接口规范

GET 方法（查）

1. 成功的 GET 方法通常返回 HTTP 状态代码 200（正常）。
2. 如果**找不到资源**，该方法应返回 404（未找到）。
3. 如果请求已完成，但 **HTTP 响应中没有包含响应正文**，则应返回 HTTP 状态代码 204（无内容）；例如，不产生匹配项的搜索操作可能会通过此行为实现。(啥也没请求着)

POST 方法（改）

如果使用 POST 方法创建了新资源，则会返回 HTTP 状态代码 201（已创建）。新资源的 URI 包含在响应的 Location 标头中。响应正文包含资源的表示形式。
如果该方法执行了一些处理但**未创建新资源**，则可以返回 HTTP 状态代码 200(还是相当于查询)，并在响应正文中包含操作结果，例如资源已经存在的情况下再次创建，应该返回内容告知具体的信息。
或者，如果没有可返回的结果，该方法可以返回 HTTP 状态代码 204（无内容）但不返回任何响应正文。
如果客户端将无效数据放入请求，服务器应返回 HTTP 状态代码 400（错误的请求）。 响应正文可以包含有关错误的其他信息，或包含可提供更多详细信息的 URI 链接。

PUT 方法（增）

与 POST 方法一样，如果 PUT 方法创建了新资源，则会返回 HTTP 状态代码 201（已创建）。如果该方法更新了现有资源，则会返回 200（正常）或 204（无内容）。
在某些情况下，可能无法更新现有资源。在这种情况下，可考虑返回 HTTP 状态代码 409（冲突）。
PUT 请求应指定集合的 URI，而请求正文则应指定要修改的资源的详细信息。此方法可帮助减少交互成本并提高性能。

PATCH 方法

客户端可以使用 PATCH 请求向现有资源发送一组修补文档形式的更新。服务器将处理该修补文档以执行更新。修补文档不会描述整个资源，而只描述一组要应用的更改。PATCH 方法的规范 (RFC 5789) 未定义修补文档的特定格式。 必须从请求中的媒体类型推断格式。
有两种基于 JSON 的主要修补格式，分别称作“JSON 修补”和“JSON 合并修补”。JSON 合并修补更简单一些。 修补文档的结构与原始 JSON 资源相同，但只包含要更改或添加的字段的子集。 此外，可以通过在修补文档中为字段值指定 null，

来删除该字段。 （这意味着，如果原始资源包含显式 null 值，则不适合使用合并修补。）

DELETE 方法（删）

如果删除操作成功，Web 服务器应以 HTTP 状态代码 204（无内容）做出响应，指示已成功处理该过程，但响应正文不包含其他信息。如果资源不存在，Web 服务器可以返回 HTTP 404（未找到）。

URI 路径约定

根据上述规范，API 对应的 URI 约定如下：
{schema}://{ip}:{port}/api/{term}/{app}/{version}/{api-name}

说明：

- schema 为 http 或者 https，建议使用 https
- ip 为接口 ip 地址
- port 为接口 listen 的端口号
- 作为接口路径，为了和其他路径区分，需要在路径中添加 api 目录
- term 为了区分部门、团队或者业务方向，比如大数据：bigdata，云：cloud 等，这个可选，如果能明确区分全局服务，可以不加
- app 为应用或服务名称 如车辆: vehicle，人员库：pl，视觉目标: vot，监控：monitor，权限：manage 等
- version 为服务的大版本号，例如：v3，当然如果能明确 API 的变动并不频繁，可以不引入版本管理。
- api-name 为接口名称，需要按照 RESTful 资源路径方式编写，比如查询所有用户：GET /users，添加点位：POST /locations
- 集合和资源需要使用多个单词描述的情况下，采用 kebab-case 命名法，要按照'-'字符进行分割，比如人脸聚类：face-groups 或者拆开使用 face/groups 更合适。
- 注意要按照资源来命名 URI 而不是按照功能来命名，例如以图搜车，面向的资源仍然是车的集合，所以应该使用/vehicles 而不是使用/image-search-vehicle
- 如果一个资源存在过多的分析或者功能，确实存在无法和 RESTful 理念完全一致的设计，那么不要一味追求符合 REST 风格而忽略可维护性，应该进行合适的抽象和分组，例如人脸同行分析：/face/groups/peer，频繁过车：/vehicles/frequent，多点碰撞：/vehicles/multi-location-collision 等，但是像按车牌搜车、按车型搜车、轨迹重现等功能仍然需要采用同样的 API /vehicles 实现比较合理。

```
1. http://192.168.1.1/api/ad/v3/accounts
2. http://192.168.1.2:8080/api/cloud/storage/v2/nfs
3. http://192.168.1.3/api/monitor/v3/managed-kafka/state
4. http://192.168.1.4/api/monitor/v3/managed-kafka/topic/{topic-name}?query=offsets
5. http://192.168.1.5/api/monitor/v3/vehicles
```

响应状态码

响应状态码通常分为下面几类：

1xx 范围的状态码是保留给底层 HTTP 功能使用的，并且估计在你的职业生涯里面也用不着手动发送这样一个状态码出来。

2xx 范围的状态码是保留给成功消息使用的，你尽可能的确保服务器总发送这些状态码给用户。

3xx 范围的状态码是保留给重定向用的。大多数的 API 不会太常使用这类状态码，但是在新的超媒体样式的 API 中会使用更多一些。

4xx 范围的状态码是保留给**客户端错误**用的。例如，客户端提供了一些错误的数据或请求了不存在的内容。这些请求应该是幂等的，不会改变任何服务器的状态。

5xx 范围的状态码是保留给**服务器端错误**用的。这些错误常常是由于代码本身异常未做处理或者服务底层的函数抛出异常。发送这类状态码的目的是确保客户端能得到一些响应，同时需要开发人员进行故障的排除。收到 5xx 响应后，客户端没办法知道服务器端的状态，所以这类状态码要尽可能的避免。

如何指定状态码？

中间件如何处理？

中间件功能：

1. 负载均衡：将请求分配给多个服务器，使得每个服务器都能够充分利用资源，提高系统的并发处理能力和吞吐量。
2. 缓存：将频繁访问的数据缓存到内存中，减少对数据库的访问，提高系统的响应速度。（**设置缓存就是在中间件中设置的嘛？**）
3. 安全控制：对请求进行安全过滤和访问控制，保护系统的安全性。
4. 日志记录：记录请求和响应的信息，便于系统的监控和排错。
5. API 管理：对外提供 API 接口的管理和发布，便于开发者使用和集成。
6. 服务发现：提供服务发现和注册功能，方便服务的管理和调用。
7. 数据转换：对请求和响应的数据进行转换和处理，使得系统能够更加灵活和可扩展。
8. 静态资源处理：对静态资源的访问进行处理和缓存，提高系统的访问速度。 9. 集成其他服务：与其他服务进行集成和协作，实现系统的功能扩展和整合。

在 GoFrame 框架中，通过鉴权中间件返回状态码通常涉及两个步骤：

首先，在鉴权中间件中进行鉴权检查，然后根据鉴权结果设置 HTTP 响应状态码。

创建一个鉴权中间件函数，用于进行鉴权检查。这个函数应该在需要鉴权的路由中使用。例如：

```
1. func AuthMiddleware(r *ghttp.Request) {
   // 进行鉴权检查    if !CheckAuth(r) {
       // 如果鉴权失败，设置响应状态码为 403 Forbidden
       r.Response.WriteStatus(http.StatusForbidden)
       return
   }
   r.Middleware.Next()
}
```

在上述示例中，AuthMiddleware 函数会检查鉴权条件（在 CheckAuth 函数中实现），如果鉴权失败，则设置响应状态码为 403 Forbidden。

注册鉴权中间件函数。

在项目初始化中，使用 ghttp.Middleware 函数注册上述定义的中间件(得提前声明一下，我要使用鉴权中间件函数了)：

```
1. func main() {
   s := g.Server()
   // 注册中间件函数    s.Use(AuthMiddleware)
   // 定义需要鉴权的路由规则    s.BindHandler("/secure", YourSecureHandler)
   s.Run()
}
```

通过上述步骤创建了一个简单的鉴权中间件，它在需要鉴权的路由中进行鉴权检查，并根据鉴权结果设置 HTTP 响应状态码。

```
1. s.Group("/api", func(group *ghttp.RouterGroup) {
   // 该中间件应用于全局接口
      group.Middleware(
          service.Middle().MiddlewareCORS,  // CORS 权限
          service.Middle().ResponseHandler, // 返回处理
       )
```

总结：

1. 创建一个鉴权中间件函数（函数内部编写需要鉴权的具体逻辑）
2. 在项目初始化中注册鉴权中间件函数（分组不同，中间件也可以不同）
3. 如何调用呢？

练习目的：

- 熟悉 GoFrame 的框架设计、开发工具、组件功能、编码规则和数据库 ORM 操作
- 熟悉 Web 服务开发的请求、返回数据规范
- 学习 Web 服务的接口规范和路由相关规则
- 熟悉当前产品项目的代码编写逻辑

登录端：

基于 JWT 的登录

登录：根据用户名决定登入的请求页，将用户信息存入 Redis，将 Token 解析之后，通过 Redis 进行查询，判断是否存在该用户。
JWT 由三部分组成：头部（Header）、载荷（Payload）、签名（Signature）

- 头部通常由两部分组成：令牌的类型（即 JWT）和正在使用的签名算法（如 HS256 RSA 等） {"alg":"HS256","typ":"JWT"} 经过 base64 编码后得到 eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9

- 载荷 Payload 由预定义 exp（过期时间），iat（签发时间）和私有载荷组成

预定义（Registered）：

公有（public）：

在使用 JWT 时可以额外定义的载荷。为了避免冲突，应该使用 IANA JSON Web Token Registry 中定义好的，或者给额外载荷加上类似命名空间的唯一标识。

私有（private）：

在信息交互的双方之间约定好的，既不是预定义载荷也不是公有载荷的一类载荷。这一类载荷可能会发生冲突，所以应该谨慎使用
而在 demo 中，私有载荷为{"id": 1, "username": "admin"}

Gf-JWT 使用流程

```
//封装中间件
```

// GfJWTMiddleware 提供了 Json Web 令牌身份验证实现。失败时，401 HTTP 响应返回。成功后，将调用封装好的中间件，并将 userID 设置为 c.Get（“userID”）。（字符串）。用户可以通过向 LoginHandler 发布 json 请求来获取令牌。然后需要传入令牌 Authentication 标头。示例：授权：承载 XXX_TOKEN_XXX

```
1. auth := jwt.New(&jwt.GfJWTMiddleware{
   Realm:           "test zone",
   Key:             []byte("secret key"),
   Timeout:         time.Minute * 5,
   MaxRefresh:      time.Minute * 5,
   IdentityKey:     "UserID",
   TokenLookup:     "header: Authorization, query: token, cookie: jwt",
   TokenHeadName:   "Bearer",
   TimeFunc:        time.Now,
   Authenticator:   Authenticator,
   Unauthorized:    Unauthorized,
   PayloadFunc:     PayloadFunc,
   IdentityHandler: IdentityHandler,
})
```

用户端：

管理员端：

功能设计

一、图书管理

1. 图书查询接口：

可以通过图书名称，图书号、作者，出版社等查询图书信息表中的图书信息。

2. 图书修改接口：

可以修改图书号 、图书名称，作者名称，出版社等信息。

3. 图书删除接口：

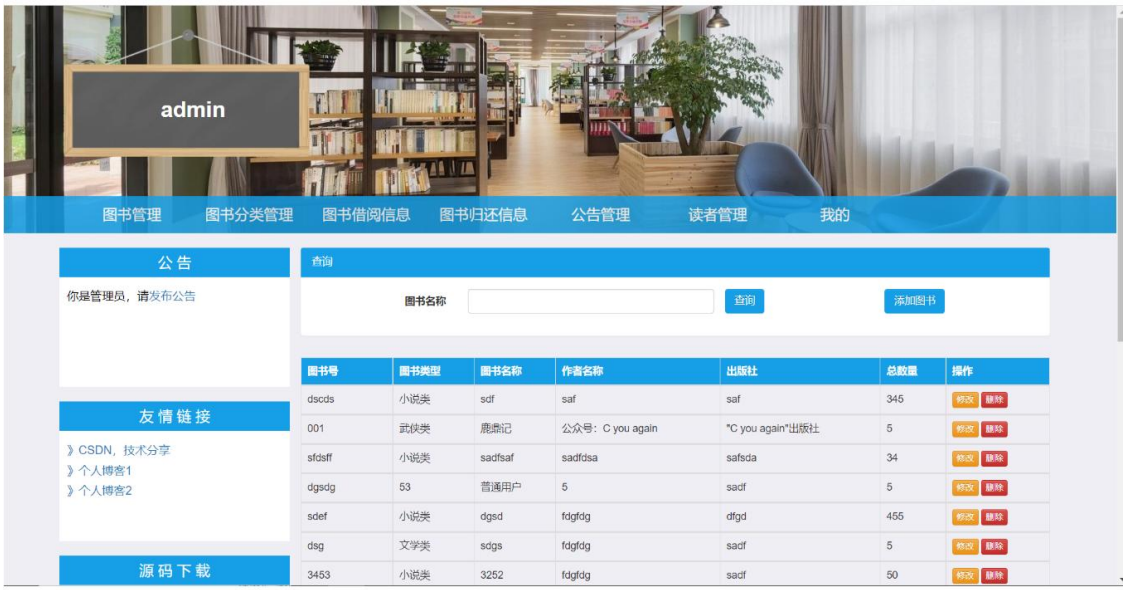
可以删除图书信息。

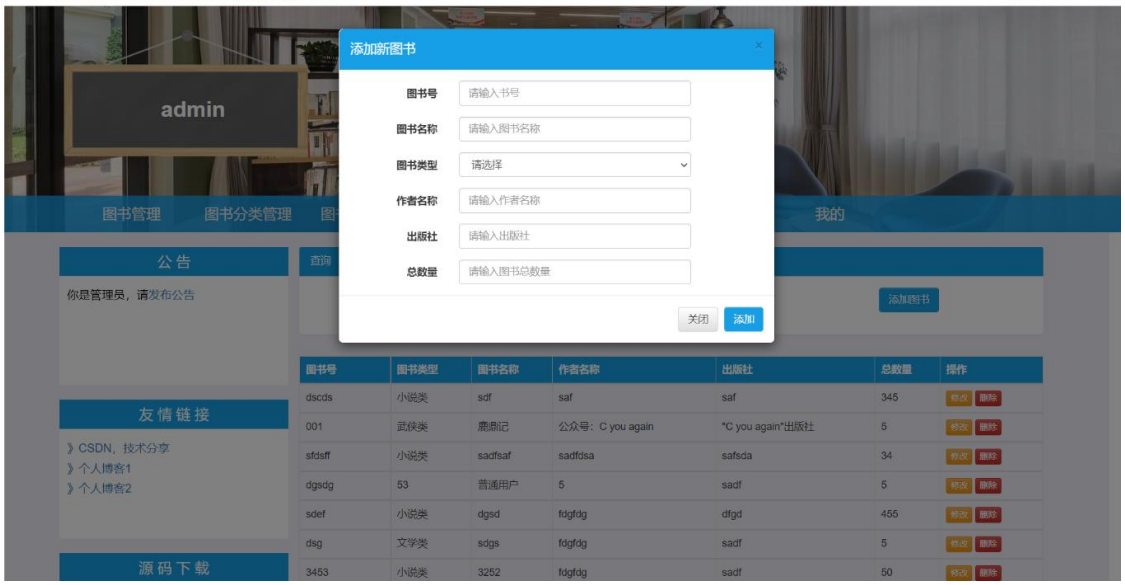
4. 图书添加接口：

执行图书添加后需要根据类型名称自动关联图书类别，即图书类型表上的该类型

图书数量 +1。

可以添加完整图书信息，系统保存至“图书信息表”中。





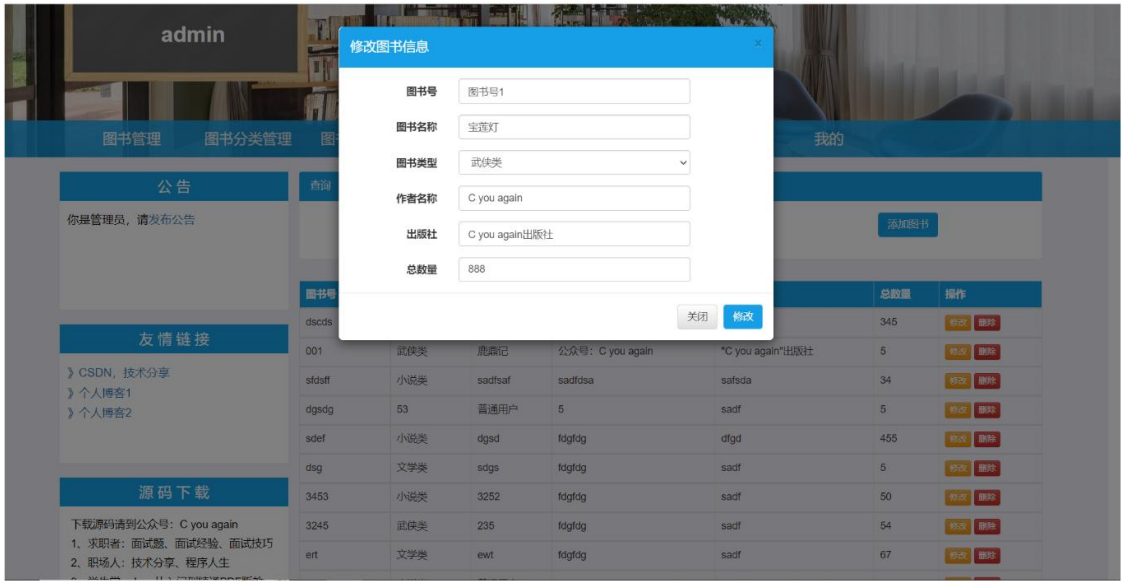


表 1 图书信息表（BookInformation）

字段	数据类型	主键否	允许为空否	默认值
ID	int(11)	是	否	
BookName	varchar(64)	否	否	
ISBN	varchar(64)	否	否	
Author	varchar(64)	否	是	NULL
Publishers	varchar(64)	否	是	NULL
BookTypeID	int(11)	否	否	
Amount	int(11)	否	否	

二、图书分类管理

1. 图书类别查询：

点击类型编号按钮，返回所有的图书类型

然后点击图书类型的查看按钮，则返回“图书信息表”中该类型囊括的图书。

（1） 图书类别名称查询接口

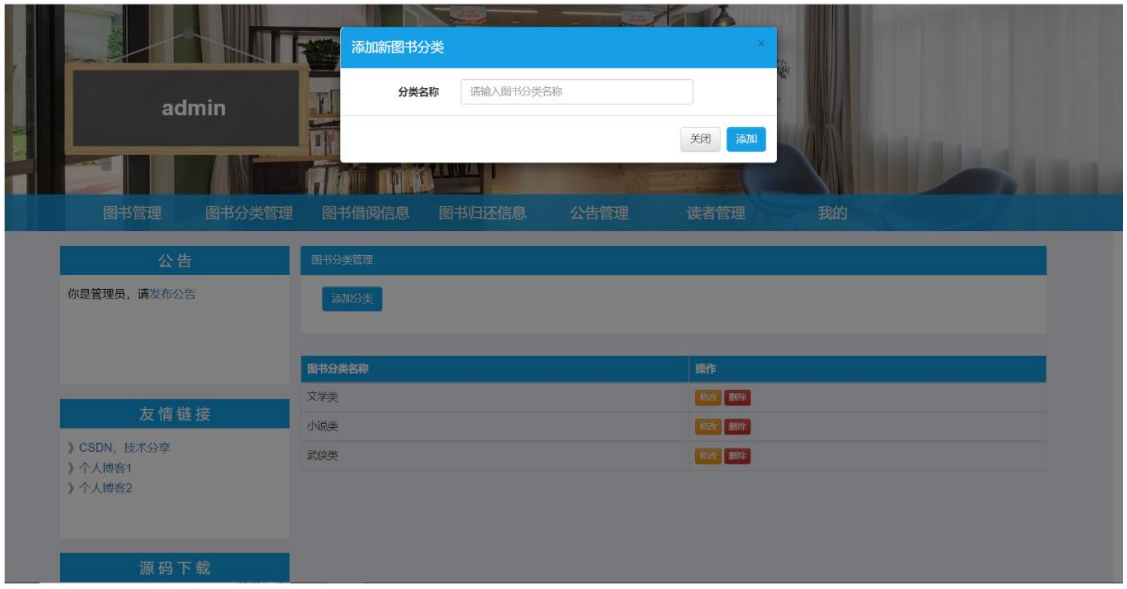
（2） 图书类别数量查询接口

2. 图书类别添加接口：

新建图书分类，之后可以在创建新书目的时候选择该类别

3. 图书类别名修改接口：

就是改名字，该类型的 ID 没有发生改变



新需求：

需要添加 Amount 字段，用来统计该类型的图书数量和关联图书具体信息，默认为 0，当操作发生时，数量自增。

表 2 图书分类表（BookType）

字段	数据类型	主键否	允许为空否	默认值
ID	int(11)	是	否	
BookTypeID	int(11)	否	否	
TypeName	varchar(64)	否	否	
Amount	int(11)	否	否	0

三、图书借阅信息

功能：显示所有的图书信息，包括图书信息、借阅者信息和时间信息。

借阅信息表很重要，主要用来存储、统计用户对图书的借阅和归还记录。当用户借阅了一本书，该表添加一条记录（包括用户名、书名啥的），flag = 1,表示借阅信息；当用户归还了一本书，该表也添加一条记录，flag = 0,表示归还信息。

```
// 还得再加一个借阅订单号，用来区分同一用户借阅同一本书的区别
// 把该字段设置为“唯一索引”，提升查询速度
```

- 普通索引(INDEX)：最基本的索引，没有任何限制
- 唯一索引(UNIQUE)：与"普通索引"类似，不同的就是：索引列的值必须唯一，但允许有空值。

1. 查询借阅信息接口：

展示所有图书的完整借阅信息（分页展示）

2. 查询归还信息接口：

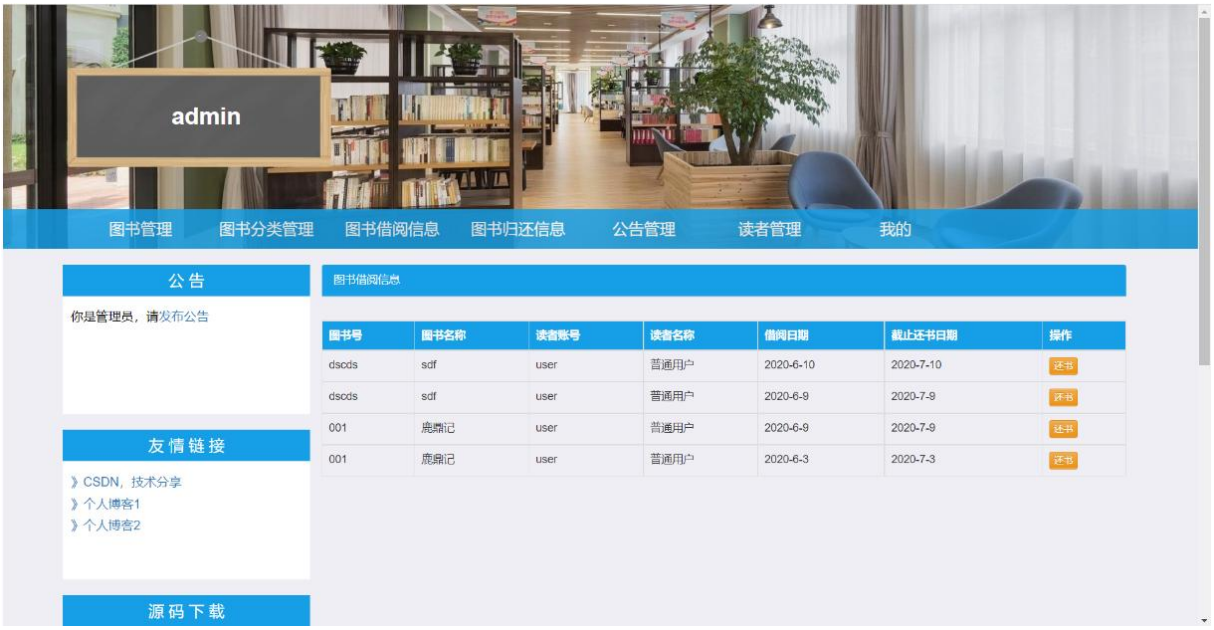
直接查询图书借阅信息表中 flag = 0 的记录。

3. 还书接口(和用户还书接口一致)：

根据借阅信息，在该表上（图书借阅信息表）上“删除”该条借阅记录（即 flag = 0），同时，在“图书信息表”中的图书数量字段中 **+1** (使用事务)。

需要调整的表：

图书借阅信息表中该记录的 flag = 0，图书归还日期更新至、图书信息表的该书数量 +1、读者信息表的该读者借阅数量 -1。



图书号	图书名称	读者账号	读者名称	借阅日期	还书日期
dscds	sdf	user	普通用户	2020-6-11	2020-6-11
dgsdg	普通用户	user	普通用户	2020-6-3	2020-6-9
001	西游记	user	普通用户	2020-6-3	2020-6-3
dgsdg	普通用户	user	普通用户	2020-6-3	2020-6-3
sss	s	user	普通用户	2020-6-3	2020-6-3
001	张	user	普通用户	2020-6-3	2020-6-3
sss	s	user	普通用户	2020-6-3	2020-6-3
001	张	user	普通用户	2020-6-3	2020-6-3
dgsdg	普通用户	null	null	2020-6-3	2020-6-3

表 3 图书借阅信息表（BookBorrowInformation）

字段	数据类型	主键否	允许为空否	默认值
ID	int(11)	是	否	
ISBN	varchar(64)	否	否	
BookName	varchar(64)	否	否	
UserIP	varchar(64)	否	否	
UserName	varchar(64)	否	是	NULL
BorrowDate	Date	否	否	
ReturnDate	Date	否	否	
BorrowingOrder	String	否	否	
Flag	Int(2)	否	否	1

测试记录	问题
第一次测试	1. 借阅日期和还书日期无法自动同步。
第二次测试	1. 点击还书功能，会归还所有借阅的同名图书，这会扰乱图书信息表中的数据自增方法。应改为归还指定图书。 2. 请求返回时 BorrowDate 和 ReturnDate 不显示。 3. BorrowDate 未更新。

GoFrame 中数据库如何连接?

1.首先需要连接数据库，得根据文档去查 config 的标准结构

```
link: "mysql:root:tyx123456@tcp(127.0.0.1:3306)/librarymanager"
```

2.然后根据文档排查语句

错误的：

```
pub, err :=
g.DB("librarymanager").Ctx(gctx.New()).Model("bookborrowinformatio
n").Where("ID", 2).All()
```

正确的：

```
pub, err :=
g.DB().Ctx(gctx.New()).Model("bookborrowinformation").Where("ID",
2).All()
```

为什么 g.DB()里面需要有库名?

3.model/do 有已连接的数据库结构

```
 g.Model("user")
// 或者
g.DB().Model("user")
```

gdb 的配置支持集群模式，数据库配置中每一项分组配置均可以是多个节点，支持负载均衡权重策略，例如：

```
database:
  default:
  - link: "mysql:root:12345678@tcp(127.0.0.1:3306)/test"
    role: "master"
  - link: "mysql:root:12345678@tcp(127.0.0.1:3306)/test"
    role: "slave"


  user:
  - link: "mysql:root:12345678@tcp(127.0.0.1:3306)/user"
    role: "master"
  - link: "mysql:root:12345678@tcp(127.0.0.1:3306)/user"
    role: "slave"
  - link: "mysql:root:12345678@tcp(127.0.0.1:3306)/user"
    role: "slave"
```

以上数据库配置示例中包含两个数据库分组 default 和 user，其中 default 分组包含一主一从，user 分组包含一主两从。

在代码中可以通过 g.DB() ---这个实际上就是调用默认的那个表了，和 g.DB("user")获取对应的数据库连接对象。

四、读者管理

功能： 显示所有已注册的读者信息，包括账号密码、姓名、邮箱、当前借阅数和历史借阅数。与此同时，当用户选择账号登录时，就得在该表中判断一下是否存在该用户。

注意：与以上表不同的是，该表是站在读者的角度。

1.读者信息查询接口：

直接返回所有读者的完整信息，包括用户账号、用户名、邮箱、当前借阅数和历史借阅数。

2.读者信息修改接口:

可以修改用户的各种信息，包括当前借阅数和历史借阅数。

需要与之更改的任务为：

```
1. // 如果仅修改用户的 Email 信息，则只需要在用户表上修改即可
2. // 如果修改了用户名，则随之修改的表为：用户信息表、图书借阅信息表(添加索引)
```

问题：多表联改

如果改了“读者管理表”中的读者名字，那么在“借阅信息表”中该读者的名字<在查询的时候>也要随之而变。

3.读者信息添加接口：

直接添加新用户，借阅数默认为 0。

4.读者信息删除接口：

直接删除该读者信息

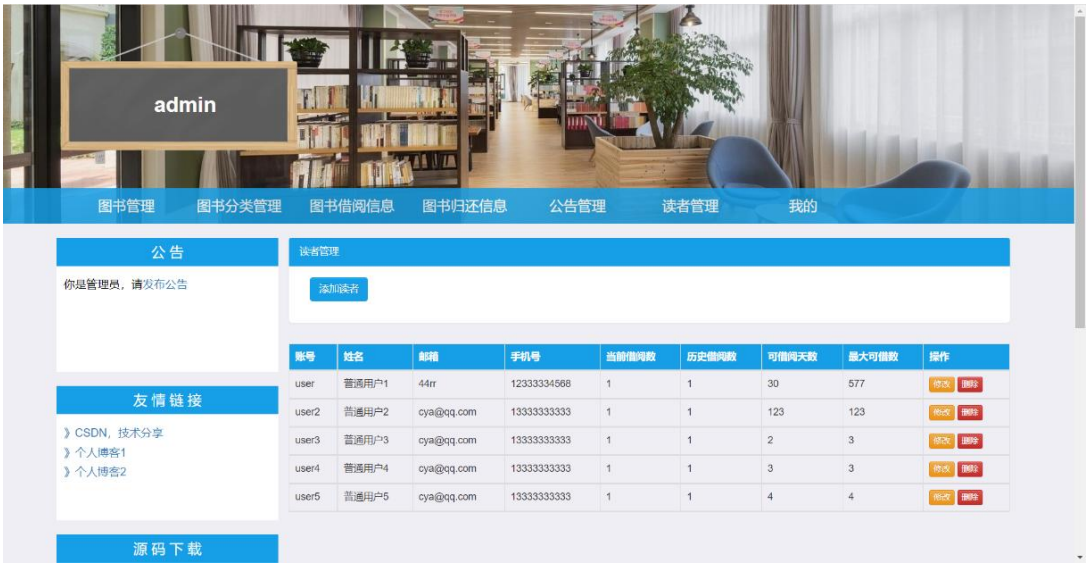


表 4 读者信息表（UserInformation）

字段	数据类型	主键否	允许为空否	默认值
ID	int(11)	是	否	
UserIP	varchar(64)	否	否	
UserName	varchar(64)	否	否	
Email	varchar(64)	否	是	NULL
CurrentNum	int(11)	否	是	0
HistoryNum	int(11)	否	是	0

用户端：

功能设计

一、图书查询

功能：显示所有还有余量的图书信息。与（表 1“图书信息表”）一致。

1. 查找接口：

输入图书名称等信息，直接给出对应的图书信息

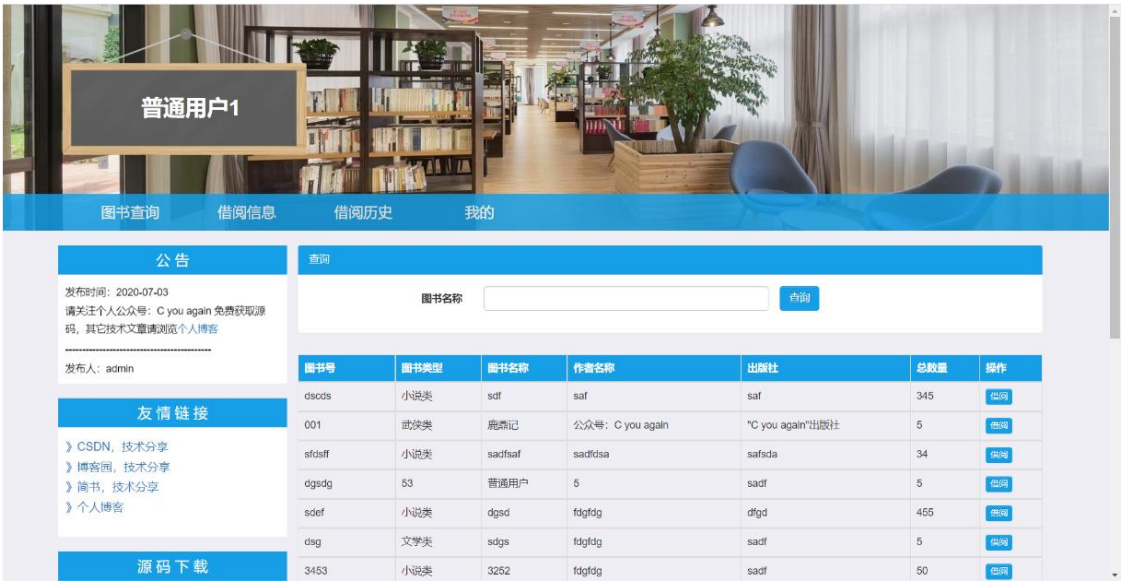
2. **借阅接口**(重要)：

根据“查找接口”得到的信息，通过书名等字段进行图书借阅。这时间所关联的表

有：图书信息表（该书数量-1）、图书借阅信息表、读者信息表。

点击借阅之后，后台会发生什么？

图书信息表的该书数量 -1、读者信息表的该读者借阅数量和历史数量同时 +1，图书借阅信息表的记录 +1



问题：

MySQL 表的 ID 设置自增之后，每次操作都增加一个序号，最后整张表变得非常混乱，有什么方法可以解决吗？

ID	BookName	ISBN	UserIP	UserName	created_at	ReturnDate	Flag	BorrowingOrder
1	人间物语	20210612	2016132	李四	2023-08-13	2023-08-31	1	10001
2	人生由我	20200901	20020305	赵六	2023-08-14	2023-08-14	0	10002
3	人生由我	20200901	20020305	赵六	2023-08-14	2023-08-14	0	10003
31	镜子	20130820	20090613	高十	2023-08-14	2023-10-23	1	10004

测试记录	问题
第一次测试	1.还书日期手动添加，改为自动设置
第二次测试	1.updated_at 更新时间在框架中被写死了，只要设置了 updated_at 字段，不管插入什么数据，都会被 updated_at(当前修改时间)所覆盖
第三次测试	1.数据库查询时，ReturnDate 和 BorrowingOrder 字段各占一条查询语句，代码冗余，合并为一个查询语句
第四次测试	1.MySQL 表的 ID 设置自增之后，每次操作都增加一个序号，最后整张表变得非常混乱。 2.未添加判空操作，即当前所要归还书目的 Flag = 0 怎么

	办？ 3.多次点击借书按钮，为什么有的正常，有的日期是错误的。如图 XX 所示：

43	活着	19901001	19822350	秦九	2023-08-16	2023-10-25	1	10043
44	人间物语	20210612	19822350	秦九	2023-08-16	NULL	1	NULL
45	人间物语	20210612	19822350	秦九	2023-08-16	NULL	1	NULL
46	人间物语	20210612	19822350	秦九	2023-08-16	NULL	1	NULL
47	镜子	20130820	19822350	秦九	2023-08-16	NULL	1	NULL
48	活着	19901001	19822350	秦九	2023-08-16	2023-10-25	1	10048
49	百年孤独	19880305	19822350	秦九	2023-08-16	2023-10-25	1	10049

二、借阅信息

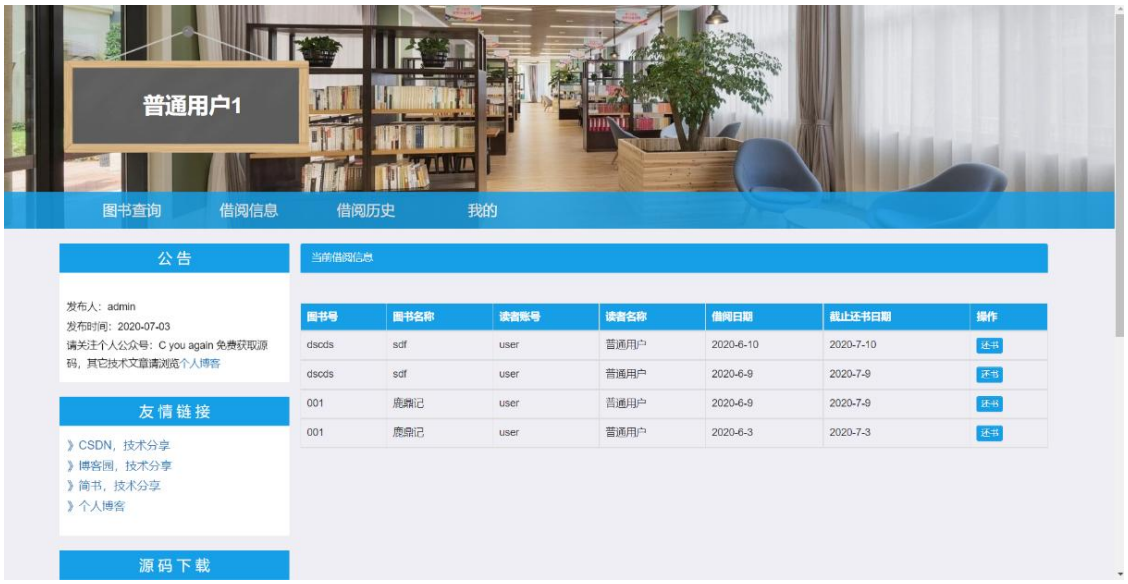
功能：用户可以查询到**自己**借阅中的书籍。

1. 查询接口：

查看当前用户在借阅信息表上的借阅信息，即查询“图书信息借阅表”中自己的名字和 flag = 1 的书目信息。其中，当前借阅数字段凭借 flag=1 的数量；历史借阅数字段凭借 flag=0 的数量。

2. 还书接口：

将该条信息塞进图书借阅信息表，让 flag = 0，同时在“图书信息表”中的该图书数量字段中 **+1。（两个表单独操作，查资料）**与四、图书借阅信息中 3.还书接口功能一致。

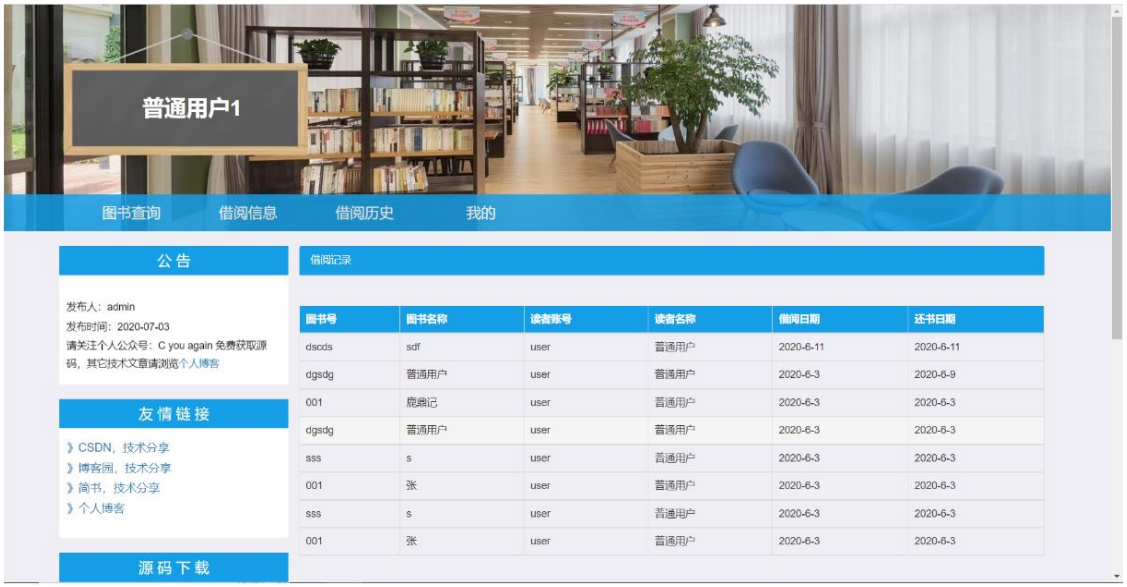


三、借阅历史

功能：用户可以查询到自己借阅中和已归还的所有图书信息

1. 个人借阅历史查询接口：

查询“图书信息借阅表”中用户名=当前用户名的所有图书信息。



四、个人信息管理(修改个人资料)

功能：用户修改自己的信息，仅能修改用户名和邮箱。

1. 查询个人资料接口

点击窗口后，弹出该用户的个人信息，个人信息包括：

- 用户 IP
- 用户名
- Email
- 当前借阅的图书数量、图书名：该书数量

```
1. // 分开操作，先查询个人信息，再查询图书数量
2. // 查询图书数量
2. select b.BookName, b.Flag from
3. bookborrowinformation b join userinformation u
4. on b.UserIP = u.UserIP
5. where b.UserIP=19822350
6. having b.Flag = 0;
```

测试记录	问题
第一次测试	1. 查询语句为：select BookName, count(1) from bookborrowinformation where UserIP=19822350 group by BookName

	having Flag = 0; 报错 Unknown column 'Flag' in 'having clause'
第二次测试	1. 在数据量比较大、并发请求量比较高的场景下不建议使用 Join 进行数据库联表查询 **考虑使用索引** **考虑使用缓存**

问题 1 的解决方案：

Flag 列在 HAVING 子句中被引用，但是在 SELECT 子句中并未出现，从而导致数据库引擎不知道如何处理这个列。

```
3. # 解决方案 1
4. # 将 Flag 列添加到 SELECT 子句中
5. 1. SELECT BookName, COUNT(1), Flag
6. 2. FROM bookborrowinformation
7. 3. WHERE UserIP = '19822350' AND Flag = 0
8. 4. GROUP BY BookName;
9. # 解决方案 2
10. 1. 将 HAVING 子句中的条件移到 WHERE 子句中
11. 1. SELECT BookName, COUNT(1)
12. 2. FROM bookborrowinformation
13. 3. WHERE UserIP = '19822350'
14. 4.   AND Flag = 0
15. 5. GROUP BY BookName;
```

优化点：使用索引

- 历史借阅图书数量

2. 修改个人资料接口

仅支持修改用户名和邮箱地址。