用户业务逻辑处理

本节核心内容

这一节是核心小节，讲解如何处理用户业务，这也是 API 的核心功能。本小节会讲解实际开发中需要的一些重要功能点，并根据笔者的开发经验，给出一些建议。功能点包括：

- 各种场景的业务逻辑处理
    - 创建用户
    - 删除用户
    - 更新用户
    - 查询用户列表
    - 查询指定用户的信息
- 数据库的 CURD 操作

> 本小节源码下载路径：[demo07 (https://github.com/lexkong/apiserver_demos/tree/mas](https://github.com/lexkong/apiserver_demos/tree/mas)
>
> 可先下载源码到本地，结合源码理解后续内容，边学边练。
>
> 本小节的代码是基于 [demo06 (https://github.com/lexkong/apiserver_demos/tree/mas](https://github.com/lexkong/apiserver_demos/tree/mas) 来开发的。

配置路由信息

需要先在 router/router.go 文件中，配置路由信息：

```
func Load(g *gin.Engine, mw ...gin.HandlerFunc)
*gin.Engine {
    ...
    // 用户路由设置
    u := g.Group("/v1/user")
    {
        u.POST("", user.Create)          // 创建用户
        u.DELETE("/:id", user.Delete)    // 删除用户
        u.PUT("/:id", user.Update)       // 更新用户
        u.GET("", user.List)             // 用户列表
        u.GET("/:username", user.Get)    // 获取指定
用户的详细信息
    }
    ...
    return g
}
```

> 在 RESTful API 开发中，API 经常会变动，为了兼容老的 API，引入了版本的概念，比如上例中的 /v1/user，说明该 API 版本是 v1。
>
> 很多 RESTful API 最佳实践文章中均建议使用版本控制，笔者这里也建议对 API 使用版本控制。

注册新的错误码

在 pkg/errno/code.go 文件中（详见 [demo07/pkg/errno/code.go](https://github.com/lexkong/apiserver_demos/blob/master/)

新增如下错误码：

```
var (
    // Common errors
        ...

    ErrValidation        = &Errno{Code: 20001,
Message: "Validation failed."}
    ErrDatabase          = &Errno{Code: 20002,
Message: "Database error."}
    ErrToken             = &Errno{Code: 20003,
Message: "Error occurred while signing the JSON
web token."}

    // user errors
    ErrEncrypt           = &Errno{Code: 20101,
Message: "Error occurred while encrypting the
user password."}
    ErrTokenInvalid      = &Errno{Code: 20103,
Message: "The token was invalid."}
    ErrPasswordIncorrect = &Errno{Code: 20104,
Message: "The password was incorrect."}
)
```

新增用户

更新 handler/user/create.go 中 Create() 的逻辑，更新后的内容见 [demo07/handler/user/create.go (https://github.com/lexkong/apiserver_demos/blob/master/](https://github.com/lexkong/apiserver_demos/blob/master/)

创建用户逻辑：

1. 从 HTTP 消息体获取参数（用户名和密码）

2. 参数校验
3. 加密密码
4. 在数据库中添加数据记录
5. 返回结果（这里是用户名）

从 HTTP 消息体解析参数，前面小节已经介绍了。

参数校验这里用的是 gopkg.in/go-playground/validator.v9 包（详见 go-playground/validator (https://github.com/go-playground/validator)），实际开发过程中，该包可能不能满足校验需求，这时候可在程序中加入自己的校验逻辑，比如在 handler/user/creater.go 中添加校验函数 checkParam：

```
package user

import (
    ...
)

// Create creates a new user account.
func Create(c *gin.Context) {
    log.Info("User Create function called.",
lager.Data{"X-Request-Id": util.GetReqID(c)})
    var r CreateRequest
    if err := c.Bind(&r); err != nil {
        SendResponse(c, errno.ErrBind, nil)
        return
    }

    if err := r.checkParam(); err != nil {
        SendResponse(c, err, nil)
        return
```

```
    }
        ...
}

func (r *CreateRequest) checkParam() error {
    if r.Username == "" {
        return errno.New(errno.ErrValidation,
nil).Add("username is empty.")
    }

    if r.Password == "" {
        return errno.New(errno.ErrValidation,
nil).Add("password is empty.")
    }

    return nil
}
```

例子通过 Encrypt() 对密码进行加密：

```
// Encrypt the user password.
func (u *UserModel) Encrypt() (err error) {
    u.Password, err = auth.Encrypt(u.Password)
    return
}
```

Encrypt() 函数引用 auth.Encrypt() 来进行密码加密，具体实现见 [demo07/pkg/auth/auth.go (https://github.com/lexkong/apiserver_demos/blob/master/](https://github.com/lexkong/apiserver_demos/blob/master/)

最后例子通过 u.Create() 函数来向数据库中添加记录，ORM 用的是 gorm，gorm 详细用法请参考 [GORM 指南 (http://gorm.io/zh_CN/docs/index.html)](http://gorm.io/zh_CN/docs/index.html)。在 Create() 函数中引

用的数据库实例是 DB.Self，该实例在 API 启动之前已经完成初始化。DB 是个全局变量，可以直接引用。

> 在实际开发中，为了安全，数据库中是禁止保存密码的明文信息的，密码需要加密保存。
>
> 笔者将接收和处理相关的 Go 结构体统一放在 handler/user/user.go 文件中，这样可以使程序结构更清晰，功能更聚焦。当然每个人习惯不一样，读者根据自己的习惯放置即可。handler/user/user.go 对 UserInfo 结构体的处理，也出于相同的目的。

删除用户

删除用户代码详见 [demo07/handler/user/delete.go (https://github.com/lexkong/apiserver_demos/blob/master/](https://github.com/lexkong/apiserver_demos/blob/master/)

删除时，首先根据 URL 路径 DELETE http://127.0.0.1/v1/user/1 解析出 id 的值 1，该 id 实际上就是数据库中的 id 索引，调用 model.DeleteUser() 函数删除，函数详见 [demo07/model/user.go (https://github.com/lexkong/apiserver_demos/blob/master/](https://github.com/lexkong/apiserver_demos/blob/master/)

更新用户

更新用户代码详见 [demo07/handler/user/update.go (https://github.com/lexkong/apiserver_demos/blob/master/](https://github.com/lexkong/apiserver_demos/blob/master/)

更新用户逻辑跟创建用户差不多，在更新完数据库字段后，需要指定 gorm model 中的 id 字段的值，因为 gorm 在更新时默认是按照 id 来匹配记录的。通过解析 PUT http://127.0.0.1/v1/user/1 来获取 id。

查询用户列表

查询用户列表代码详见 [demo07/handler/user/list.go (https://github.com/lexkong/apiserver_demos/blob/master/](https://github.com/lexkong/apiserver_demos/blob/master/)

一般在 handler 中主要做解析参数、返回数据操作，简单的逻辑也可以在 handler 中做，像新增用户、删除用户、更新用户，代码量不大，所以也可以放在 handler 中。有些代码量很大的逻辑就不适合放在 handler 中，因为这样会导致 handler 逻辑不是很清晰，这时候实际处理的部分通常放在 service 包中。比如本例的 LisUser() 函数：

```
package user

import (
    "apiserver/service"
    ...
)

// List list the users in the database.
func List(c *gin.Context) {
    ...
    infos, count, err :=
service.ListUser(r.Username, r.Offset, r.Limit)
    if err != nil {
        SendResponse(c, err, nil)
        return
    }
    ...
}
```

查询一个 REST 资源列表，通常需要做分页，如果不做分页返回的列表过多，会导致 API 响应很慢，前端体验也不好。本例中的查询函数做了分页，收到的请求中传入的 offset 和 limit 参数，分别对应于 MySQL 的 offset 和 limit。

service.ListUser() 函数用来做具体的查询处理，代码详见 [demo07/service/service.go](https://github.com/lexkong/apiserver_demos/blob/master/

在 ListUser() 函数中用了 sync 包来做并行查询，以使响应延时更小。在实际开发中，查询数据后，通常需要对数据做一些处理，比如 ListUser() 函数中会对每个用户记录返回一个 sayHello 字段。sayHello 只是简单输出了一个 Hello shortId 字符串，其中 shortId 是通过 util.GenShortId() 来生成的（GenShortId 实现详见 [demo07/util/util.go](https://github.com/lexkong/apiserver_demos/blob/master/
像这类操作通常会增加 API 的响应延时，如果列表条目过多，列表中的每个记录都要做一些类似的逻辑处理，这会使得整个 API 延时很高，所以笔者在实际开发中通常会做并行处理。根据笔者经验，效果提升十分明显。

读者应该已经注意到了，在 ListUser() 实现中，有 sync.Mutex 和 IdMap 等部分代码，使用 sync.Mutex 是因为在并发处理中，更新同一个变量为了保证数据一致性，通常需要做锁处理。

使用 IdMap 是因为查询的列表通常需要按时间顺序进行排序，一般数据库查询后的列表已经排过序了，但是为了减少延时，程序中用了并发，这时候会打乱排序，所以通过 IdMap 来记录并发处理前的顺序，处理后再重新复位。

获取指定用户的详细信息

代码详见 demo07/handler/user/get.go (https://github.com/lexkong/apiserver_demos/blob/master/

获取指定用户信息时，首先根据 URL 路径 GET http://127.0.0.1/v1/user/admin 解析出 username 的值 admin，然后调用 model.GetUser() 函数查询该用户的数据库记录并返回，函数详见 demo07/model/user.go (https://github.com/lexkong/apiserver_demos/blob/master/

编译并运行

1. 下载 apiserver_demos 源码包（如前面已经下载过，请忽略此步骤）

```
$ git clone
https://github.com/lexkong/apiserver_demos
```

2. 将 apiserver_demos/demo07 复制为 $GOPATH/src/apiserver

```
$ cp -a apiserver_demos/demo07/
$GOPATH/src/apiserver
```

3. 在 apiserver 目录下编译源码

```
$ cd $GOPATH/src/apiserver
$ gofmt -w .
$ go tool vet .
$ go build -v .
```

创建用户

```
$ curl -XPOST -H "Content-Type: application/json"
http://127.0.0.1:8080/v1/user -
d'{"username":"kong","password":"kong123"}'

{
  "code": 0,
  "message": "OK",
  "data": {
    "username": "kong"
  }
}
```

查询用户列表

```
$ curl -XGET -H "Content-Type: application/json"
http://127.0.0.1:8080/v1/user -d'{"offset": 0,
"limit": 20}'

{
  "code": 0,
  "message": "OK",
  "data": {
    "totalCount": 2,
    "userList": [
      {
        "id": 2,
        "username": "kong",
        "sayHello": "Hello qhXO5iIig",
        "password":
"$2a$10$vE9jG71oyzstWVwB/QfU3u00Pxb.ye8hFIDvnyw60
nHBv/xsJZoUO",
        "createdAt": "2018-06-02 14:47:54",
        "updatedAt": "2018-06-02 14:47:54"
```

```
      },
      {
        "id": 0,
        "username": "admin",
        "sayHello": "Hello qhXO5iSmgz",
        "password":
"$2a$10$veGcArz47VGj7l9xN7g2iuT9TF21jLI1YGXarGzvA
RNdnt4inC9PG",
        "createdAt": "2018-05-28 00:25:33",
        "updatedAt": "2018-05-28 00:25:33"
      }
    ]
  }
}
```


可以看到，新增了一个用户 kong，数据库 id 索引为 2。admin 用户是上一节中初始化数据库时初始化的。

> 笔者建议在 API 设计时，对资源列表进行分页。

获取用户详细信息

```
$ curl -XGET -H "Content-Type: application/json"
http://127.0.0.1:8080/v1/user/kong

{
  "code": 0,
  "message": "OK",
  "data": {
    "username": "kong",
    "password":
"$2a$10$vE9jG71oyzstWVwB/QfU3u00Pxb.ye8hFIDvnyw60
nHBv/xsJZoUO"
  }
}
```

更新用户

在 **查询用户列表** 部分，会返回用户的数据库索引。例如，用户 kong 的数据库 id 索引是 2，所以这里调用如下 URL 更新 kong 用户：

```
$ curl -XPUT -H "Content-Type: application/json"
http://127.0.0.1:8080/v1/user/2 -
d'{"username":"kong","password":"kongmodify"}'

{
  "code": 0,
  "message": "OK",
  "data": null
}
```

获取 kong 用户信息：

```
$ curl -XGET -H "Content-Type: application/json"
http://127.0.0.1:8080/v1/user/kong

{
  "code": 0,
  "message": "OK",
  "data": {
    "username": "kong",
    "password":
"$2a$10$E0kwtmtLZbwW/bDQ8qI8e.eHPqhQOW9tvjwpyo/p0
5f/f4Qvr3OmS"
  }
}
```

可以看到密码已经改变（旧密码为 $2a$10$vE9jG71oyzstWVwB/QfU3u00Pxb.ye8hFIDvnyw60nH

删除用户

在 **查询用户列表** 部分，会返回用户的数据库索引。例如，用户 kong 的数据库 id 索引是 2，所以这里调用如下 URL 删除 kong 用户：

```
$ curl -XDELETE -H "Content-Type:
application/json" http://127.0.0.1:8080/v1/user/2

{
  "code": 0,
  "message": "OK",
  "data": null
}
```

获取用户列表：

```
$ curl -XGET -H "Content-Type: application/json"
http://127.0.0.1:8080/v1/user -d'{"offset": 0,
"limit": 20}'

{
  "code": 0,
  "message": "OK",
  "data": {
    "totalCount": 1,
    "userList": [
      {
        "id": 0,
        "username": "admin",
        "sayHello": "Hello EnqntiSig",
        "password":
"$2a$10$veGcArz47VGj7l9xN7g2iuT9TF21jLI1YGXarGzvA
RNdnt4inC9PG",
        "createdAt": "2018-05-28 00:25:33",
        "updatedAt": "2018-05-28 00:25:33"
      }
    ]
  }
}
```

可以看到用户 kong 未出现在用户列表中，说明他已被成功删除。

小结

本小节通过对用户增删改查和查询列表的操作，介绍了实际开发中如何对 REST 资源进行操作，并结合笔者的实际开发经验给出了一些开发习惯和建议。