在网络应用程序中进行设置

为了让其他用户也可以使用应用程序，最好的选择是做Web应用程序，Go语言本身提供了一个Web服务器来处理HTTP请求，并为HTML提供模板。

新建web目录，包含三个其他目录的目录

- `web/tpl`：包含所有的HTML页面（模板）
- `web/static`：包含所有CSS，Javascript，图片...
- `web/controllers` ：包含将呈现模板的所有函数

使用MVC（模型 - 视图 - 控制器）模式使其更具可读性。模型将是区块链部分，视图是模板和控制器由`controllers`目录中的功能提供

- `web/tpl/login.html`
- `web/tpl/issue.html`
- `web/tpl/bills.html`
- `web/tpl/billInfo.html`
- `web/tpl/waitAccept.html`
- `web/tpl/waitAcceptInfo.html`
- `web/controller/controllerHandler.go`
- `web/controller/controllerResponse.go`
- `web/controllers/userInfo.go`
- `web/app.go`
- `web/static`

在`web`中添加`controller`目录

在 `controller` 目录中新建 `controllerHandler` 用于处理各种请求

在 `controller` 目录中新建 `controllerResponse` 用于响应请求

用户登录: `userInfo.go`

系统指定用户: `admin, alice, bob, jack`

初始化用户

```
$ vim web/controller/userInfo.go
```

```
package controller

type User struct {
    UserName    string  `json:"UserName"`
    Name        string  `json:"Name"`
    Password    string  `json:"Password"`
    CmId        string  `json:"CmId"`
    Acct        string  `json:"Acct"`
}

var Users []User

func init()  {
    admin := User{UserName: "admin", Name: "管理员", Password:
"123456", CmId: "HODR01", Acct: "管理员"}
    alice := User{UserName: "alice", Name: "A公司", Password:
"123456", CmId: "ACMID", Acct: "A公司"}
    bob := User{UserName: "bob", Name: "B公司", Password: "123456",
CmId: "BCMID", Acct: "B公司"}
    jack := User{UserName: "jack", Name: "C公司", Password:
"123456", CmId: "CCMID", Acct: "C公司"}
```

```
    Users = append(Users, admin)
    Users = append(Users, alice)
    Users = append(Users, bob)
    Users = append(Users, jack)
}
```

指定响应处理

```
$ vim web/controller/controllerResponse.go
```

```
package controller

import (
    "net/http"
    "path/filepath"
    "html/template"
    "fmt"
)

func showView(w http.ResponseWriter, r *http.Request, templateName
string, data interface{})  {
    page := filepath.Join("web", "tpl", templateName)

    // 创建模板实例
    resultTemplate, err := template.ParseFiles(page)
    if err != nil {
        fmt.Println("创建模板实例错误: ", err)
        return
    }

    // 融合数据
    err = resultTemplate.Execute(w, data)
    if err != nil {
        fmt.Println("融合模板数据时发生错误", err)
        return
```

```
    }
}
```

请求处理控制器

```
$ vim web/controller/controllerHandler.go
```

```
package controller

import (
    "github.com/kongyixueyuan.com/bill/service"
    "net/http"
    "encoding/json"
    "fmt"
)

var cuser User

type Application struct {
    Fabric *service.FabricSetupService
}

func (app *Application) LoginView(w http.ResponseWriter, r
*http.Request){
    showView(w, r, "login.html", nil)
}

func (app *Application) IssueView(w http.ResponseWriter, r
*http.Request)  {
    data := &struct {
        Cuser User
        Flag bool
    }{
        Cuser:cuser,
        Flag:false,
```

```
    }
    showView(w, r, "issue.html", data)
}

func (app *Application) Loginout(w http.ResponseWriter, r
*http.Request)  {
    cuser = User{}
    showView(w, r, "login.html", nil)
}

func (app *Application) Login(w http.ResponseWriter, r
*http.Request){
    // 接收请求参数

    // 封装响应数据

    // 验证请求数据的正确性

    // 根据结果响应不同View
}

// 查询我的票据列表
func (app *Application) QueryMyBills(w http.ResponseWriter, r
*http.Request)  {
    //holdeCmId := r.FormValue("holdeCmId")
    holdeCmId := cuser.CmId
    // 调用业务层

    if err != nil{
        fmt.Println("查询当前用户的票据列表失败: ", err.Error())
    }
    // 反序列化并封装响应数据

    // 响应客户端
}

// 发布票据
```

```
func (app *Application) Issue(w http.ResponseWriter, r
*http.Request)  {
    // 获取请求提交数据

    // 调用业务层

    // 封装响应数据

    // 响应客户端

}

// 查询票据详情
func (app *Application) QueryBillInfo(w http.ResponseWriter, r
*http.Request)  {
    // 获取提交数据

    // 调用业务层，反序列化

    // 封装响应数据

    // 响应客户端
}

// 票据背书请求
func (app *Application) Endorse(w http.ResponseWriter, r
*http.Request)  {
    // 获取提交数据

    // 调用业务层

    // 封装响应数据

    // 响应客户端
}

// 查询待背书票据列表
```

```
func (app *Application) WaitEndorBills(w http.ResponseWriter, r
*http.Request)  {
    // 获取提交数据

    // 调用业务层，反序列化

    // 封装响应数据

    // 响应客户端
}

// 查询待背书票据详情
func (app *Application) WaitEndorseInfo(w http.ResponseWriter, r
*http.Request)  {
    // 获取提交数据

    // 调用业务层，反序列化

    // 封装响应数据

    // 响应客户端
}

// 处理背书请求
func (app *Application) EndorseOper(w http.ResponseWriter, r
*http.Request)  {
    // 获取提交数据

    // 调用业务层，反序列化

    // 封装响应数据

    // 响应客户端
}
```

添加路由信息

在 `web` 目录中添加 `app.go` 文件

```
$ vim web/app.go
```

编辑 `app.go`

```
package web

import (
    "net/http"
    "fmt"
    "github.com/kongyixueyuan.com/bill/web/controller"
)

func  WebStart(app *controller.Application)  {
    // 指定文件服务器
    fs := http.FileServer(http.Dir("web/static"))
    http.Handle("/static/", http.StripPrefix("/static/", fs))

    http.HandleFunc("/", app.LoginView)
    http.HandleFunc("/login.html", app.LoginView)
    http.HandleFunc("/issue.html", app.IssueView)
    http.HandleFunc("/loginout", app.Loginout)

    http.HandleFunc("/userLogin", app.Login)
    http.HandleFunc("/myBills", app.QueryMyBills)
    http.HandleFunc("/billInfo", app.QueryBillInfo)
    http.HandleFunc("/endorse", app.Endorse)
    http.HandleFunc("/issue", app.Issue)
    http.HandleFunc("/waitEndorBills", app.WaitEndorBills)
    http.HandleFunc("/waitEndorseInfo", app.WaitEndorseInfo)
    http.HandleFunc("/endorseOper", app.EndorseOper)

    fmt.Println("启动Web服务, 监听端口号: 9000")

    err := http.ListenAndServe(":9000", nil)
    if err != nil {
        fmt.Println("启动Web服务错误")
```

```
    }

}
```

启动Web服务

最后**编辑** `main.go` ，以便使用Web界面实现应用程序

```
$ vim main.go
```

添加如下内容:

```
func main(){}
    [......]

    app := controller.Application{
            Fabric: fsService,
        }
        web.WebStart(&app)
}
```

打开浏览器访问: localhost:9000/


区块链部落
专注于区块链技术
识别图中二维码关注我们