go 语言与区块链技术：期中设计

姓名：于汇洋

学号：2018150801020

所属学院和班级：经济与管理学院互联网金融

一、实验要求：

用 Go 设计一个 Web 的计算器，计算放到后台，以 Web 方式输入计算数据，传入到 Go 服务后端进行计算，要求至少实现四则运算，$\sqrt[2]{x}, x^2, \frac{1}{x}$。也可以实现 sin,cos 等其它功能（加分）。形成报告，16 周，周三提交。

结果示例：

二、实验内容

1. 实验分析：

实验过程分为两个部分：前端，要在网页上打开这个计算器；后端执行一系列计算，使用 golang。即：

前端：js、html、css；

后端：go；

功能：基本的计算器功能+sin、cos 等等；

2. 代码实现（使用的编译器是 VIsual Studio Code）

main.go 文件

```
package main

import (
   "encoding/json"
   "fmt"
   "io/ioutil"
   "log"
   "math"
   "net/http"
   "strconv"
   "unicode"
)

var ERROR_EXPRESSION = "1001"
var SUCCESS = "2000"
```

```
var msgCodeTable = map[string]string {
   ERROR_EXPRESSION: "表达式错误",
   SUCCESS: "计算成功",
}

/*
web 程序分为两个端，前端（浏览器页面），和后端（处理前端的请求），比如一个注册页面，用户再前端填写好用户名密码后，点击登录按钮，会把已经填写好的用户名、密码通过 Http 请求发送给服务器（也就是后端程序）。前后端本质上是分开的两个项目，他们通过 Http 请求建立连接。
*/
func main(){
   registRouter()
 /*注册路由: 简单的说就是配置后端程序中的哪一个函数处理前端发过来的哪一个请求（因为请求那么多，有登录的、有注册的，不同请求用不同函数处理）*/

   error := http.ListenAndServe(":8080", nil)       // 启动服务器，监听端口

   fmt.Println()
   handleError(error)
}

func registRouter(){
   http.HandleFunc("/index", HandleIndex)
/* 这里是处理 localhost:8080/index 这个请求，当前端访问这个 URL 时候，由 HandleIndex 这个函数处理，以下也是一样。*/
   http.Handle("/js/",http.StripPrefix("/js/",http.FileServer(http.Dir("./js"))))
/* /index（简写，省略了主机名和端口）主要是返回前端 html
   http.HandleFunc("/result", HandleGetValue)  // /js/这一条是处理静态资源（css 样式文件、js 文件、图片称为静态资源），39 行的意思是将/js/这个 url 路径映射到./js 这个目录下，直接让前端访问（就不用读取文件在返回了）
*/

}  // 40 行处理 /result 这个 URL，在项目中就是用户点击 = 号后的请求。


/*
处理用户点击等号的请求。前端传来的是一个字符串（要计算的表达式）
*/
func HandleGetValue(w http.ResponseWriter, r *http.Request){
   log.Println("【请求】/result, 【method】" + r.Method)
   if r.Method == "GET" {
    if err:=r.ParseForm(); err!=nil {
        log.Println("【error】参数解析失败！")
        return
    }

    expression := r.Form["expression"][0]        // 获取前端数据
```

```
    log.Println("【expression】" + expression)
    msgCode, value := GetValueByExpression(expression)
 /* 这个函数时核心，计算字符串的表达式。这里调用计算字符串获取值的函数之后，将结果返回给前端*/
    resultMap := map[string]string{
        "MsgCode": msgCode,
        "result": strconv.FormatFloat(value, 'f', 2, 32),
    }
    resultJson, _ := json.Marshal(resultMap)
    fmt.Fprintf(w, string(resultJson))
 /* 可以看到，和前面处理 /index 时类似的，也是将结果字符串输入到 w 中（w 代表 writer，给前端写入数据的意思）*/

   }  // 还要说明的是，返回的是要给 json 字符串。一种轻量级的数据交换格式。

}

func GetValueByExpression(expression string) (string, float64){
/* 传过来的表达式时中缀表达式*/
   if len(expression)>1 && expression[0] == '-'{
// 计算机计算中缀表达式要首先转换为后缀表达式（也有其他方法，但是我用的时这一种）
    expression = "0" + expression

// 转换为后缀表达式之后，在利用栈计算表达式的值。 }

   resultStack := make([]string, 0)
   operateStack, top := make([]string, len(expression)), -1
   lastIndex := len(expression)
   // 中缀转后缀
   for i:=0; i<lastIndex;  i++{
    switch {
    case unicode.IsDigit(rune(expression[i])):
   // 如果是数字，直接输出到结果字符串中
        beginIndex := i
        for i<lastIndex && isNumber(expression[i]){
            i++
        }

        if i<lastIndex && expression[i] == '.' {    // 是浮点数

            i++ // 跳过'.'

            for i<lastIndex && isNumber(expression[i]){
                i++
            }
        }
        numStr := expression[beginIndex: i]
        resultStack = append(resultStack, numStr)
```

```
        i-- // 回退

    case expression[i] == '(':      // 遇到左括号直接入栈

        top++
        operateStack[top] = string(expression[i])

    case expression[i] == ')'://遇到右括号将操作栈中，第一个左括号之前符号输出

        for operateStack[top] != "("  {
            resultStack = append(resultStack, operateStack[top])
            top--
        }

        top-- // 将左括号出栈

    case expression[i]=='+' || expression[i]=='-':
// +、-是低优先级的操作符，因此要将栈中的高优先级操作符先出栈
        for    top    >    -1    &&    operateStack[top]!=string(expression[i])    &&
operateStack[top]!="(" {
            resultStack = append(resultStack, operateStack[top])
            top--
        }
        top++

        operateStack[top] = string(expression[i])     // 将+、-操作符入栈

    case expression[i]=='*' || expression[i]=='/':
// 遇到乘除、函数都直接入栈
        top++
        operateStack[top] = string(expression[i])

    default:   // 遇到函数的情况。如上所述，直接入栈。

        beginIndex := i
        for unicode.IsLetter(rune(expression[i])) {
            i++
        }
        top++
        operateStack[top] = expression[beginIndex: i]

        i-- // 回退

    }
   }


   for top >= 0 {  // 将剩余的符号输出到结果字符串中

    resultStack = append(resultStack, operateStack[top])
```

```
 top--
}

// 计算后缀表达式的值
valueStack := make([]float64, len(expression))
top = -1
for i:=0; i<len(resultStack); i++ {
 curStr := resultStack[i]

 if unicode.IsDigit(rune(curStr[0])) { // 当前元素是否为数值，直接入栈

    value, _ := strconv.ParseFloat(curStr, 32)
    top++
    valueStack[top] = float64(value)

 } else if curStr == "+" || curStr=="-" || curStr== "*" || curStr == "/"{ // 如
果是双元操作符，将 valueStack 顶部两个栈出栈
    if top == 0 {

        return "1000", 0  // 需要两个数进行操作，却只有一个数，则表达式错误

    }
    secondValue := valueStack[top]
    top--
    topValue := valueStack[top]
    var newValue float64 = 0.0
    switch curStr {
    case "+": newValue = topValue + secondValue
    case "-": newValue = topValue - secondValue
    case "*": newValue = topValue * secondValue
    case "/": newValue = topValue / secondValue
    }
    valueStack[top] = newValue

 } else {  // 当前操作符为函数

    topValue := valueStack[top]
    var newValue float64 = 0.0
    switch curStr {
    case "sin": newValue = math.Sin(topValue)
    case "cos": newValue = math.Cos(topValue)
    case "sqrt": newValue = math.Sqrt(topValue)
    }
    valueStack[top] = newValue
 }
}
return "2000", valueStack[0]
```

```
/* 这里要解释以下这个函数的返回值。双返回值，一个字符串，MsgCode，代表计算的状态（表达式错误无法计算、计算成功），一个 float64，代表计算成功的值*/
}      // 2000 代表计算成功，1000 代表表达式错误。

func isOperateChar(u uint8) bool {
   if u == '+' || u == '-' || u == '*' || u == '/' {
    return true
   }
   return false
}

func isNumber(targetChar uint8) bool {
   if value:=targetChar-'0'; value<=9 {
    return true
   }
   return false
}

func handleError(err error){
   if err != nil {
    log.Fatalln(err)
   }
}

func HandleIndex(w http.ResponseWriter, r *http.Request){
   log.Println("【页面】index")
   index_byte, err := ioutil.ReadFile("index.html")
 // 读取 index.html 这个文件，读取后的格式是 byte 数组
   if err != nil {
    _, _ = fmt.Fprintf(w, "服务器读取文件异常")
   } else {
    index_page := string(index_byte)
// 把 html 文件的 byte 数组转为字符串（也就是正常的代码）
    _, _ = fmt.Fprintf(w, index_page)
 // 把 html 文件的字符串格式返回给前端，Fprintf(target, data) 函数即将 data 输入到 target 中
   }
}
```

Index.html 文件

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>计算器</title>
    <script src="/js/jquery-3.4.1.min.js"></script>
    <style>
        main{
            width: 300px;
            margin: 0 auto;
            margin-top: 100px;
            position: relative;
        }
        .title{
            text-align: center;
            font-size: 12px;
            color: #666666;
        }
        .content{
            border: 1px solid #666666;
            width: 100%;
        }
        .console{
            padding: 30px 6px 0 5px;
            height: 35px;
            font-size: 30px;
            line-height: 35px;
            font-weight: bold;
            text-align: right;
            cursor: pointer;
            overflow: hidden;
        }
        .button-table{
            width: 100%;
            text-align: center;
            user-select: none;
        }
        .button-table td{
            border: 1px solid rgba(0, 0, 0, 0);
            background: rgba(40, 40, 40, 0.1);
            width: 74px;
            height: 45px;
            cursor: pointer;
            font-size: 20px;
```

```
        }
        .button-table td:hover{
            background: rgba(40, 40, 40, 0.5);
            color: white;
        }
        .tip{
            position: absolute;
            width: 99%;
            height: 20px;
            top: 25px;
            right: 0.5px;
            font-size: 12px;
            color: red;
            text-align: right;
        }
    </style>
</head>
<body>
    <main>
        <div class="tip" id="tip"></div>
        <div class="title">简单计算器</div>
        <div class="content">
            <div class="console" id="console"></div>
            <table class="button-table">
                <tbody>
                <tr><td
class="func">sin</td><td
class="func">cos</td><td
class="func">sqrt</td><td class="delete">&larr;</td></tr>
                <tr><td
class="character number">7</td><td
class="character number">8</td><td
class="character number">9</td><td
class="character operate">+</td></tr>
                <tr><td
class="character number">4</td><td
class="character number">5</td><td
class="character number">6</td><td
class="character operate">-</td></tr>
                <tr><td
class="character number">3</td><td
class="character number">2</td><td
class="character number">1</td><td
class="character operate">*</td></tr>
```

```
                <tr><td
class="character number">.</td><td
class="character number">0</td><td id="submit-btm">=</td ><td
class="character operate">/</td></tr>
                </tbody>
            </table>
        </div>
    </main>
</body>
<script>
    var MsgTable = new Map();
    var getValued = false;
    MsgTable.set("1000", "表达式错误！");

    var protocol = "http";
    var hostAddress = "localhost";
    var port = "8080";
    var getValuePath = "result";
    var getValueURL = protocol + "://" + hostAddress + ":" + port + "/" + getValuePath;
// 拼接后的结果就是 http://localhost:8080/result

    $(".number").click(function () {
        if (getValued) { // 如果 console 上已经获得一次结果了，清空它
            $("#console").html("");
            $("#tip").html("");
            getValued = false;
        }
    });

    $(".operate").click(function () {
        if (getValued) {  // 如果 console 上已经获得一次结果了，且继续追加表达式
            getValued = false;
        }
    });

    /*
    * 为所有的字符（数字、操作符）绑定点击事件。每次点击，将当前点击字符追加到表达式中。
    */
    $(".character").click(function () {
        let consoleDiv = $("#console");
        let expression = consoleDiv.html();          // 获取当前的表达式
```

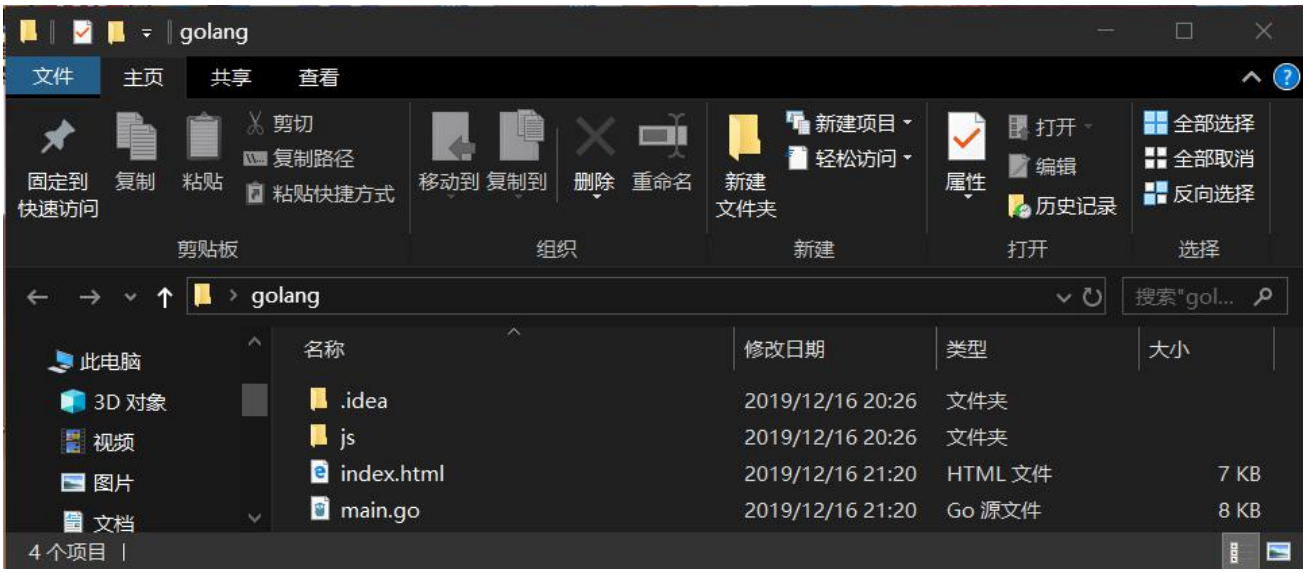
```
        let clickedChar = $(this).html();            // 获取当前点击的字符
        consoleDiv.html(expression + clickedChar);   // 将拼接后的新表达式刷新到展示框中
        setSmallerFont();                            // 调整展示框中字体大小
    });

    /*
    * 为4个函数按钮绑定点击事件。和为字符绑定点击事件在逻辑上差不多，不同之处在于要把函数名字加在表达式前面。
    */
    $(".func").click(function () {
        let consoleDiv = $("#console");
        let expression = consoleDiv.html();
        let clickedChar = $(this).html();
        consoleDiv.html(clickedChar + "(" + expression + ")");
        setSmallerFont();
    });

    /*
    * 为删除按钮绑定点击事件，删除表达式最后一个符号。如果是右括号的话同时删除前面的函数名
    */
    $(".delete").click(function () {
        let consoleDiv = $("#console");
        let expression = consoleDiv.html();
        let exprLen = expression.length;
        let lastChar = expression.charAt(exprLen-1);

        let beginSubIndex = -1;
        if (lastChar === ")"){
            beginSubIndex = expression.indexOf("(");
        }
        consoleDiv.html(expression.substring(beginSubIndex+1, exprLen-1));
        // setGigFont();
    });

    /*
    * 为等号按钮绑定点击事件。点击时，将当前表达式上传到服务器，并根据服务器返回的MsgCode，选择展示计算结果，还是展示错误信息（表达式错误）。
    */
    $("#submit-btm").click(function () {
        let expression = $("#console").html();
        console.log(getValueURL+"?expression="+expression);
```

```
    // 发送 ajax 请求
    $.ajax({
        url: getValueURL,
        method: "GET",
        dataType: "json",
        data: {
            "expression": expression          // 这个要和后端获取数据时写的一样
        },
        success: function (data) {
            let MsgCode = data["MsgCode"];
            let exprsResult = data["result"];


            if (MsgCode === "1000"){            // 返回值代码为 1001 代表表达式错误
                $("#tip").html(MsgTable.get(MsgCode));
                return null;
            }
            // 成功的情况
            $("#console").html(exprsResult);
            $("#tip").html("");
            getValued = true;
        }
    })
});

function setSmallerFont() {
    let consoleDiv = $("#console");
    let expression = consoleDiv.html();
    let maxFontSize = 30;
    let rangeLen = (expression.length>16)? (expression.length-16)/1.3: 0;
    let newFontSize = Math.round(maxFontSize - rangeLen);
    newFontSize = (newFontSize<=12)? 12: newFontSize;
    consoleDiv.css("font-size", ""+newFontSize+"px");
}
/*
function setGigFont() {
    let consoleDiv = $("#console");
    let expression = consoleDiv.html();
    let maxFontSize = 30;
    let rangeLen = (expression.length>16)? (expression.length-16)/1.3: 0;
    let newFontSize = Math.round(maxFontSize - rangeLen);
    newFontSize = (newFontSize<=12)? 12: newFontSize;
```

```
        consoleDiv.css("font-size", ""+newFontSize+"px");
    }
    */
</script>
</html>
```

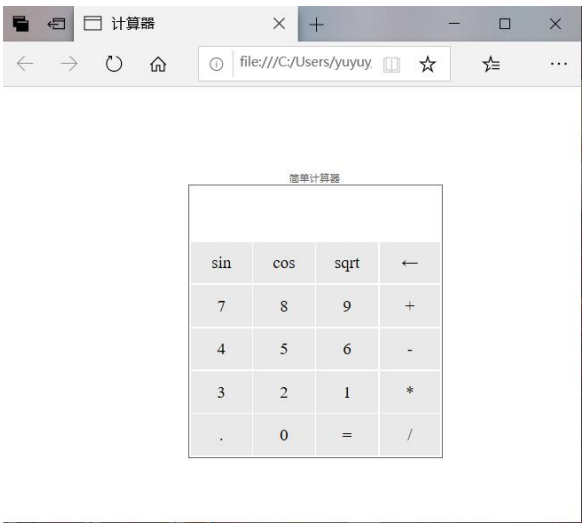
注：使用了前端库 Jquery。

三、实验结果：



打开 index.html 文件:

计算器如图：



为了能够运行，先在后端打开 main.go:

文件目录下打开 CMD，go run main.go

然后在浏览器中输入 localhost:8080/index，回车，便可以计算：

计算结果：

			33+66
sin	cos	sqrt	←
7	8	9	+
4	5	6	-
3	2	1	*
.	0	=	/

			99.00
sin	cos	sqrt	←
7	8	9	+
4	5	6	-
3	2	1	*
.	0	=	/

			99.00-60
sin	cos	sqrt	←
7	8	9	+
4	5	6	-
3	2	1	*
.	0	=	/

			39.00
sin	cos	sqrt	←
7	8	9	+
4	5	6	-
3	2	1	*
.	0	=	/

			39.00*10
sin	cos	sqrt	←
7	8	9	+
4	5	6	-
3	2	1	*
.	0	=	/

			390.00
sin	cos	sqrt	←
7	8	9	+
4	5	6	-
3	2	1	*
.	0	=	/

			390.00/5
sin	cos	sqrt	←
7	8	9	+
4	5	6	-
3	2	1	*
.	0	=	/

			78.00
sin	cos	sqrt	←
7	8	9	+
4	5	6	-
3	2	1	*
.	0	=	/

sin1=0.8414709848079

cos1=0.5403023058681

			sin(1)
sin	cos	sqrt	←
7	8	9	+
4	5	6	-
3	2	1	*
.	0	=	/

			0.84
sin	cos	sqrt	←
7	8	9	+
4	5	6	-
3	2	1	*
.	0	=	/

			cos(1)
sin	cos	sqrt	←
7	8	9	+
4	5	6	-
3	2	1	*
.	0	=	/

			0.54
sin	cos	sqrt	←
7	8	9	+
4	5	6	-
3	2	1	*
.	0	=	/

sqrt（16）=4

			sqrt(16)
sin	cos	sqrt	←
7	8	9	+
4	5	6	-
3	2	1	*
.	0	=	/

			4.00
sin	cos	sqrt	←
7	8	9	+
4	5	6	-
3	2	1	*
.	0	=	/

以上实验结果经检验全部正确。

简单计算器

sin	cos	sqrt	←
7	8	9	+
4	5	6	-
3	2	1	*
.	0	=	/

四、

五、