API 设计

毛剑/bilibili 基础架构部

目录

IDL: Protobuf

Protocol Buffers 是一种简单的语言中立并且平台中立的接口定义语言（IDL），用于定义数据结构模式和编程接口。它支持二进制和文本传输格式，并且在不同平台上使用不同的传输协议。

- *数据结构定义；*
- *配置文件定义；*
- *编程接口定义；*

唯一定义是我使用 Protobuf 的重要原因。声明即代码，声明即文档。

```
service LibraryService {
  rpc GetBook(GetBookRequest) returns (Book) {
    option (google.api.http) = {
      get: "/v1/{name=shelves/*/books/*}"
    };
  };
  rpc CreateBook(CreateBookRequest) returns (Book) {
    option (google.api.http) = {
      post: "/v1/{parent=shelves/*}/books"
      body: "book"
    };
  };
}

message Book {
  //书的资源名称。格式必须是："shelves/*/books/"
  //比如："shelves/shelf1/books/book2"。
  string name = 1;

  // ... 其他属性
}

message GetBookRequest {
  //书的资源名称。"shelves/shelf1/books/book2"。
  string name = 1;
}

message CreateBookRequest {
  // 新建书的父资源的资源名称
  // 比如"shelves/shelf1".
  string parent = 1;
  // 要创建的书籍资源，客户端绝不能设置‘Book.name’属性
  Book book = 2;
}
```

IDL: Protobuf 文档化

在 .proto 文件中可以使用 Protocol Buffers 通常的注释格式(\\)来添加注释。

- API *描述*
- *资源描述*
- *字段和参数描述*
- *方法描述*


```yaml
openapi: 3.0.3
info:
    title: Greeter
    description: The greeting service definition.
    version: 0.0.1
paths:
    /helloworld/{name}:
        get:
            summary: Sends a greeting
            operationId: Greeter_SayHello
            parameters:
                - name: name
                  in: query
                  schema:
                    type: string
            responses:
                "200":
                    description: OK
                    content:
                        application/json:
                            schema:
                                $ref: '#/components/schemas/HelloReply'
components:
    schemas:
        HelloReply:
            properties:
                message:
                    type: string
            description: The response message containing the greetings
```

gRPC 官方就使用 proto 作为描述文件，各类工具都使用 proto 文件作为元信息，编译出各语言适用的 client SDK；
HTTP 的 API 我们使用 OpenAPI 作为描述文件，每个服务应当提供准确无误的 OpenAPI 文件作为 API 的调用描述。各类工具应当根据 OpenAPI 文件生成相应的 API SDK。
如果该 API 和 gRPC 均使用 proto 定义，那么直接使用我们提供的 protoc-openapi 工具即可生成相应的 OpenAPI 文件。

目录

API 设计规范

在面向资源设计的 API 里，资源是有命名的实体，资源路径就是其标识。每一个资源必须有其独立的资源路径。资源路径由资源名称、其父资源的名称及其业务域组成。

- *业务域（Business Domain）：业务域是指可以独立完成某一个业务功能的服务相关设施集合；*
- *资源路径（Resource Path）:资源绝对路径由业务域和资源相对路径构成；*
  - Path：/<appid>.<version>.<Service>/<Method>
  - *用户信息：*/account.user.interface.v1.Profile/GetProfile
- *应用服务（Application）：表示一个服务的名称，三段式命名；*
  - appid = <domain>.<biz>.<app>

API 名称	示例
产品名称	Google **Calendar** API
服务名称	**calendar.googleapis.com**
包名称	**google.calendar.v3**
接口名称	**google.calendar.v3.CalendarService**
源代码目录	**//google/calendar/v3**
API 名称	**calendar**

使用向左的斜线 / 来划分资源名中不同的部分，其转发规则：

- DCDN（边缘 CDN 加速）：/<appdomain>
- API Gateway：/<appid>.<version>

API 设计规范

在谷歌 API 仓库中，超过70%的 API 属于标准方法。标准方法更容易学习和使用。包括了 List, Get, Create, Update, and Delete。很多不同类型的 API 都拥有非常类似的语义，把它们归纳为标准方法能够显著降低复杂度并提高一致性。

方法	HTTP 方法映射	HTTP 请求体	HTTP 返回体
List	**GET <集合URL>**	空	资源* 列表
Get	**GET <资源URL>**	空	资源*
Create	**POST <集合URL>**	资源	资源*
Update	**PUT or PATCH <资源URL>**	资源	资源*
Delete	**DELETE <资源URL>**	空	空**

字段名	类型	描述
name	**string**	**name**字段应该包含相对资源名
parent	**string**	对于资源定义和 `List` / `Create` 请求，**parent**字段应包含父级相对资源名
create_time	**Timestamp**	一个实体的创建时间戳
update_time	**Timestamp**	一个实体的最后更新时间戳；注意update_time会被create/patch/delete等操作更新
delete_time	**Timestamp**	实体的删除时间戳，仅当支持保留时。
time_zone	**string**	时区名，它应该符合IANA时区标准，如”America/Los_Angeles”。有关详细信息，请参阅 https://en.wikipedia.org/wiki/List_of_tz_database_time_zones.
region_code	**string**	位置的Unicode国家/地区代码（CLDR），例如“US”和“419”。有关详细信息，请参阅 http://www.unicode.org/reports/tr35/#unicode_region_subtag。

方法名	自定义动词	HTTP动词	备注
Cancel	**:cancel**	**POST**	取消一个未完成的操作（构建，计算等等）
BatchGet	**:batchGet**	**GET**	批量获取多个资源（查阅 `List` 标准方法中的详细描述）
Move	**:move**	**POST**	将一个资源从一个父级移到另一个
Search	**:search**	**GET**	`List` 的语义不足够时，搜索获取数据
Undelete	**:undelete**	**POST**	恢复之前删除的数据；推荐的数据的保留时间是30天。

“Gofmt's style is no one's favorite, yet gofmt is everyone's favorite.” - Rob Pike

API 设计规范

向后兼容的修改：

- *将 API 接口添加到 API 服务定义*
- *向 API 接口添加方法*
- *向方法添加 HTTP 绑定*
- *向请求消息中添加字段*
- *向响应消息中添加字段*
- *向枚举添加值*
- *添加仅输出的资源字段*

向后不兼容的修改：

- *删除或重命名服务，字段，方法或枚举值*
- *更改 HTTP 绑定*
- *更改字段的类型*
- *更改资源命名格式*
- *更改现有请求的可见行为*
- *更改 HTTP 定义中的 URL 格式*
- *向资源消息添加读/写字段*

目录

API 错误处理

用简单的协议无关错误模型，这使我们能够在不同的 API，API 协议（如 gRPC 或 HTTP）以及错误上下文（例如，异步，批处理或工作流错误）中获得一致的体验。

- 使用一小组标准错误配合大量资源
  - *服务器没有定义不同类型的“找不到”错误，而是使用一个标准 google.rpc.Code.NOT_FOUND 错误代码并告诉客户端找不到哪个特定资源。*
  - *状态空间变小降低了文档的复杂性，在客户端库中提供了更好的惯用映射，并降低了客户端的逻辑复杂性，同时不限制是否包含可操作信息。*

```
message Error {
  int32 code = 1;
  string reason = 2;
  string message = 3;
  map<string, string> metadata = 4;
};
```

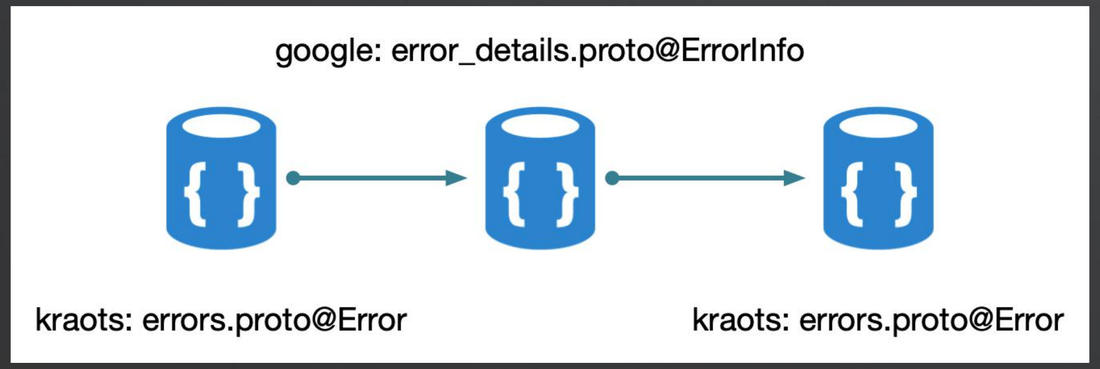
```
{
    // 错误码，跟 http-status 一致，并且在 grpc 中可以转换成 grpc-status
    "code": 500,
    // 错误原因，定义为业务判定错误码
    "reason": "USER_NOT_FOUND",
    // 错误信息，为用户可读的信息，可作为用户提示内容
    "message": "invalid argument error",
    // 错误元信息，为错误添加附加可扩展信息
    "metadata": {}
}
```

API 错误处理

- 错误传播
  - *如果您的 API 服务依赖于其他服务，则不应盲目地将这些服务的错误传播到您的客户端。*

在翻译错误时，我们建议执行以下操作：

- *隐藏实现详细信息和机密信息。*
- *调整负责该错误的一方。例如，从另一个服务接收 INVALID_ARGUMENT 错误的服务器应该将 INTERNAL 传播给它自己的调用者。*




```protobuf
message Error {
  int32 code = 1;
  string reason = 2;
  string message = 3;
  map<string, string> metadata = 4;
};
```

```
{
    // 错误码，跟 http-status 一致，并且在 grpc 中可以转换成 grpc-status
    "code": 500,
    // 错误原因，定义为业务判定错误码
    "reason": "USER_NOT_FOUND",
    // 错误信息，为用户可读的信息，可作为用户提示内容
    "message": "invalid argument error",
    // 错误元信息，为错误添加附加可扩展信息
    "metadata": {}
}
```

API 错误处理

从 Client 消费端只能看到 api.proto 和 errors.proto 文件，相应的生成的代码，就是调用侧的 api 以及 errors enum 定义。

- *使用 kratos errors.As() 拿到具体类型，然后通过 Reason 字段进行判定；*
- *使用 kratos errors.Reason() helper 方法（内部依赖标准库 errors.As)快速判定；*

```proto
enum ErrorReason {
  GEETER_UNSPECIFIED = 0;
  USER_NOT_FOUND = 1;
  USER_USER_BLOCKED = 2;
}
```

```go
result, err := uc.repo.CreateGreeter(ctx, g)
if err != nil {
    if errors.Reason(err) == v1.ErrorReason_USER_NOT_FOUND) {
        // TODO: do something
    }
    return err
}

if err != nil {
    if se := new(errors.Error); errors.As(err, &se) {
        switch se.Reason {
        case v1.ErrorReason_USER_NOT_FOUND:
            // TODO
        }
    }
    return err
}
```

目录

References

[1] https://github.com/go-kratos/kratos

[2] https://github.com/googleapis/googleapis/blob/master/google/rpc/error_details.proto#L112

[3] https://github.com/pkg/errors

[4] https://mp.weixin.qq.com/s/cBXZjg_R8MLFDJyFtpjVVQ

[5] Modifying gRPC Services over Time

[6] https://www.bookstack.cn/read/API-design-guide/API-design-guide-README.md

[7] https://google.aip.dev/general

[8] https://go-kratos.dev/docs/guide/api-protobuf/

毛剑
资深 Go 工程师
某互联网公司技术总监

全新升级

Go 进阶训练营

抢占先机，成为未来 3 年抢手的后端开发人才

请选择班期

- 课程设计对标字节跳动 2–2 级能力模型
- 实践驱动，系统提升你的代码硬实力
- 简历直推一线互联网公司
- 掌握云计算时代第一编程语言

NEW 60 小时	13 大	NEW 20 小时	3 大	456 天	2 次
课程内容	内容模块	领教直播	项目实战	课程有效期	企业内推

领取优惠

直播优惠价：￥5399

本期开营时间：5月9日

如何高效学习 Go？

极客时间 训练营

主讲老师：毛剑

资深 Go 工程师 某大型互联网公司技术总监

毛剑有近十年的服务端研发经验，他擅长高性能、高可用的服务端研发。是忠实的 Go 语言粉丝，也是 Go 语言老手。

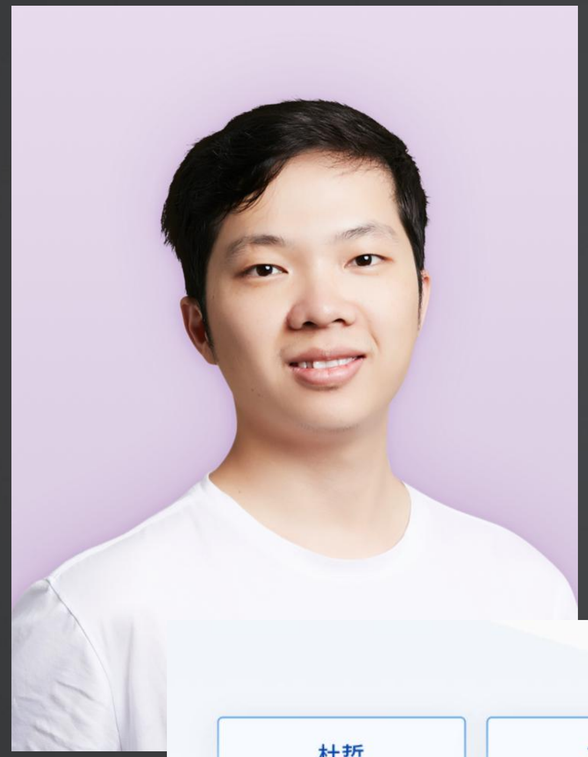
他曾在 GitHub 上开源了几个基于 Go 语言的项目。

全程参与了某大型互联网公司从单体架构到微服务架构的完整转型、对微服务治理、可用性设计、数据一致性设计、缓存、消息队列、监控、日志、负载均衡，以及 RPC 框架等非常有经验。

另外，毛剑作为嘉宾，也在 QCon、ArchSummit、GopherChina 等业界知名会议上做 Go 相关技术分享

如何高效学习 Go ？

足够优秀的领教



邓大明

本硕就读于南京大学。

曾就职于大众点评和 eBay，目前在新加坡某头部电商企业工作。

热爱开源，是 Beego dubbogo 等多个 Go 语言开源项目的代码贡献者。

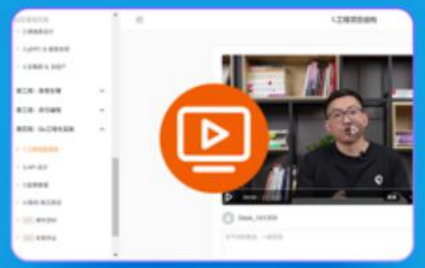
助教团

杜哲 前百度研发工程师	**方圆** 前新浪微博架构师	**Tony** 哔哩哔哩 资深开发工程师	**听闻** 腾讯高级工程师	**Viktor** 前 360 公司架构师	**Dandy** 前腾讯视频 高级后台开发工程师
Joe Apache/Dubbo-go PMC 资深软件工程师	**邓大明** 前大众点评软件 工程师	**煎鱼** 畅销书《Go 语言编程之旅》 作者	**Lincoln** 前腾讯高级工程师	**乔乔** 前 YY 高级工程师	**麦浪** 美图秀秀研发工程师

上课形式和时间

毛剑（主讲）

设计学习内容
系统讲解知识点

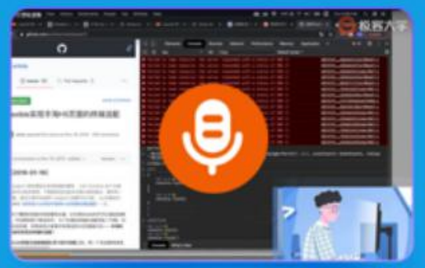


毛剑视频课

学习时长 40 小时

邓明（领教）

每周一次直播
搭配课程大纲讲解重难点



领教直播课

直播时长 20 小时

课程设计

- 课程设计对标字节 2-2 级能力模型
- 实践驱动，系统提升你的代码硬实力
- 简历直推一线互联网公司
- 掌握云计算时代第一编程语言

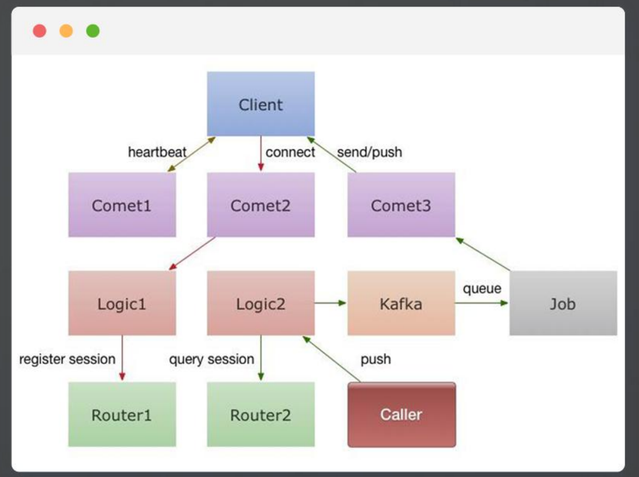
案例一：goim 网络编程实战

案例分析

海量网络连接的处理

连接的管理和消息的发送

大型分布式 IM 架构的设计



核心要点

Go 语言 TCP 网络的优化和处理

Go 语言海量连接的管理，内存优化

Go 语言分布式架构的实践

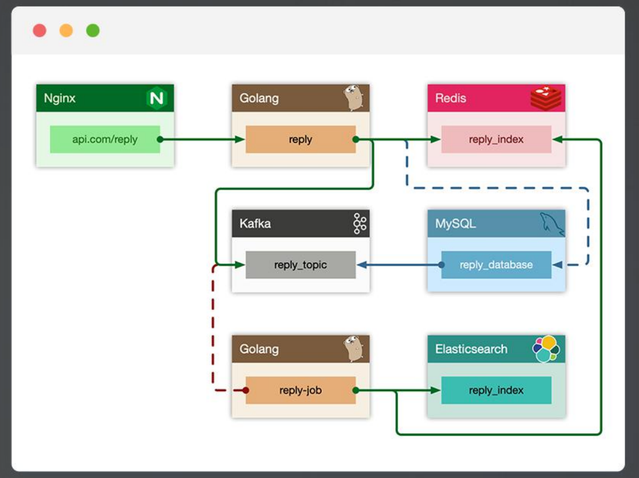
案例二：评论系统架构设计

案例分析

评论系统的存储、缓存设计

评论系统的可用性设计

评论系统的架构设计



核心要点

缓存的单飞加载，缓存的流控，缓存的优化技巧

缓存和存储的数据一致性设计，多级缓存的设计，热点缓存的应对

异步消息队列消峰设计

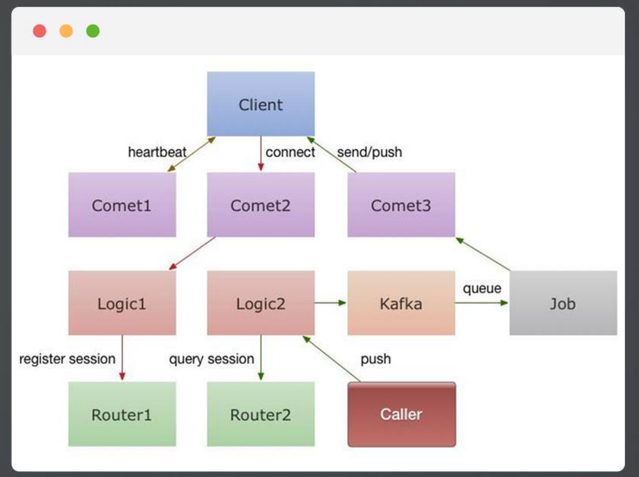
案例三：抽奖系统架构设计

案例分析

大型并发系统（抽奖）的设计

抽奖系统的缓存设计

抽奖系统的数据一致性设计



核心要点

抽奖系统的核心交易逻辑

Redis list 的奖池处理

1.每周按时完成当周全部课程内容

2.每周按时完成并提交当周作业

3.在规定时间内完成并提交毕业设计

往期学员学习效果

Go-2期[redacted]
对方不同意存档会话内容

心心 我来报告 我看了毛大的教程后 找到工作 准备去字节了

我谈下来的title是3-1 毛大的视频中讲的一些知识点 面试的时候确实提到了 在我后面谈薪的过程中蛮有优势的

嗯嗯 这个时候就是自己获得成就感的时

收获3 Offer 比毕业先来

嗨～沈同学～咱们跳槽啦?

是啊..

哇 怎么样！有没有涨薪什么的呀

涨薪60%多吧

不错不错～

是什么公司呀 方便说嘛

算法14期-[redacted]-Go0期 @微信

嗯嗯，是的[LetMeSee]

哈哈你入职哪里了

我跟着毛老师学完

果然功夫不负有心人 确认过眼神 是优秀的男孩子

明天去入职

b站

哈哈哈哈 这么棒！！

后来过了

可真优秀

算法训练营入门后，一直在做题

Go-2期[redacted]
对方不同意存档会话内容

一倍多，总包120w

年薪120?

嗯

我天呐 贫穷限制了我的想象

那你工作了几年啦 去字节做什么岗位呢

golang开发吗

我工作8年了，谈下来的title是3-1，过去带个小团队

算是个技术leader，字节进入基本都让转go了

喜报


Offer 喜报
京东 Offer
年薪 45w 涨薪 60%
Go 1 期 王同学
Golang 工程师
学习 13 周后
斩获
京东 offer
课程好 | 服务好 | 学风好
极客时间训练营让你学会，并学有所成


Offer 喜报
字节跳动 Offer
年薪 35w 涨薪 60%
Go 1 期 辛小鱼同学
Golang 工程师
学习 13 周后
斩获
字节跳动 offer
课程好 | 服务好 | 学风好
极客时间训练营让你学会，并学有所成


Offer 喜报
B 站 Offer
年薪 51w 涨薪 30%
Go 1 期 伯爵同学
Golang 工程师
学习 13 周后
斩获
B 站 offer
课程好 | 服务好 | 学风好
极客时间训练营让你学会，并学有所成

THANKS