 Python, Ruby, JavaScript 等脚本语言借助服务框架编写 RESTful (Representational State Transfer) 网络服务已经成为了很多网络应用的开发方式[8]。事实上，大多数网络应用开发者都会选择使用 HTTP 作为数据传输协议，使用 XML 或 JSON 作为数据传输格式。

本项目选择使用 JavaScript 作为服务端编程语言，借助高效的 Node.js 微服务框架 Koa 进行编写。Koa 是一个全新的网络应用开发框架，由前 Express 团队的成员开发，旨在于成为一个更轻便、高效、强大的服务框架。Koa 的特性在于使用 JavaScript 语言标准 ECMAScript 2017 中的异步函数（async functions），使得开发者可以完全抛弃回调函数，并用更优雅的方式处理异常。Koa 本质上是一个中间件（middleware）执行器，并且其本身不捆绑任何中间件，这使得其体积小巧，同时开发者的可定制程度也更高。

6.1 代码结构

服务端代码的结构（仅罗列了重要的代码文件，部分文件并未列出）如下：

- index.js 服务入口文件，负责启动 HTTP 服务并与数据库建立连接
- lib/ 文件夹内为一些通用的函数库文件
  - crypt.js 加密/解密相关功能函数库
  - log.js 日志打印功能函数库
  - time.js 日期时间相关功能函数库
- database/ 文件夹内代码负责数据库相关功能

  - index.js 数据库连接，暴露对外接口
  - con.js 控制与配置信息表的数据交互
  - usr.js 控制与用户信息表的数据交互
  - run.js 控制与跑步信息表的数据交互
  - loc.js 控制与路线信息表的数据交互
- auser/ 文件夹内代码负责网络请求逻辑处理
  - index.js 负责网络服务中间件的绑定
  - login.js 用户登录接口实现
  - info.js 获取信息接口实现
  - start.js 开始跑步接口实现
  - end.js 结束跑步接口实现
  - report.js 位置上报接口实现
  - record.js 获取记录接口实现

6.2 逻辑实现

很多接口以及功能的实现比较简单，不需要进行多余的解释说明，下面将介绍稍复杂一些的功能的具体实现。

6.2.1 合法跑步时间判定

服务端在处理开始跑步的请求时，需要先进行一个合法跑步时间的判断：如果当前时间不可以跑步，则拒绝生成跑步 ID 并返回对应错误码。

根据配置信息表很容易得出，一个时间是否为合法的跑步时间是由三个配置项决定的：stime, etime, period。stime 和 etime 的问题很好解决，直接获取当前的时间戳并与它们比对大小即可，任何小于 stime 或者大于 etime 的时间戳都是非法的。问题的重点在于如何根据 period 来确定当前时间是否是合法的跑步时间段。

为了解决上述问题，应当先确定当前时间在一天 48 个等分时间段中的哪一段。我们设 h 为当前时间的小时数（24 小时制），则如果当前分钟数小于 30 时，段序

号 t 计算公式为：

$$t = 2 \cdot h$$

如果当前分钟数大于等于 30 时，段序号 t 计算公式为：

$$t = 2 \cdot h + 1$$

在拿到段序号 t 之后，我们只需要判断 2^t 与 period 按位和运算的结果是否非零，即可判断当前时间段是否合法。

整个过程伪代码如下：

Algorithm 1 判断是否为合法的跑步时间

```
1:  function IsValidTime(stime, etime, period)
2:      ts ← GetTimestamp                          ▷ 获取当前时间戳
3:      if ts < stime or etime < ts then
4:          return false                           ▷ 非法跑步日期
5:      end if
6:      t ← 2· GetHours                            ▷ GetHours 为获取当前小时数
7:      if GetMinutes ≥ 30 then                    ▷ GetMinutes 为获取当前分钟数
8:          t ← t + 1
9:      end if
10:     if (2^t & period) = 0 then                 ▷ & 为位运算与
11:         return false                           ▷ 非法跑步时间段
12:     else
13:         return true                            ▷ 合法跑步时间
14:     end if
15: end function
```

6.2.2 成绩判定

服务端获取到客户端上传的成绩之后，需要对该次跑步的成绩有效性进行判定。在这里有两个细节性的地方需要注意:

1. 开始跑步请求与结束跑步请求之间的的时间间隔十分重要，举个例子：服务器在 19:00 接收到一个开始跑步（start）请求，并为其分配了一个跑步 ID，之后服务器在 19:05 接收到该跑步 ID 的结束跑步（end）请求，并且其耗时（cost）参数为 1000。这里就出现了一个矛盾的事情：19:00 到 19:05 一共 300 秒的时间，客户端却告知服务端用时 1000 秒。这种情况基本可以判定用户存在作弊行为，不可为其算作有效成绩。
2. 根据配置信息表中的 count 字段，单日计算有效成绩的次数是有限制的，比如用户今日已经有一次有效成绩，并且配置信息中 count 为 1，那么即使该同学当日又在限制条件内完成了跑步，也不能再算成绩。这个时候应该预先统计该用户当日已有的有效成绩次数再决定该次成绩是否有效。

对于上面的第一个问题，只需要在最开始做一个判定，用当前时间减去跑步开始的时间（跑步信息表中的 time 字段），然后与客户端发送的 cost 参数进行比较，只要这个时间间隔大于或等于 cost 即可进行后续判断（因为网络传输是存在延时的）。

对于上面的第二个问题，需要借助的是跑步信息表中的 date 字段，虽然不同的 rid 可能对应了了不同的 time，但是只要是在同一天生成的跑步 ID，其 date 字段必然是相同的。所以可以用如下 SQL 语句去获取一个学生当日已有的有效成绩:

```
SELECT
        COUNT(*)
FROM
        run
WHERE
        sid = %该学生学号% AND status = 0 AND
        date = %当天的 yyyy-MM-dd 格式日期%
```

做完上述两个步骤之后，就可以用下面伪代码的逻辑去计算 status 字段：

Algorithm 2 计算 status 字段

```
function CalculateStatus(cost, dist, mint, maxt)
    rdist ← GetDistRequirement    ▷ 获取该学生规定路程（不同性别要求不同）
    cnt ← GetValidCount           ▷ GetValidCount 为获取该学生当日有效成绩次数
    if dist < rdist then
        return 11                                          ▷ 路程未达标
    else if cost < mint or maxt < cost then
        return 12                                          ▷ 时间超限制
    else if cnt ≥ count then
        return 13                                  ▷ 超出当日有效次数上限
    else
        return 0                                           ▷ 有效成绩
    end if
end function
```