我们先从最最最最最基本的一个上传下载写起来，一步步加功能给大家讲

不然确实我就算一行一行给你们写注释你也不知道看的顺序

放心，会更完

STEP1

首先需要一个main函数作为服务器的入口

```
func main() {
    http.HandleFunc( pattern: "/objects/", objects.Handler)        //处理以"/object/"开头的URL，那么就交给objects.Handler来具体实现
    log.Fatal(http.ListenAndServe(os.Getenv( key: "LISTEN_ADDRESS"), handler: nil))        //就是记录日志 这个不是重点 但服务器必须要记录操作日志
}
```

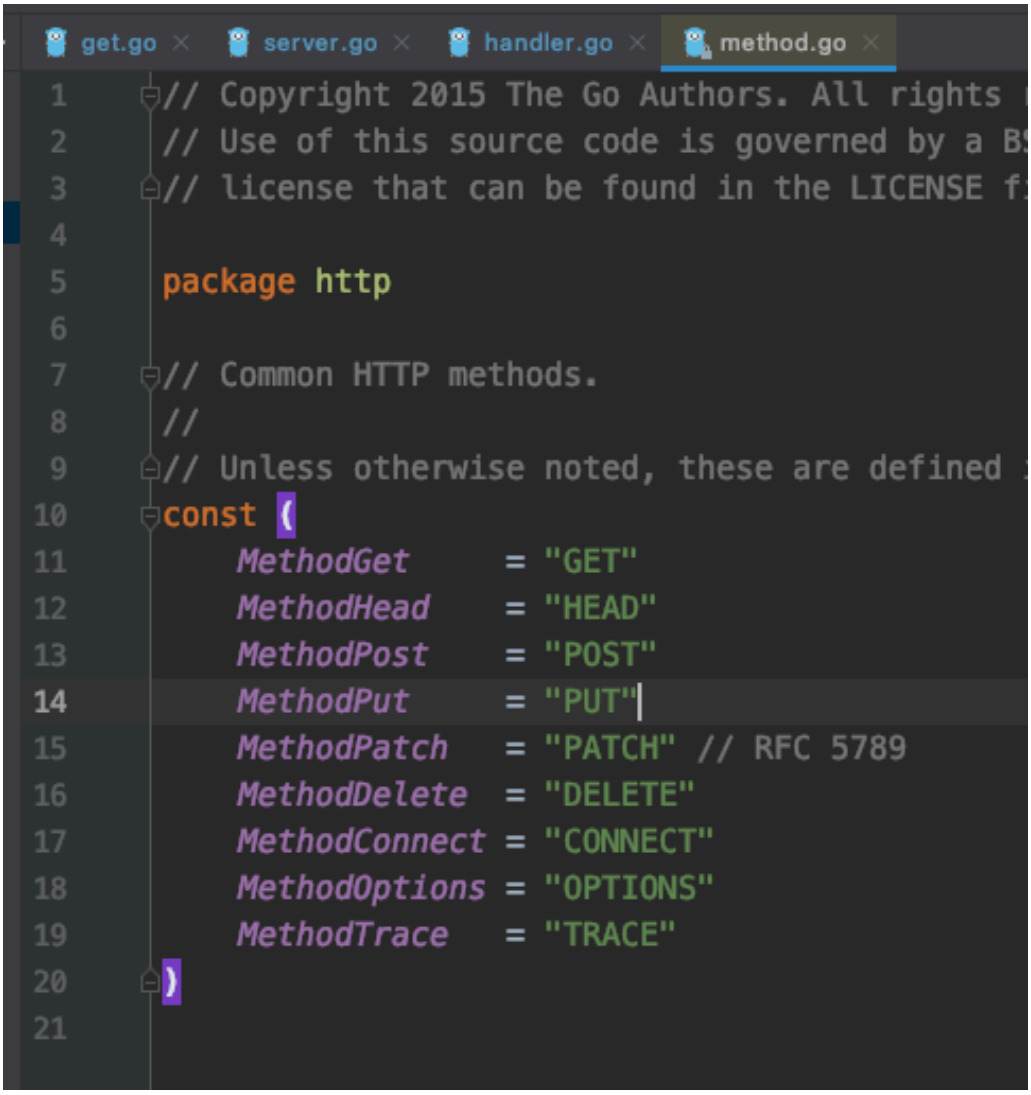
这个能看懂吧，run一下服务器就在运行监听，等着有URL是以这个开头的，就抓住，给后面的函数执行

那我们看看object包下面的Handler他handle了啥

```
/**
        ResponseWriter, Request我就不解释了
*/
func Handler(w http.ResponseWriter, r *http.Request) {
    m := r.Method
    if m == http.MethodPut {
        put(w, r)
        return
    }
    if m == http.MethodGet {
        get(w, r)
        return
    }
    w.WriteHeader(http.StatusMethodNotAllowed)
}
```

这request和response我就不解释了，所有语言的网络处理都是处理这俩东西，大家可以自己点进去看看有什么函数。

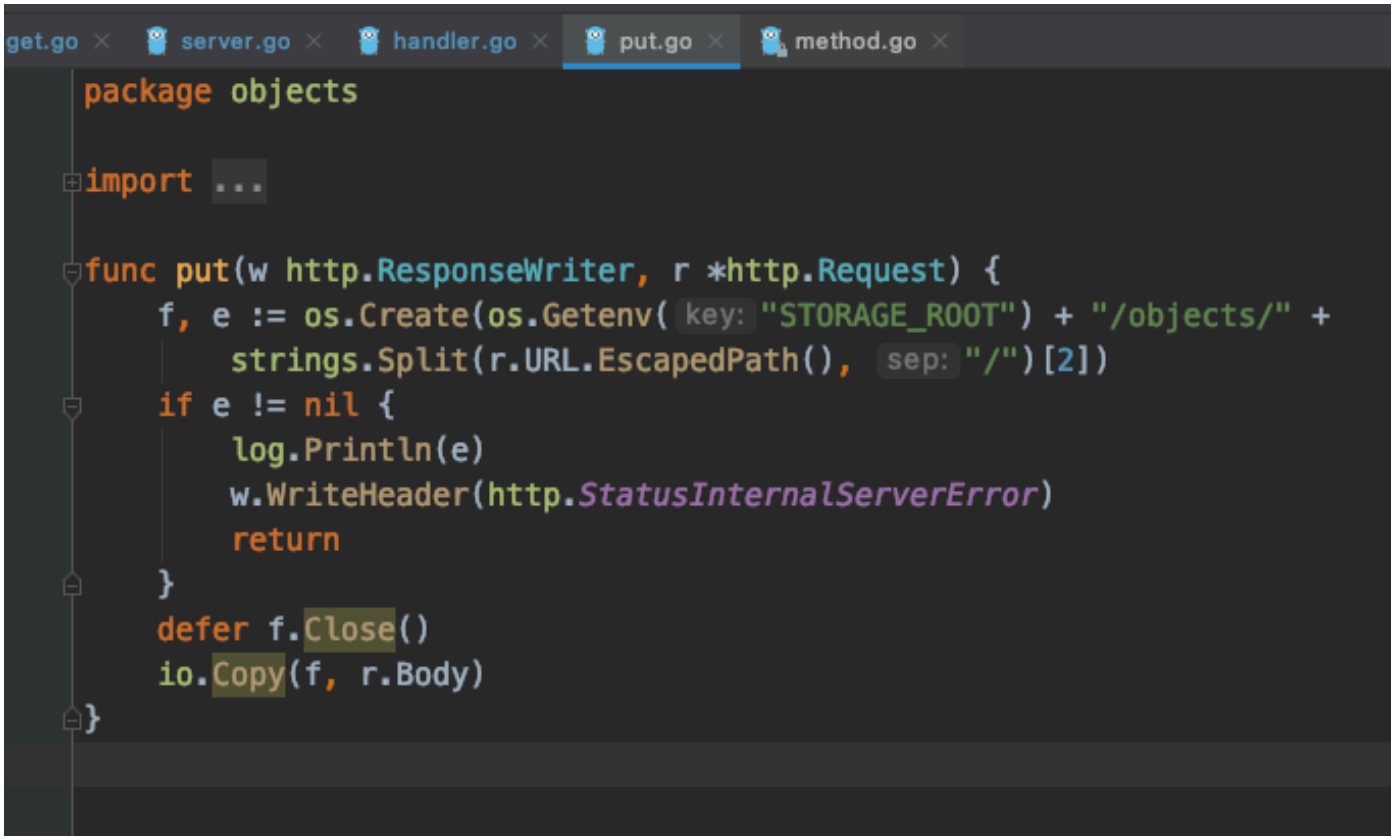
首先提取出方法，看看由server截获来的这个HTTP请求是请求我们做什么方法？PUT还是GET？看不懂MethodPut在哪定义的就直接control（Mac是command）然后点一下就点进去了



Go框架自己写好的对吧，能理解了吧

所以handler的操作总结起来就是，看他要我做什么操作，我就调用写好的PUT和GET函数去具体执行

那我们就看一下put和get具体咋写的



看着复杂，其实很简单，就是一个拼接字符串操作，首先从URL中获取要存的obeject的name，然后前面加上之前设定好的STORGAE_ROOT根目录对吧，再建一层储存文件夹，这个你可以自己设置没所谓，后面能找到就行

通过这个路径打卡一个文件流对吧，然后写入r.body，对吧你要存的内容

还有就是处理错误，go语言写100行有50行在处理error

别忘了defer把文件流关了

就是这么简单

get函数呢？更简单，像put一样，拼接好要找的东西路径，然后去找，找不到了就报错，结构跟put可以说一模一样

get.go | server.go | handler.go | put.go | method.go

```go
package objects

import (
    "io"
    "log"
    "net/http"
    "os"
    "strings"
)



func get(w http.ResponseWriter, r *http.Request) {
    f, e := os.Open(os.Getenv( key: "STORAGE_ROOT") + "/objects/" +
        strings.Split(r.URL.EscapedPath(), sep: "/")[2])
    if e != nil {
        log.Println(e)
        w.WriteHeader(http.StatusNotFound)
        return
    }
    defer f.Close()
    io.Copy(w, f)
}

```

这就是一个最基本，的上传下载服务器

有人说存的东西是啥，存啥都一样，都是字节，你要存图片，也是读取字节分片存（后面更新分片的操作），存电影也一样，万物皆文件，计算机里都是字节

这套最初的代码我会叫step1发给你们

接下来看分布式结构，

STEP2

架构图给你们发过了，直接讲代码

这里会多一层，就分为了api层和data层

api做逻辑的对吧，业务层

data做实际的存储，数据层

data层

先看服务器有啥变化

为了管理存储节点，需要心跳机制来确认存活或者down掉对吧

为了在存取之前确认你这个东西我到底该去哪个节点找，加一个locate的过程，毕竟如果百万节点，我不能便利挨个服务器都找一下吧

这里就用到了goroutine，如果不用，那顺序执行我们心跳就发一次？显然不行，这得持续监控发送心跳信号对吧，能理解吧。Locate操作也一样，开个协程在那等着就行

我们先看心跳信号是如何写的哈 data/heartbeat/heatbeat.go

其实说白了心跳信号很简单，就是我一直给你周期发个消息，你还活着你就回个话，你不回我当你死了，就是这样

这里我们用mq去新建了一个rabbitmq.RabbitMQ结构体，理解成实例化了一个对象吧就行不必钻牛角尖

heartbeat.go

```go
package heartbeat

import (
    "../../../src/lib/rabbitmq"
    "os"
    "time"
)

func StartHeartbeat() {
    q := rabbitmq.New(os.Getenv( key: "RABBITMQ_SERVER"))
    defer q.Close()
    for {
        q.Publish( exchange: "apiServers", os.Getenv( key: "LISTEN_ADDRESS"))
        time.Sleep(5 * time.Second)
    }
}

```

pubilish推送我们要监听的地址

New和Publish都是封装好mq的方法，点进去就可以看到，但再具体的mq的API建议大家亲自动手查一查文档好吧

然后一个死循环包起来，每五秒一次对吧

这个在一个协程里执行的，死循环不耽误别的执行

理解了吗

看看data下面locate包干啥用的

```go
package locate

import (
    "../../../src/lib/rabbitmq"
    "os"
    "strconv"
)

func Locate(name string) bool {
    _, err := os.Stat(name)
    return !os.IsNotExist(err)
}

func StartLocate() {
    q := rabbitmq.New(os.Getenv( key: "RABBITMQ_SERVER"))
    defer q.Close()
    q.Bind( exchange: "dataServers")
    c := q.Consume()
    for msg := range c {
        object, e := strconv.Unquote(string(msg.Body))
        if e != nil : e *
        if Locate(os.Getenv( key: "STORAGE_ROOT") + "/objects/" + object) {
            q.Send(msg.ReplyTo, os.Getenv( key: "LISTEN_ADDRESS"))
        }
    }
}
```

首先一个小的Locate函数，用来判断找的东西在不在

主体是Start Locate函数

还是先实例一个q

绑定dataserver exchange

Consume方法自己点进去看，都是封装好的，看不懂的话记住返回了一个go特有的channel

```
func (q *RabbitMQ) Consume() <-chan amqp.Delivery {
    c, e := q.channel.Consume(q.Name,
        "",
        true,
        false,
        false,
        false,
        nil,
    )
    if e != nil { e *
    return c
}
```

通过便利这个channel我们可以接受消息

```
for msg := range c {
    object, e := strconv.Unquote(string(msg.Body))
    if e != nil { e *
    if Locate(os.Getenv( key: "STORAGE_ROOT") + "/objects/" + object) {
        q.Send(msg.ReplyTo, os.Getenv( key: "LISTEN_ADDRESS"))
    }
}
```

```
object, e := strconv.Unquote(string(msg.Body))
```

这句是处理双引号，以后这种东西我就不单独说了，希望大家这种小事就查一下即可，我主要给大家讲设计讲思想讲流程

我教大家如何看代码哈，比如这个ReplyTo是不是红了，那你想知道这是啥你怎么看，他是红的点不了，那就往前点msg，发现是到了这行

```
    for msg := range c {
        object, e := strconv.Unquote(string(msg.Body))
        if e != nil { e *
        if Locate(os.Getenv( key: "STORAGE_ROOT") + "/objects/" + object) {
            q.Send(msg.ReplyTo, os.Getenv( key: "LISTEN_ADDRESS"))
        }
    }
```

原来是从c里面拿出来的，一堆msg组成了c，对吧

那我们就看c是啥，command点进去发现哦是q的一个consume方法返回的

```
c := q.Consume()
for msg := range c {
    object, e := strconv.Unquote(str
```

继续点进去

```
func (q *RabbitMQ) Consume() <-chan amqp.Delivery {
    c, e := q.channel.Consume(q.Name,
        "",
        true,
        false,
        false,
        false,
        nil,
    )
    if e != nil : e *
    return c
}
```

原来c的类型是amqp.Delivery 那么问题解决了，是import没引全，RepyTo显然是Delivery的成员对吧

但其实真的没必要这么扣深的，你扣下去会研究mq源码的，没必要，还是那句话，先理解设计思想，最后有空再自己扣这种（这种细节对于面试没用，大家聊的都是设计架构思想）

说回来Start Locate干了什么事?

就是挡在data层前面，你取之前我先查查这东西存在哪了对吧，不然你也不知道去哪个机器上取，如果没查到那就别尝试取了，浪费资源

接口层

```
func main() {
    go heartbeat.ListenHeartbeat()
    http.HandleFunc( pattern: "/objects/", objects.Handler)
    http.HandleFunc( pattern: "/locate/", locate.Handler)
    log.Fatal(http.ListenAndServe(os.Getenv( key: "LISTEN_ADDRESS"), handler: nil))
}
```

接口层服务器也很明显做的事情，心跳、存取、定位、日志

但一定注意！！！

函数名字一样，内容不一样！！！！

接口层和data层都有object包对吧，但接口层只负责转发，data层的才负责操作磁盘

locate同理

```go
package heartbeat

import ...

var dataServers = make(map[string]time.Time)
var mutex sync.Mutex

func ListenHeartbeat() {
    q := rabbitmq.New(os.Getenv( key: "RABBITMQ_SERVER"))
    defer q.Close()
    q.Bind("apiServers")
    c := q.Consume()
    go removeExpiredDataServer()
    for msg := range c {
        dataServer, e := strconv.Unquote(string(msg.Body))
        if e != nil : e *
        mutex.Lock()
        dataServers[dataServer] = time.Now()
        mutex.Unlock()
    }
}

func removeExpiredDataServer() {
    for {
        time.Sleep(5 * time.Second)
        mutex.Lock()
        for s, t := range dataServers {
            if t.Add(10 * time.Second).Before(time.Now()) {
                delete(dataServers, s)
            }
        }
        mutex.Unlock()
    }
}

func GetDataServers() []string {
    mutex.Lock()
    defer mutex.Unlock()
    ds := make([]string, 0)
```

接口层的心跳包我就不赘述了，一开始的注释版本就非常的详细了，其实看函数名字也很好理解，管理节点嘛，节点增删改查呗

我想在这里讲解一下流的概念

NewPutStream函数是生成PutStream结构体的

使用io.Pipe创建了一对reader和writer，类型分别是*io.PipeReader和 *io.PipeWriter。这是一个管道，类似于channel这边写进去那边读出来。

具体的有关函数我在下一次更新里讲，并且下一次争取再更新两个功能的讲解

大家体谅下，我在尽快写，但我白天还有别的工作，加上写6.824很肝。

项目方面主要理解设计思想，像流这种工程实现其实可以先往后放一放，这种东西入职了看看别人写的照着写几次自然就会了，设计思考的思想才是最主要的。

2022.3.19