

Go API编程

基于beego打造高效API之路

Apr 25, 2013

谢孟军

高级研究员，盛大云

Agenda

- 什么是API
- 为什么使用API开发
- API开发中几个关键点
- Go适合API开发
- API框架beego的介绍
- beego如何开发项目
- 实际项目的经验和教训

什么是API

API———Application programming interface

Timelines

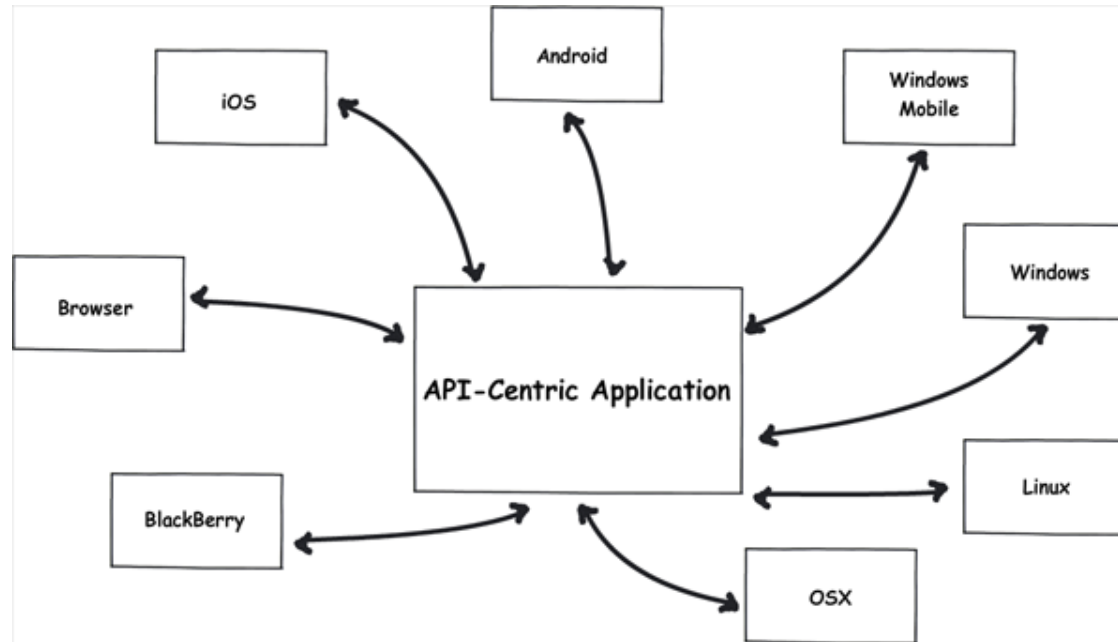
Timelines are collections of Tweets, ordered with the most recent first.

Resource	Description
GET statuses/mentions_timeline	Returns the 20 most recent mentions (tweets mentioning a specific user) for the user specified by the user_id parameter. The timeline returned is the equivalent of the user's "mentions" tab. This method can only return up to 800 tweets.
GET statuses/user_timeline	Returns a collection of the most recent Tweets posted by the user specified by the user_id parameter. User timelines belonging to an authenticated user either "owns" the timeline or is a user that the authenticated user follows. This method can only return up to 800 tweets.
GET statuses/home_timeline	Returns a collection of the most recent Tweets posted by the users that the user specified by the user_id parameter follows. The home timeline is centered on the user specified by the user_id parameter. This method can only return up to 800 Tweets.
GET statuses/retweets_of_me	Returns the most recent tweets authored by users that the user specified by the user_id parameter follows. This timeline is a subset of the user's GET statuses/user_timeline. This method can only return up to 800 tweets. See the instructions on traversing timelines.

谁在使用API

- Google
- Facebook
- Twitter
- GitHub
- Amazon
- 微博
- 淘宝
- 腾讯
- 百度

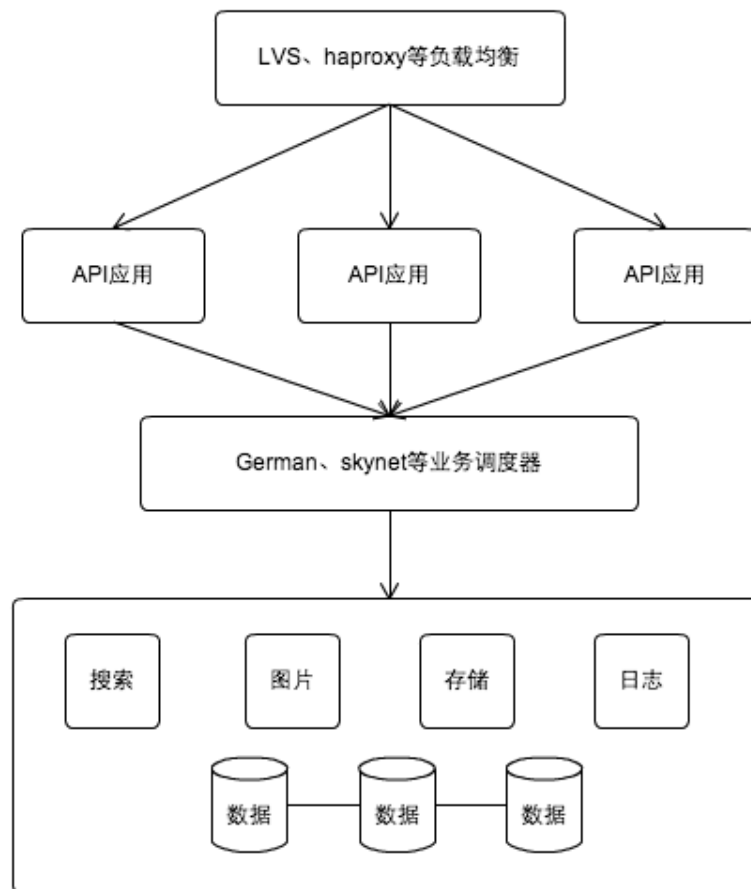
API为中心的应用设计



API-Centric的好处

- 服务多个终端，特别是现在移动互联网的快速发展
- 标准化，无论是对外还是对内都能实现统一标准(http+json/xml)
- 适合团队之间的协作
- 代码清晰，结构化，可维护性强
- 安全性，可以利用认证系统做到API的安全
- 去中心化，易扩展，API的无状态化，可以很方便的进行扩展
- 实现模块化，实现最小化的API之后，可以通过组合又可以拼装新的API
- 可以快速将内部的API进行开放，让应用迅速的得到开发者的支持

API的典型部署方案



API的关键点

高富帅要求：

- 运行快速
- 开发高效
- 运行稳定

Go语言特别适合API编程，它在这三个特点中找到了平衡。

@CodeBox-腾讯：

为了开发效率与运行效率之间的平衡，facebook使用php+HipHop，google使用python+psyco，冥冥之中感觉，最后可能都会被开发效率和运行效率都不错的golang统一了。

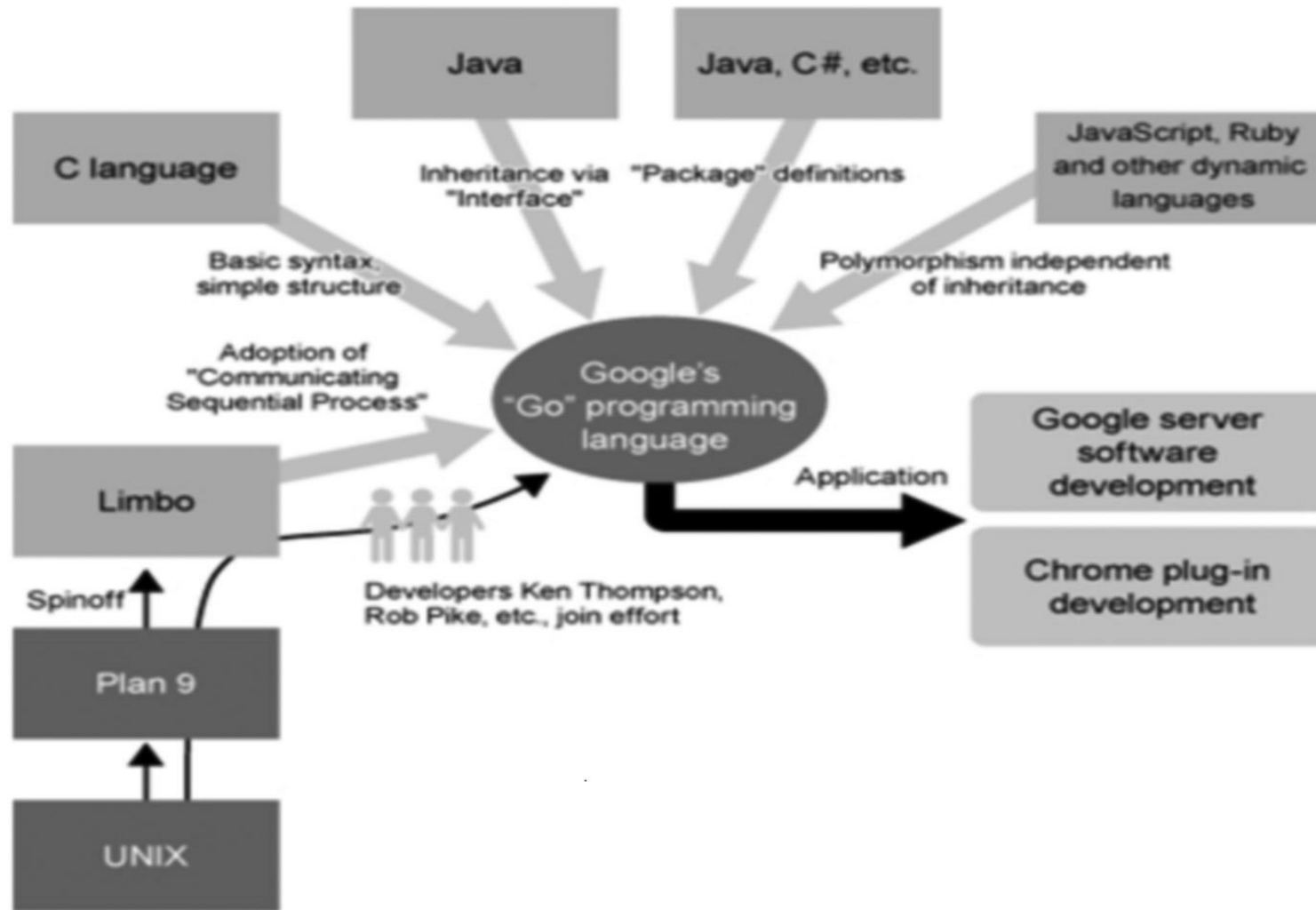
什么是Go?

Go出自名门Google公司，是一门并发支持、垃圾回收的编译型系统编程语言，是一门具有在静态编译语言的高性能和动态语言的高效开发之间拥有良好平衡点的编程语言。

该项目的三位领导者均是著名的 IT 工程师：

- Rob Pike，Go 语言项目总负责人，贝尔实验室 Unix 团队成员，参与的项目包括 Plan 9，Inferno 操作系统和 Limbo 编程语言；
- Ken Thompson，贝尔实验室 Unix 团队成员，C 语言、Unix 和 Plan 9 的创始人之一，与 Rob Pike 共同开发了 UTF-8 字符集规范，图灵奖获得者。
- Robert Griesemer，参与开发 Java HotSpot 虚拟机；

Go的语法特性



Go的主要特点

- 语法简单，和C类似，入门很快
- 类型安全和内存安全
- 以非常直观和极低代价的方案实现高并发
- 高效的垃圾回收机制
- 快速编译（同时解决C语言中头文件太多的问题）
- 为多核计算机提供性能提升的方案
- UTF-8编码支持
- 跨平台编译
- 集成工具丰富，特别是gofmt、godoc

对于Go的担心

- 新语言是不是不够稳定?
- 很多包都是新的，是不是存在很多bug?
- GC和Scheduler是不是不够成熟?
- Google扼杀了很多有用的项目，就如最近的Google Reader, Go语言会是下一个吗?

Who are use

- Google
- Youtube
- BBC Worldwide
- HeroKu
- dotCloud
- CloudFlare
- SoundCloud
- Mozilla Services
- Apcera
- 七牛
- 盛大
- 360

Go的例子之入门

```
package main

import (
    "fmt"
)

const (
    HELLO = "hello"
    WORLD = "world"
)

var (
    name string
    i     int
    j     string
)

func main() {
    name = "astaxie"
    i = 100
    j = "qcon"
    fmt.Println(HELLO, WORLD, name, i, j)
}
```

[Run](#)

Go的例子之函数

```
package main

import (
    "fmt"
)

func delay(a string) {
    fd,err:=os.Open("a.txt")
    if err!=nil{
        return
    }
    defer fd.Close()
    defer fmt.Println("exit the func delay exec")
    fmt.Println("hi", a)
}

func AddMult(a, b int) (sum, mult int) {
    return a + b, a * b
}

func main() {
    a := []int{1, 2}
    for _, v := range a {
        fmt.Println(AddMult(2, v))
        delay("astaxie")
    }
}
```

```
3 2
hi astaxie
exit the func delay exec
0
1
2
3
4
4 4
hi astaxie
exit the func delay exec
0
1
```

Run Kill Close

Go的例子之网络编程

```
package main

import (
    "fmt"
    "net/http"
)

func main() {
    http.HandleFunc("/", hello)
    http.ListenAndServe("localhost:8000", nil)
}

func hello(w http.ResponseWriter, r *http.Request) {
    fmt.Fprintln(w, "Hello, Gophers! Welcome to Qcon!")
}
```

[Run](#)

Go的例子之并发

```
package main

import (
    "fmt"; "net/http"; "time"
)

func main() {
    urls := []string{"http://www.google.com.hk/", "http://cn.bing.com/"}
    start := time.Now()
    done := make(chan string)
    for _, u := range urls {
        go func(u string) {
            resp, err := http.Get(u)
            if err != nil {
                done <- u + " " + err.Error()
            } else {
                done <- u + " " + resp.Status
            }
        }(u)
    }
    for _ = range urls {
        fmt.Println(<-done, time.Since(start))
    }
}
```

[Run](#)

Go的例子之json

```
package main

import (
    "encoding/json"; "fmt"; "os"
)

type ColorGroup struct {
    ID      int
    name    string
    Colors []string
}

func main() {
    group := ColorGroup{
        ID:      1,
        name:    "Reds",
        Colors: []string{"Crimson", "Red", "Ruby", "Maroon"},
    }
    b, err := json.Marshal(group)
    if err != nil {
        fmt.Println("error:", err)
    }
    os.Stdout.Write(b)
}
```

[Run](#)

Go适合API编程

- 静态编译跨平台，很多问题扼杀在编译阶段
- 内置net/http，处理网络应用得心应手
- 内置JSON、XML处理，方便处理表现层
- 语言级别的并发支持，编写并发简单方便
- 丰富的内置包，借鉴了很多Python的思想
- 足够简单就会让我们的程序足够稳定
- 相比PHP、python、Ruby具有性能优势

API开发关注点:

- 并发处理——go并发
- http处理——http内置包
- 数据操作——io/os/strings/strconv/sql包
- XML/JSON处理——json/xml包
- 日志处理——log包
- 用户认证——oauth/oauth2/basic/Digest authentication
- 数据加密—— aes/dec/md5/hash/sha1等包

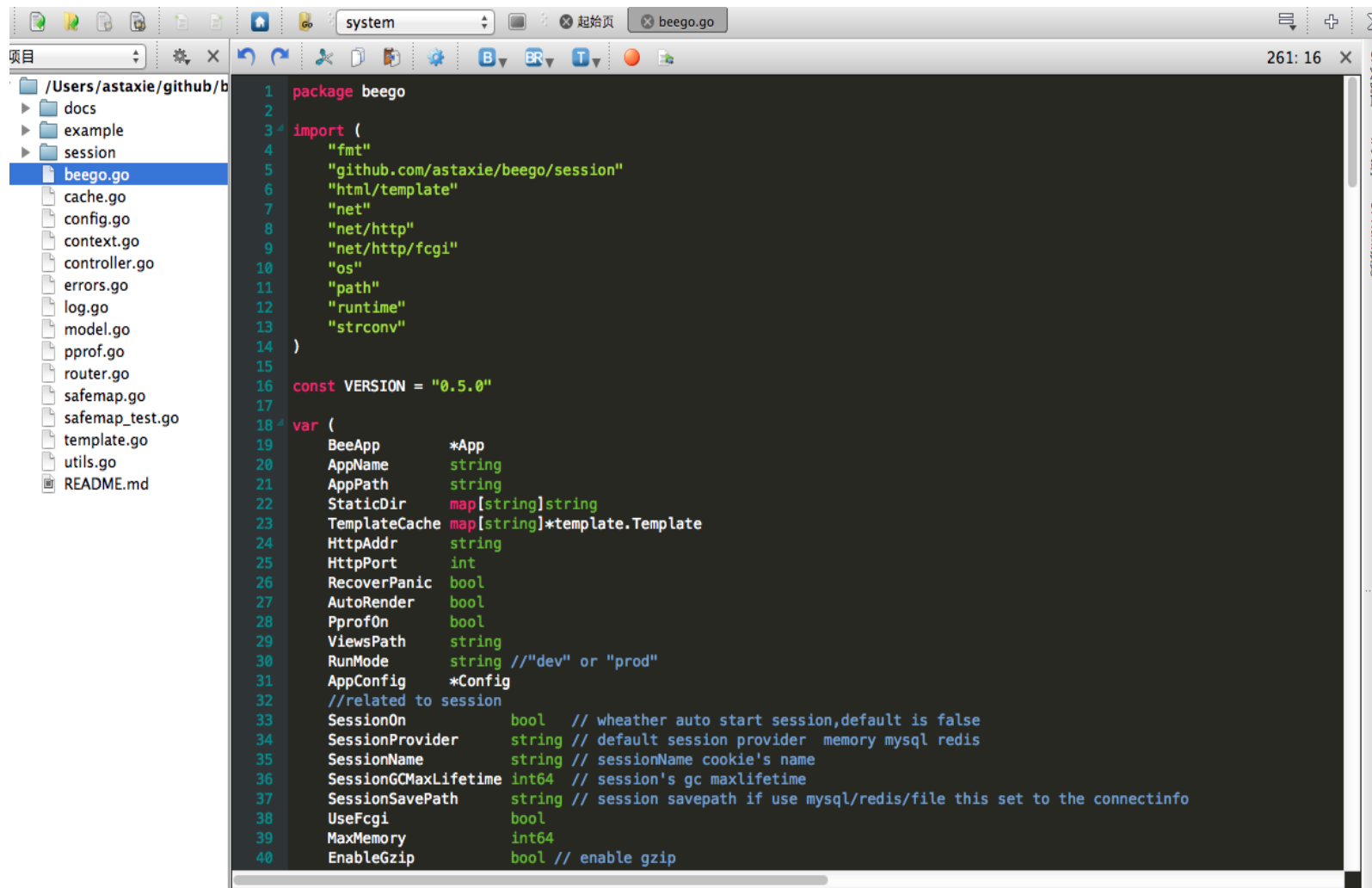
beego是什么

beego是一个基于Go语言开发的应用框架，主要思路来源于tornado和sintra，主要有如下这些特性：

- REST设计
- 加强型路由
- MVC模式
- 智能日志模块
- ini配置模块
- XML、JSON输出
- 常用模板函数
- 缓存系统
- Session模块

github.com/astaxie/beego (<https://github.com/astaxie/beego>)

beego之目录结构



The screenshot shows a code editor window with a file explorer on the left and a code editor on the right. The file explorer shows the directory structure of the beego project, with the following files and folders listed:

- docs
- example
- session
- beego.go
- cache.go
- config.go
- context.go
- controller.go
- errors.go
- log.go
- model.go
- pprof.go
- router.go
- safemap.go
- safemap_test.go
- template.go
- utils.go
- README.md

The code editor shows the following Go code for the beego package:

```
1 package beego
2
3 import (
4     "fmt"
5     "github.com/astaxie/beego/session"
6     "html/template"
7     "net"
8     "net/http"
9     "net/http/cgi"
10    "os"
11    "path"
12    "runtime"
13    "strconv"
14 )
15
16 const VERSION = "0.5.0"
17
18 var (
19     BeeApp      *App
20     AppName     string
21     AppPath     string
22     StaticDir   map[string]string
23     TemplateCache map[string]*template.Template
24     HttpAddr    string
25     HttpPort    int
26     RecoverPanic bool
27     AutoRender  bool
28     PprofOn     bool
29     ViewsPath   string
30     RunMode     string // "dev" or "prod"
31     AppConfig  *Config
32     //related to session
33     SessionOn bool // wheather auto start session,default is false
34     SessionProvider string // default session provider memory mysql redis
35     SessionName string // sessionName cookie's name
36     SessionGCMaxLifetime int64 // session's gc maxlifetime
37     SessionSavePath string // session savepath if use mysql/redis/file this set to the connectinfo
38     UseFcgi bool
39     MaxMemory int64
40     EnableGzip bool // enable gzip
```

beego之入门

```
package main

import (
    "github.com/astaxie/beego"
)

type MainController struct {
    beego.Controller
}

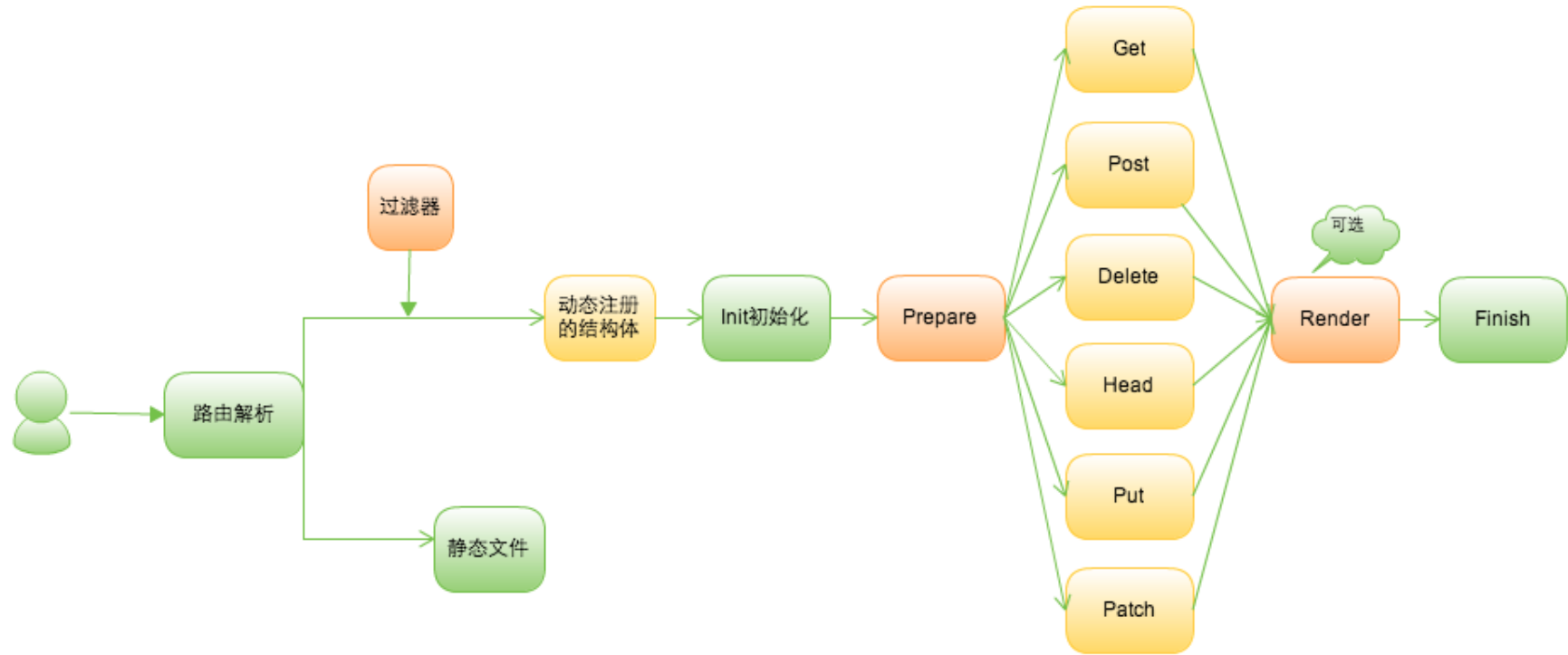
func (this *MainController) Get() {
    this.Ctx.WriteString("hello world")
}

func (this *MainController) Post(){
    this.Ctx.WriteString("hi post")
}

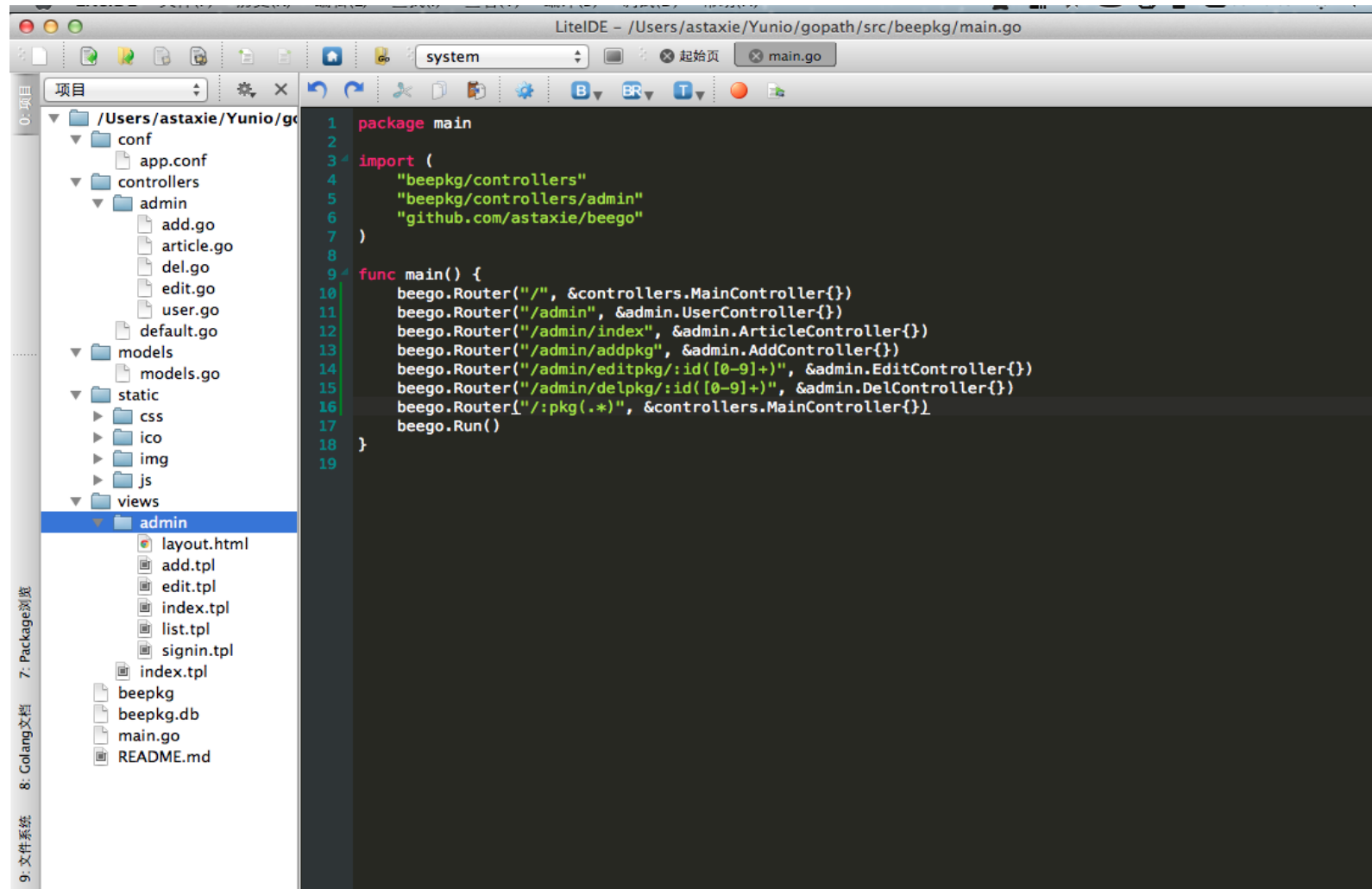
func main() {
    beego.Router("/", &MainController{})
    beego.Run()
}
```

[Run](#)

beego执行逻辑



beego项目目录结构



beego之辅助工具

bee工具

```
go get github.com/astaxie/bee
```

```
xiemengjundeMacBook-Pro:~ astaxie$ bee
Bee is a tool for managing beego framework.

Usage:

    bee command [arguments]

The commands are:

    create      create an application base on beego framework
    start       start the app which can hot compile

要有如下这些特性:
Use "bee help [command]" for more information about a command.

Additional help topics:

Use "bee help [topic]" for more information about that topic.
```

beego之API开发

- 1、使用bee工具创建项目
- 2、编写控制器，设计API的对应逻辑
- 3、注册控制器路由
 - 以短域名为例，设计一个应用

```
bee create shorturl
```

- API接口设计

```
shorturl/shorten    长链转短链  
shorturl/expand    短链转长链
```

短域名逻辑之一 shorturl.go

```
package controllers

import (
    "github.com/astaxie/beego"
    "github.com/astaxie/beego/cache"
    "shorturl/models"
)

var (
    urlcache cache.Cache
)

func init() {
    urlcache, _ = cache.NewCache("memory", `{"interval":0}`)
}

type ShortResult struct {
    UrlShort string
    UrlLong  string
}

type ShortController struct {
    beego.Controller
}
```

短域名逻辑之一 shorturl.go

```
func (this *ShortController) Post() {
    var result ShortResult
    longurl := this.Input().Get("longurl")
    beego.Info(longurl)
    result.UrlLong = longurl
    urlmd5 := models.GetMD5(longurl)
    beego.Info(urlmd5)
    if urlcache.IsExist(urlmd5) {
        result.UrlShort = urlcache.Get(urlmd5).(string)
    } else {
        result.UrlShort = models.Generate()
        err := urlcache.Put(urlmd5, result.UrlShort, 0)
        if err != nil {
            beego.Info(err)
        }
        err = urlcache.Put(result.UrlShort, longurl, 0)
        if err != nil {
            beego.Info(err)
        }
    }
    this.Data["json"] = result
    this.ServeJson()
}
```

短域名逻辑之二 expand.go

```
package controllers

import (
    "github.com/astaxie/beego"
)

type ExpandResult struct {
    beego.Controller
}

func (this *ExpandResult) Get() {
    var result ShortResult
    shorturl := this.Input().Get("shorturl")
    result.UrlShort = shorturl
    if urlcache.IsExist(shorturl) {
        result.UrlLong = urlcache.Get(shorturl).(string)
    } else {
        result.UrlLong = ""
    }
    this.Data["json"] = result
    this.ServeJson()
}
```

短域名之逻辑 model.go

```
func Generate() (tiny string) {
    globalnum++
    num := globalnum
    fmt.Println(num)
    alpha := merge(getRange(48, 57), getRange(65, 90))
    alpha = merge(alpha, getRange(97, 122))
    if num < 62 {
        tiny = string(alpha[num])
        return tiny
    } else {
        var runes []rune
        runes = append(runes, alpha[num%62])
        num = num / 62
        for num >= 1 {
            if num < 62 {
                runes = append(runes, alpha[num-1])
            } else {
                runes = append(runes, alpha[num%62])
            }
            num = num / 62
        }
        for i, j := 0, len(runes)-1; i < j; i, j = i+1, j-1 {
            runes[i], runes[j] = runes[j], runes[i]
        }
    }
}
```


注册路由

```
func main() {  
    beego.Router("/", &controllers.MainController{})  
    beego.Router("/v1/shorten", &controllers.ShortController{})  
    beego.Router("/v1/expand", &controllers.ExpandResult{})  
    beego.Run()  
}
```

最后的效果及测试

缩短域名：

```
POST http://localhost:8080/v1/shorten
```

获取长域名：

```
GET http://localhost:8080/v1/expand
```

beego不完善的地方

- 模板
- 测试
- 表单
- 验证
- 热编译

基于beego的实际项目(I)

1、短域名服务

使用beego开发了一个类似bitly的短域名服务，提供盛大内部项目使用，目前一台机器：32G内存、8核、centos64，redis数据库，数据量在1000w多点，运行半年多时间了，至今没有出现过一次down机或者奔溃

2、政府合作项目

使用beego提供系统级别服务，监控集成电路板信号，智能分析nginx配置，提供大数据量的下载和页面访问

3、内部监控系统

目前这个项目还在开发中，主要是利用beego做两个服务，一个是服务器端，收集信息，一个是客户端，收集信息并上报给服务器端，如果和服务端断开，那么本地可以暂存数据，防止数据丢失，同时还支持类似puppet的功能，支持程序自动升级，部署其他程序等功能。

基于beego的实际项目(II)

4、日志分析系统

以前采用hadoop来分析日志的来源和省份信息，发现hadoop分析这么大的数据，性能不是很好，延迟比较大，目前采用自己的一套架构，squid日志每小时分割上报，每小时对日志进行分割，然后进行UV、PV、省份、运营商、浏览器、数据量等的分析，同时把日志进行按域名分割，提供用户原始日志下载

5、下载分发系统

实现了类似BT的块分发机制，但是实现了机房之间的单文件流，机房内的块copy等功能，实现了提供文件下载的智能分发，从文件上传到下发到每一台服务器，以前采用BT的方式，性能和流量控制都不是很好，目前采用新的架构，性能提升十几倍

6、视频直播调度器

这是一个和盛大游戏合作的项目，做的是游戏中类似YY的一个视频直播间的项目，单机承受8w的并发请求转发，根据用户的IP、对应服务服务器的负载、squid的负载、IO等情况实现一个302跳转。

项目中的经验和教训

主要经验是：

- 要能对linux、进程等基础知识有了解，很多时候需要知道底层的运行机制(文件描述符事件)
- 敢于在项目中使用，不用你永远不知道Go到底有多好(稳定性)
- 不要听别人说，要自己去测试，至于好不好，数据说了算
- Go的模板现在不是很好用，还没办法和PHP、python、ruby下的那些模板引擎比

教训

- Go的import包不支持版本，有时候升级容易导致项目不可运行，所以需要自己控制相应的版本信息
- Go的goroutine无法控制数量，导致很多时候其实是在不同的goroutine之间切换，容易失控
- goroutine调试最好的方式还是log+print，gdb不是很好用
- GC延迟有点大，日志系统上过一次当
- pkg下面的图片处理库很多bug，还是使用成熟产品好，调用这些成熟库imagemagick的接口比较靠谱

Question?

Don't be shy



欢迎大家来交流

QQ群: 259316004

论坛: <http://bbs.mygolang.com>

微博: Golang中国



Thank you

Apr 25, 2013

谢孟军

高级研究员，盛大云

<mailto:xiemengjun@gmail.com>

<http://beego.me>

[@asta谢](http://twitter.com/asta%E8%B0%A2) (<http://twitter.com/asta%E8%B0%A2>)