云联壹云

云联壹云 Meetup 16

Part 2：Golang实践

云联壹云核心产品融合云 云联壹云 秉承：简单、开放、融合、智能的产品理念，能够帮助企业实现异构IT基础设施的全面云化、统一管理及成本优化，提高运维效率的同时，降低企业运营成本。

2021-03 创始人/CEO 邱剑

扫码进技术交流群

更多资讯关注我

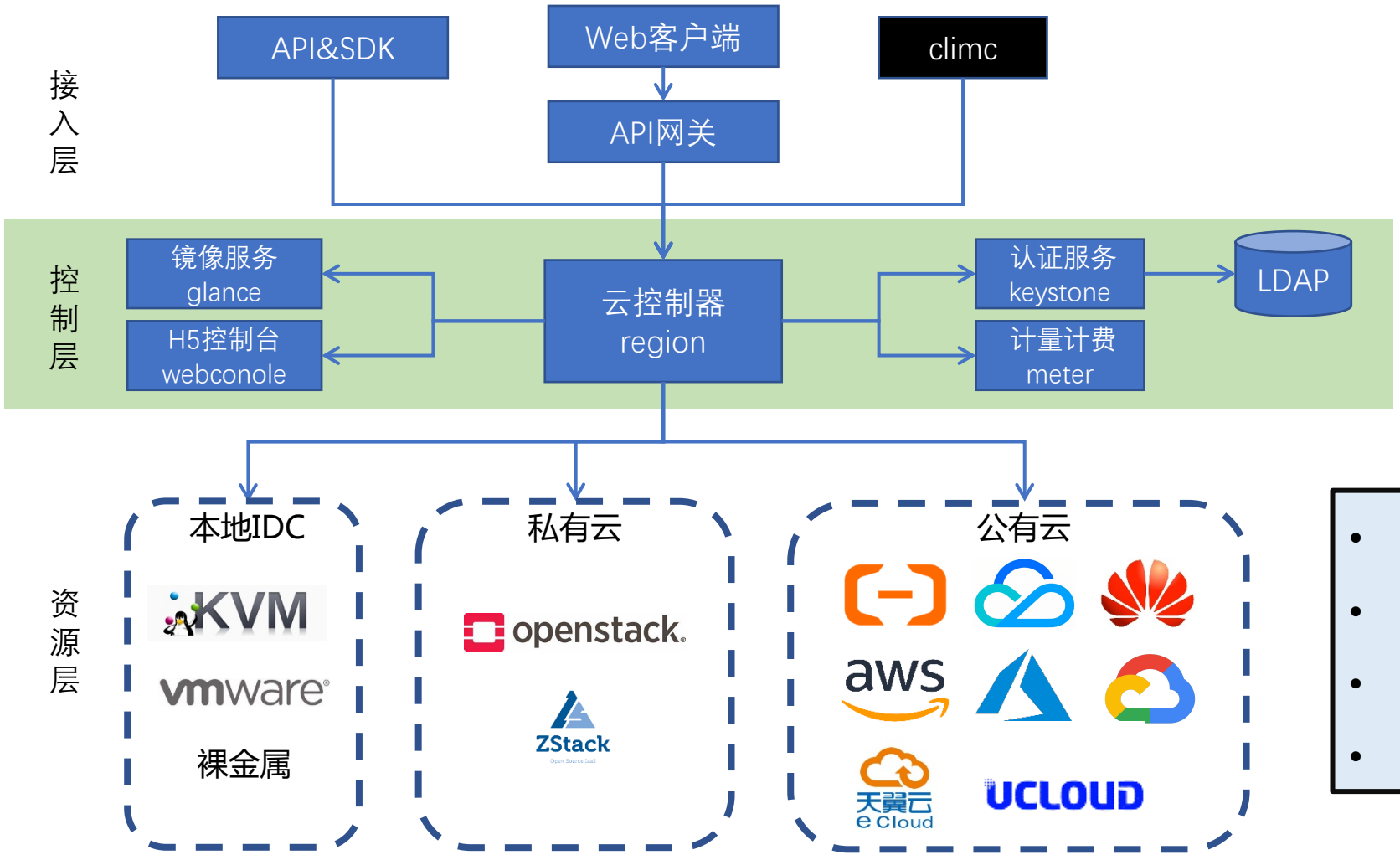
目录

CONTENTS

01

背景介绍

云联壹云融合云平台介绍



- 后端纯golang: ~60万行
- 前端Vue.js
- 认证和权限控制基于Keystone
- 数据库：mysql

云联壹云Golang积累

一个Golang服务框架

四个Golang工具库

- **jsonutils**：JSON序列化和反序列化

  https://github.com/yunionio/jsonutils

- **sqlchemy**：模仿sqlalchemy的ORM库

  https://github.com/yunionio/sqlchemy

- **structarg**：基于结构体的命令行参数生成和解析工具

  https://github.com/yunionio/structarg

- **pkg**：其他一些小工具和方法

  https://github.com/yunionio/pkg

Golang框架

Golang服务框架 – 介绍

- 一个方便的CRUD脚手架
- 内置
  - keystone认证
  - 权限控制
  - 服务配置管理和更新
  - 异步任务管理

Golang服务框架 – 数据模型

- 一类资源对应一张mysql数据表
- 每个资源由一对ModelManager和Model实现
  - ModelManager实现集合资源相关的接口，例如：Create，List
  - Model实现单个资源的接口，例如：Update，Delete，PerformAction

Golang服务框架 – 数据模型同步

- 自动实现Model到数据库表schema的单向同步

```
type SUser struct {
    Id               string            `width:"64" charset:"ascii" nullable:"false" primary:"true"`
    Extra            string            `charset:"utf8" nullable:"true"`
    Enabled          tristate.TriState `nullable:"true"`
    DefaultProjectId string            `width:"64" charset:"ascii" nullable:"true" index:"true"`
    CreatedAt        time.Time         `nullable:"true"`
    LastActiveAt     time.Time         `nullable:"true"`
    DomainId         string            `width:"64" nullable:"false" index:"true"`
}
```

```
+--------------------+-------------+------+-----+---------+-------+
| Field              | Type        | Null | Key | Default | Extra |
+--------------------+-------------+------+-----+---------+-------+
| id                 | varchar(64) | NO   | PRI | NULL    |       |
| extra              | text        | YES  |     | NULL    |       |
| enabled            | tinyint(1)  | YES  |     | NULL    |       |
| default_project_id | varchar(64) | YES  | MUL | NULL    |       |
| created_at         | datetime    | YES  |     | NULL    |       |
| last_active_at     | date        | YES  |     | NULL    |       |
| domain_id          | varchar(64) | NO   | MUL | NULL    |       |
+--------------------+-------------+------+-----+---------+-------+
```

Golang服务框架 – CRUD脚手架

接口功能	METHOD	PATH	实现模型	接口方法	举例
创建	POST	/resources	Manager	ValidateCreateData PostCreate	POST /servers
列表	GET	/resources	Manager	ListItemFilter	GET /servers
获取集合属性	GET	/resources/<property>	Manager	GetPropertyXXXX	GET /servers/sync-tasks
执行集合操作	POST	/resources/<action>	Manager	PerformXXXX	POST /servers/validate-create-data
获取单个资源	GET	/resources/<res_id>	Model	GetDetails	GET /servers/srv1
获取单个资源的特性属性	GET	/resources/<res_id>/<spec>	Model	GetDetailsXXXX	GET /servers/srv1/vnc
更新属性	PUT	/resources/<res_id>	Model	ValidateUpdateData PostUpdate	PUT /servers/srv1
执行特定操作	POST	/resources/<res_id>/<action>	Model	PerformXXXX	POST /servers/srv1/start
删除	DELETE	/resources/<res_id>	Model	ValidateDeleteCondition Delete	DELETE /servers/srv1

Golang库

jsonutils (yunion.io/x/jsonutils)

JSON序列化和反序列的工具库

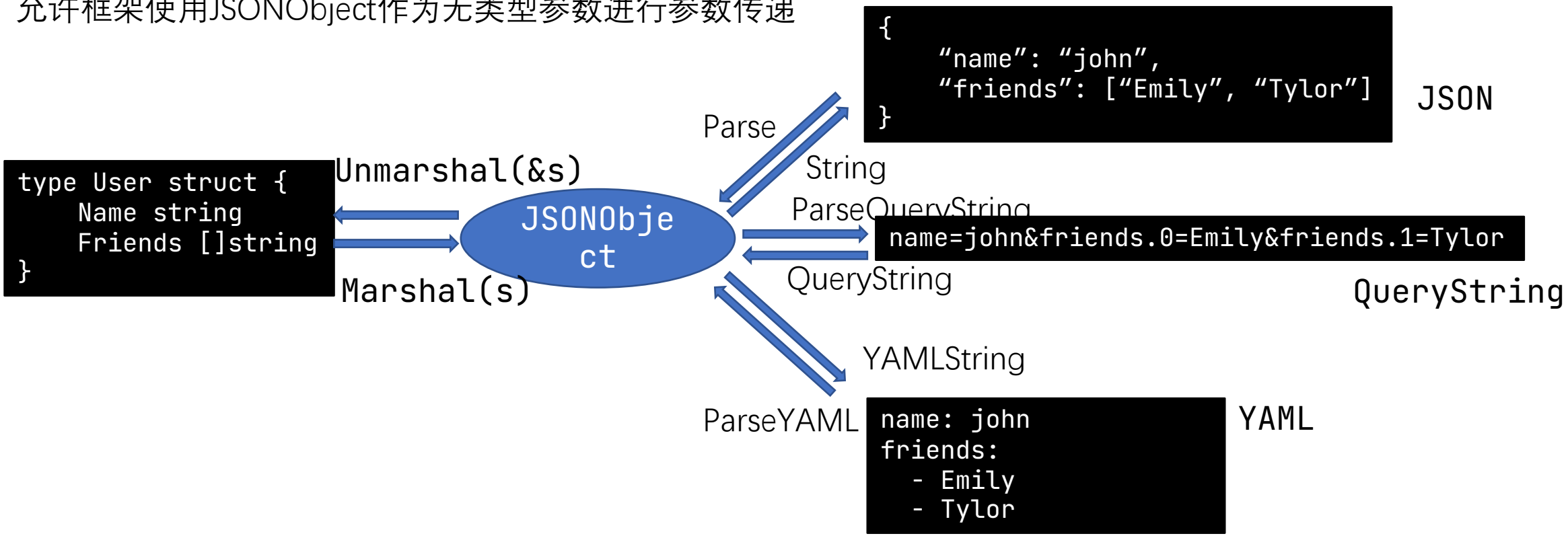
encoding/json

```
type User struct {
    Name string
    Friends []string
}
```

```
{
    “name”: “john”,
    “friends”: [“Emily”, “Tylor”]
}
```

jsonutils

- 实现JSONObject和JSON, QueryString和YAML的相互转换
- 允许API输入输出采用无类型的JSONObject
- 允许框架使用JSONObject作为无类型参数进行参数传递



jsonutils - 处理结构体字段的版本变更

```
type Input struct {
    TenantId  string `json:"tenant_id"`
}
```

输入参数变更

```
type Input struct {
    // Deprecated
    TenantId  string `json:"tenant_id" yunion-deprecated-by:"project_id"`
    ProjectId string `json:"project_id"`
}
```

Unmarshal(&input)

```
{
    "tenant_id": "abcd1234"
}
```

```
Input {
    TenantId:  "abcd1234",
    ProjectId: "abcd1234"
}
```

旧的输入

新的输入参数数据结构

sqlchemy (yunion.io/x/sqlchemy)

受python SQLAlchemy启发的golang ORM

- 通过结构体定义数据库表schema

```
type SUser struct {
    Id               string            `width:"64" charset:"ascii" nullable:"false" primary:"true"`
    Extra            string            `charset:"utf8" nullable:"true"`
    Enabled          tristate.TriState `nullable:"true"`
    DefaultProjectId string            `width:"64" charset:"ascii" nullable:"true" index:"true"`
    CreatedAt        time.Time         `nullable:"true"`
    LastActiveAt     time.Time         `nullable:"true"`
    DomainId         string            `width:"64" nullable:"false" index:"true"`
}
```

实现结构体到数据库表schema的单向同步

```
+--------------------+-------------+------+-----+---------+-------+
| Field              | Type        | Null | Key | Default | Extra |
+--------------------+-------------+------+-----+---------+-------+
| id                 | varchar(64) | NO   | PRI | NULL    |       |
| extra              | text        | YES  |     | NULL    |       |
| enabled            | tinyint(1)  | YES  |     | NULL    |       |
| default_project_id | varchar(64) | YES  | MUL | NULL    |       |
| created_at         | datetime    | YES  |     | NULL    |       |
| last_active_at     | date        | YES  |     | NULL    |       |
| domain_id          | varchar(64) | NO   | MUL | NULL    |       |
+--------------------+-------------+------+-----+---------+-------+
```

sqlchemy – 结构化查询

- 结构化的数据库查询语句

```
q = UserTable().Query()
q = q.Equals(“domain_id”, “7140640a189d48e78994257f9802b1e5”)
```

```
SELECT * FROM user WHERE domain_id=‘7140640a189d48e78994257f9802b1e5’
```

- 避免人工拼凑SQL，通过golang的语法检查来保证SQL正确性
- 允许数据库查询逻辑的代码复用

structarg (yunion.io/x/structarg)

结构化的命令行参数定义

代码定义

```
type UserOptions struct {
    Help    bool     `help:"show Help messages"`
    Name    string   `help:"specify user's name" positional:"true"`
    Friends []string `help:"specify list of user's friends"`
}

parser, err := structarg.NewArgumentParser(&UserOptions{},
    "user",
    `Command-line interface test prog`,
    `See "user --help" for detailed help.`)
```

```
./user --help
Usage: user [--help] [--friends FRIENDS] <NAME>

Command-line interface test prog

Positional arguments:
    <NAME>
        specify user's name

Optional arguments:
    [--help]
        show Help messages
    [--friends FRIENDS]
        specify list of user's friends
```

配置文件（KV格式）

```
name = 'john'
friends = ['Emily', 'Tylor']
```

配置文件（YAML格式）

```
name: john
Friends:
  - Emily
  - Tylor
```

其他库 (yunion.io/x/pkg)

- 模仿python prettytable的prettytable (yunion.io/x/pkg/prettytable)
- Error处理 (yunion.io/x/pkg/error)
- 保证key值顺序的Map：sortedmap (yunion.io/x/pkg/sortedmap)

Golang的坑

01.Golang踩坑案例 - for...range

```
for k, v := range myMap {
    // v is a copy of myMap[k]
    v.modify() // null operation
}
```

建议：使用for...range遍历map或slice时，如果不确定，请只对key(for map)或index(for slice)进行遍历，在for ...range的body里，通过map[key]或slice[index]访问map或slice里的元素。

02.Golang踩坑案例 – “虚函数”

```
type Hello interface {
    GetName() string
    Hi()
}

type Base struct {
}

func (b *Base) GetName() string {
    return “nobody”
}

func (b *Base) Hi() {
    fmt.Println(“Hi, I’m %s”, b.GetName())
}

type Upper struct {
     Base
     Name string
}

func (u *Upper) GetName() string {
    return u.Name
}

func main() {
    u = &Upper{Name: “robot”}
    u.Hi() // Expect Hi, I’m robot. Got Hi, I’m nobody
}
```

```
type Base struct {
    virtual Hello
}

func (b *Base) SetVirtual(h Hello) {
    b.virtual = h
}

Func (b *Base) GetVirtual() Hello {
    return b.virtual
}

func (b *Base) Hi() {
    fmt.Println(“Hi, I’m %s”, b.GetVirtual().GetName())
}

func main() {
    u = &Upper{Name: “robot”}
    u.SetVirtual(u)
    u.Hi() // Hi, I’m robot.
}
```

03.Golang踩坑案例 – init()调用顺序

```
// a.go

func init() {
    fmt.Println(b+c) // Expect 101, Got 0
}
```

```
// b.go

var b int
func init() {
    b = 1
}
```

```
// c.go
var c int
func init() {
    c = 100
}
```

建议：golang文件的init方法是按照目录中文件名排序顺序调用的，不同文件的init方法应该避免相互依赖

04.Golang踩坑案例 – 值和指针的默认转换

```
type Hello interface {
    Hi()
    GetName() string
}

func (b *Base) Hi() {}
func (b *Base) GetName() string {
    return “name”
}

func main() {
    var h Hello
    h = &Base{} // compile success
    h = Base{} // compile fail
}
```

```
type Hello interface {
    Hi()
    GetName() string
}

func (b Base) Hi() {}
func (b *Base) GetName() string {
    return “name”
}

func main() {
    var h Hello
    h = &Base{} // compile success
    h = Base{} // compile fail
}
```

```
type Hello interface {
    Hi()
    GetName() string
}

func (b Base) Hi() {}
func (b Base) GetName() string {
    return “name”
}

func main() {
    var h Hello
    h = &Base{} // compile success
    h = Base{} // compile success
}
```

Golang调用变量的方法时，会静默地进行指针到值的转换。实现变量方法时，除非明确需要，应实现指针的方法以避免混淆。

2021

THANK YOU

云联壹云核心产品融合云 云联壹云 秉承：简单、开放、融合、智能的产品理念，能够帮助企业实现异构IT基础设施的全面云化、统一管理及成本优化，提高运维效率的同时，降低企业运营成本。

邱剑 2021-03-31

扫码进技术交流群

更多资讯关注我