读取和返回 HTTP 请求

本节核心内容

- 如何读取 HTTP 请求数据
- 如何返回数据
- 如何定制业务的返回格式

> 本小节源码下载路径：demo06 (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 demo05 (https://github.com/lexkong/apiserver_demos/tree/mas 来开发的。

读取和返回参数

在业务开发过程中，需要读取请求参数（消息体和 HTTP Header），经过业务处理后返回指定格式的消息。apiserver 也展示了如何进行参数的读取和返回，下面展示了如何读取和返回参数：

读取 HTTP 信息： 在 API 开发中需要读取的参数通常为：HTTP Header、路径参数、URL参数、消息体，读取这些参数可以直接使用 gin 框架自带的函数：

- Param()：返回 URL 的参数值，例如

```
router.GET("/user/:id", func(c *gin.Context) {
    // a GET request to /user/john
    id := c.Param("id") // id == "john"
})
```

- Query()：读取 URL 中的地址参数，例如

```
 // GET /path?id=1234&name=Manu&value=
 c.Query("id") == "1234"
 c.Query("name") == "Manu"
 c.Query("value") == ""
 c.Query("wtf") == ""
```

- DefaultQuery()：类似 Query()，但是如果 key 不存在，会返回默认值，例如

```
//GET /?name=Manu&lastname=
c.DefaultQuery("name", "unknown") == "Manu"
c.DefaultQuery("id", "none") == "none"
c.DefaultQuery("lastname", "none") == ""
```

- Bind()：检查 Content-Type 类型，将消息体作为指定的格式解析到 Go struct 变量中。apiserver 采用的媒体类型是 JSON，所以 Bind() 是按 JSON 格式解析的。

- GetHeader()：获取 HTTP 头。

返回HTTP消息： 因为要返回指定的格式，apiserver 封装了自己的返回函数，通过统一的返回函数 SendResponse 来格式化返回，小节后续部分有详细介绍。

增加返回函数

API 返回入口函数，供所有的服务模块返回时调用，所以这里将入口函数添加在 handler 目录下，handler/handler.go 的源码为：

```
package handler

import (
    "net/http"

    "apiserver/pkg/errno"

    "github.com/gin-gonic/gin"
)

type Response struct {
    Code    int         `json:"code"`
    Message string      `json:"message"`
    Data    interface{} `json:"data"`
}

func SendResponse(c *gin.Context, err error, data
interface{}) {
    code, message := errno.DecodeErr(err)

    // always return http.StatusOK
    c.JSON(http.StatusOK, Response{
        Code:    code,
        Message: message,
        Data:    data,
    })
}
```

可以看到返回格式固定为：

```
type Response struct {
    Code    int         `json:"code"`
    Message string      `json:"message"`
    Data    interface{} `json:"data"`
}
```

在返回结构体中，固定有 Code 和 Message 参数，这两个参数通过函数 DecodeErr() 解析 error 类型的变量而来（DecodeErr() 在上一节介绍过）。Data 域为 interface{} 类型，可以根据业务自己的需求来返回，可以是 map、int、string、struct、array 等 Go 语言变量类型。SendResponse() 函数通过 errno.DecodeErr(err) 来解析出 code 和 message，并填充在 Response 结构体中。

在业务处理函数中读取和返回数据

通过改写上一节 handler/user/create.go 源文件中的 Create() 函数，来演示如何读取和返回数据，改写后的源码为：

```
package user

import (
    "fmt"

    . "apiserver/handler"
    "apiserver/pkg/errno"

    "github.com/gin-gonic/gin"
    "github.com/lexkong/log"
)

// Create creates a new user account.
func Create(c *gin.Context) {
```

```
    var r CreateRequest
    if err := c.Bind(&r); err != nil {
        SendResponse(c, errno.ErrBind, nil)
        return
    }

    admin2 := c.Param("username")
    log.Infof("URL username: %s", admin2)

    desc := c.Query("desc")
    log.Infof("URL key param desc: %s", desc)

    contentType := c.GetHeader("Content-Type")
    log.Infof("Header Content-Type: %s",
contentType)

    log.Debugf("username is: [%s], password is
[%s]", r.Username, r.Password)
    if r.Username == "" {
        SendResponse(c,
errno.New(errno.ErrUserNotFound,
fmt.Errorf("username can not found in db:
xx.xx.xx.xx")), nil)
        return
    }

    if r.Password == "" {
        SendResponse(c, fmt.Errorf("password is
empty"), nil)
    }

    rsp := CreateResponse{
        Username: r.Username,
```

```
    }

    // Show the user information.
    SendResponse(c, nil, rsp)
}
```

这里也需要更新下路由，router/router.go（详见 [demo06/router/router.go](https://github.com/lexkong/apiserver_demos/blob/master/d) (https://github.com/lexkong/apiserver_demos/blob/master/d

上例展示了如何通过 Bind()、Param()、Query() 和 GetHeader() 来获取相应的参数。

根据笔者的研发经验，建议：如果消息体有 JSON 参数需要传递，针对每一个 API 接口定义独立的 go struct 来接收，比如 CreateRequest 和 CreateResponse，并将这些结构体统一放在一个 Go 文件中，以方便后期维护和修改。这样做可以使代码结构更加规整和清晰，本例统一放在 handler/user/user.go 中，源码为：

```
package user

type CreateRequest struct {
    Username string `json:"username"`
    Password string `json:"password"`
}

type CreateResponse struct {
    Username string `json:"username"`
}
```

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/demo06 复制为 $GOPATH/src/apiserver

```
$ cp -a apiserver_demos/demo06/
$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 .
```

测试

启动apiserver：./apiserver，发送 HTTP 请求：

```
$ curl -XPOST -H "Content-Type: application/json" http://127.0.0.1:8080/v1/user/admin2?desc=test -d'{"username":"admin","password":"admin"}'

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

查看 apiserver 日志：

```
[api@centos apiserver]$./apiserver
[GIN-debug] [WARNING] Running in "debug" mode. Switch to "release" mode in production.
 - using env:   export GIN_MODE=release
 - using code:  gin.SetMode(gin.ReleaseMode)

[GIN-debug] POST   /v1/user/:username        --> apiserver/handler/user.Create (5 handlers)
[GIN-debug] GET    /sd/health                --> apiserver/handler/sd.HealthCheck (5 handlers)
[GIN-debug] GET    /sd/disk                  --> apiserver/handler/sd.DiskCheck (5 handlers)
[GIN-debug] GET    /sd/cpu                   --> apiserver/handler/sd.CPUCheck (5 handlers)
[GIN-debug] GET    /sd/ram                   --> apiserver/handler/sd.RAMCheck (5 handlers)
{"level":"INFO","timestamp":"2018-06-01 15:31:37.316","file":"apiserver/main.go:59","msg":"Start to listening the incoming requests on http address: :8080"}
{"level":"INFO","timestamp":"2018-06-01 15:31:37.316","file":"apiserver/main.go:56","msg":"The router has been deployed successfully."}
{"level":"INFO","timestamp":"2018-06-01 15:31:41.307","file":"user/create.go:22","msg":"URL username: admin2"}
{"level":"INFO","timestamp":"2018-06-01 15:31:41.307","file":"user/create.go:25","msg":"URL key param desc: test"}
{"level":"INFO","timestamp":"2018-06-01 15:31:41.307","file":"user/create.go:28","msg":"Header Content-Type: application/json"}
{"level":"DEBUG","timestamp":"2018-06-01 15:31:41.307","file":"user/create.go:30","msg":"username is: [admin], password is [admin]"}
```

可以看到成功读取了请求中的各类参数。并且 curl 命令返回的结果格式为指定的格式：

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

小结

本小节介绍了如何进行 HTTP 请求的读取和返回，读取主要用 gin 框架自带的函数，返回要统一用函数 SendResponse。