API 工程化

毛剑/bilibili 基础架构部

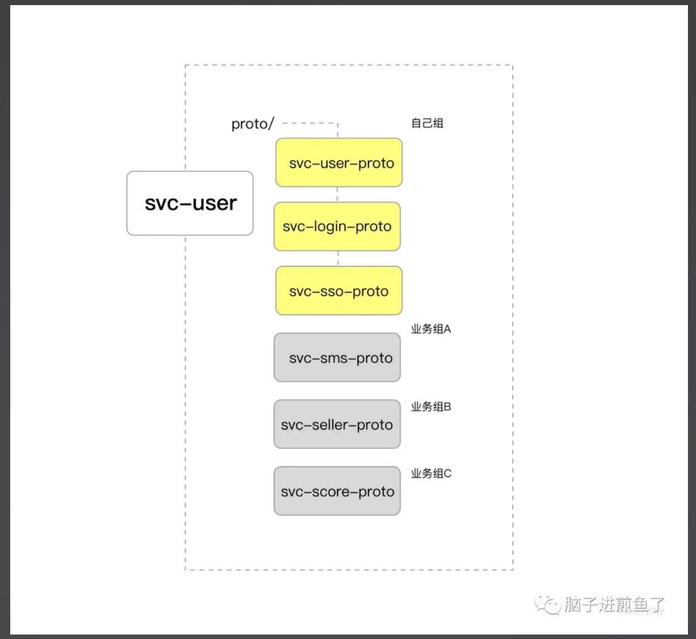
目录

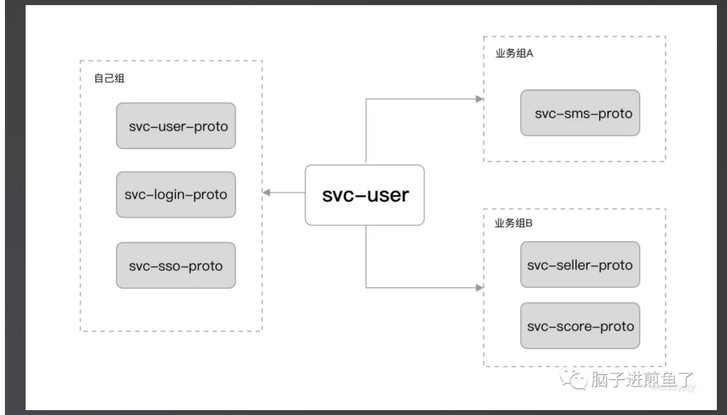
Proto 管理方式

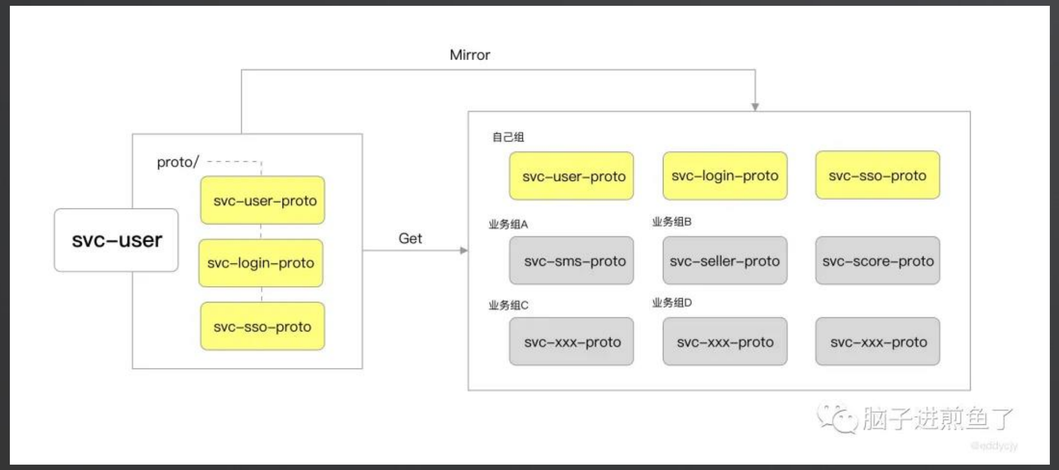
煎鱼的一篇文章：

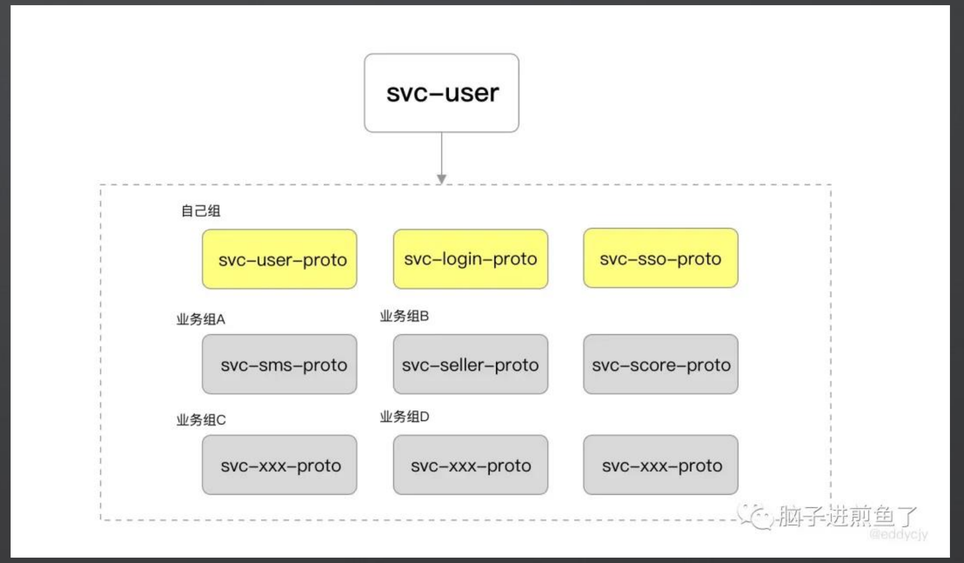
[真是头疼，Proto 代码到底放哪里？]

- 代码仓库
- 独立仓库
- 集中仓库
- 镜像仓库







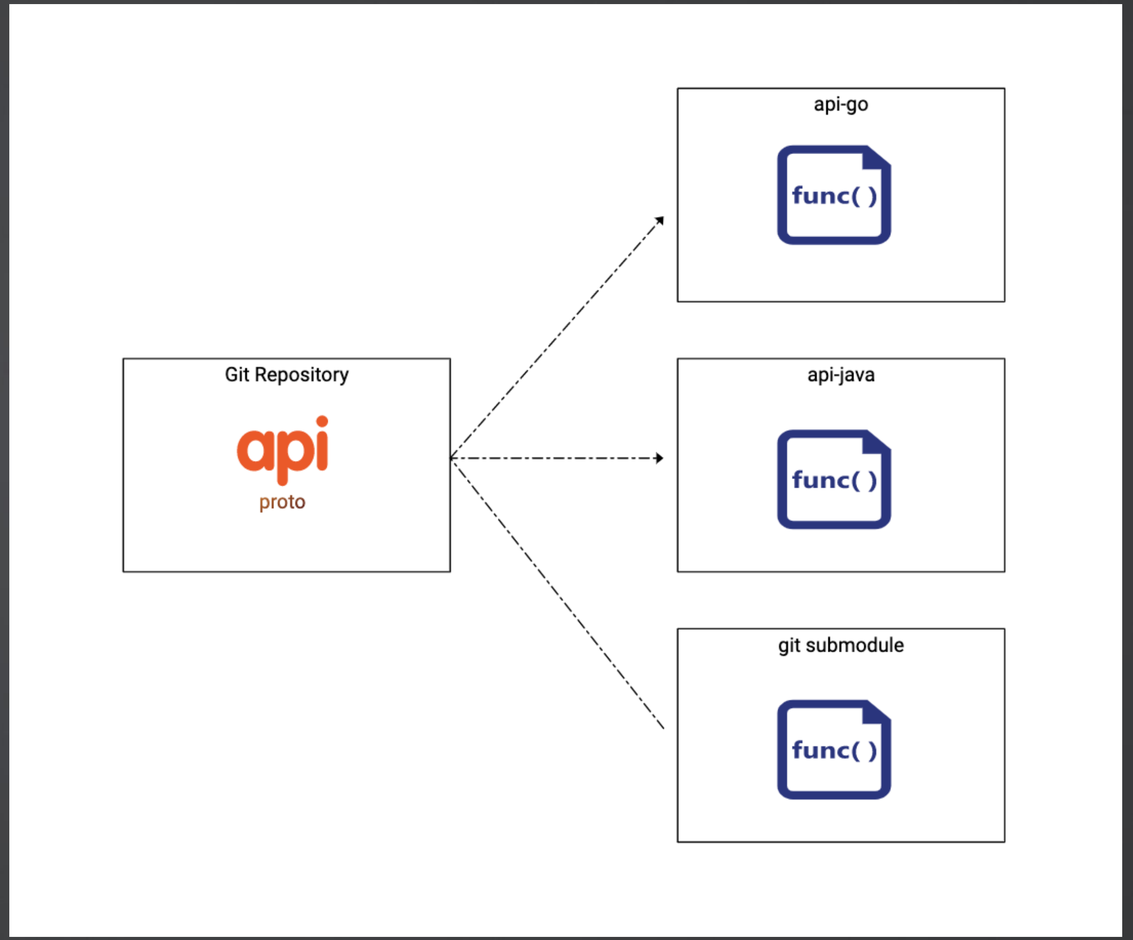


Proto 分仓源码方式

过去为了统一检索和规范 API，我们内部建立了一个统一的 bapis 仓库，整合所有对内对外 API。

- API 仓库，方便跨部门协作；
- 版本管理，基于 git 控制；
- 规范化检查，API lint；
- API design review，变更 diff；
- 权限管理，目录 OWNERS；

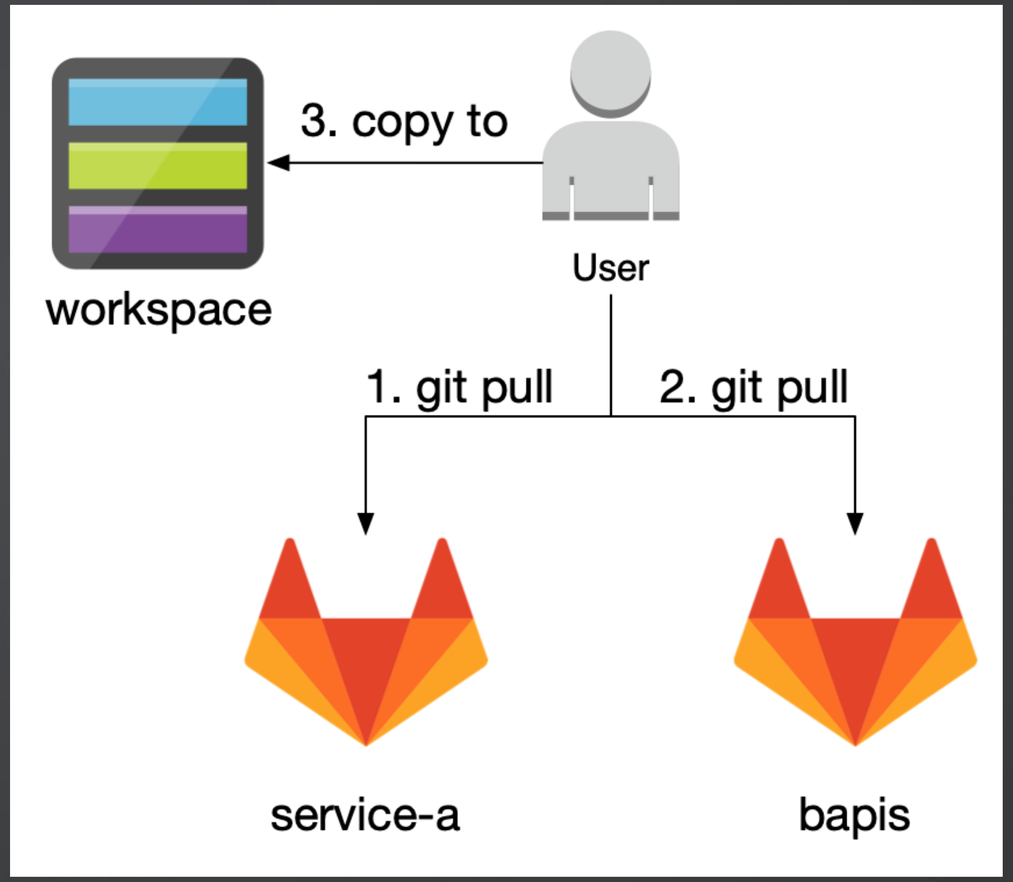
源码依赖会引入很多新的问题！



Proto 独立同步方式

移动端采用自定义工具方式，在同步代码阶段，自动更新最新的 proto 仓库到 workspace 中，之后依赖 bazel 进行构建整个仓库。

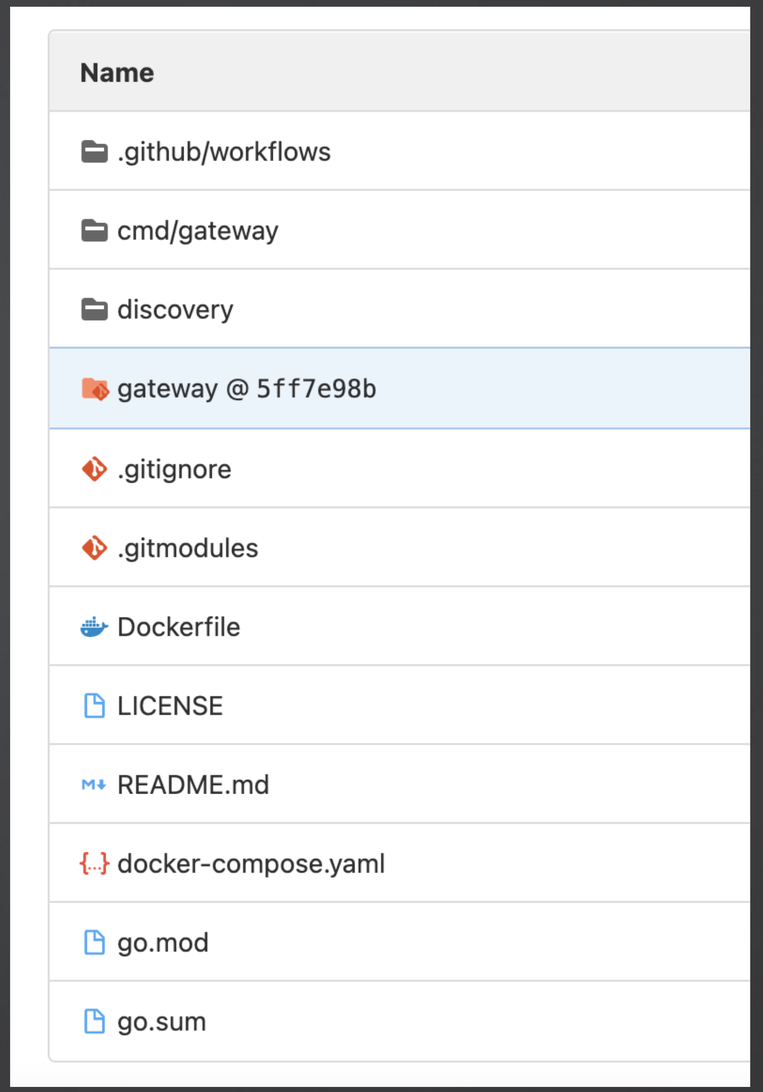
- *业务代码中不依赖 target 产物，比如 Objective-C 的 .h/.a 文件，或者 Go 的 .go 文件（钻石依赖、proto 未更新问题）*



Proto git submobule 方式

经过多次讨论，有几个核心认知：

- *proto 只有一份，不使用镜像方式同步，使用 git submobule 方式以仓库中目录形式来承载；*
- *本地构建工具 protoc 依赖 go module 下的相对路径即可；*
- *基于分支创建新的 proto，submodule 切换分支生成 stub 代码，同理 client 使用联调切换同一个分支；*
- *维护 Makefile，使用 protoc + go build 统一处理；*
- *声明式依赖方式，指定 protoc 版本和 proto 文件依赖（基于 BAZEL.BUILD 或者 Yaml 文件）*



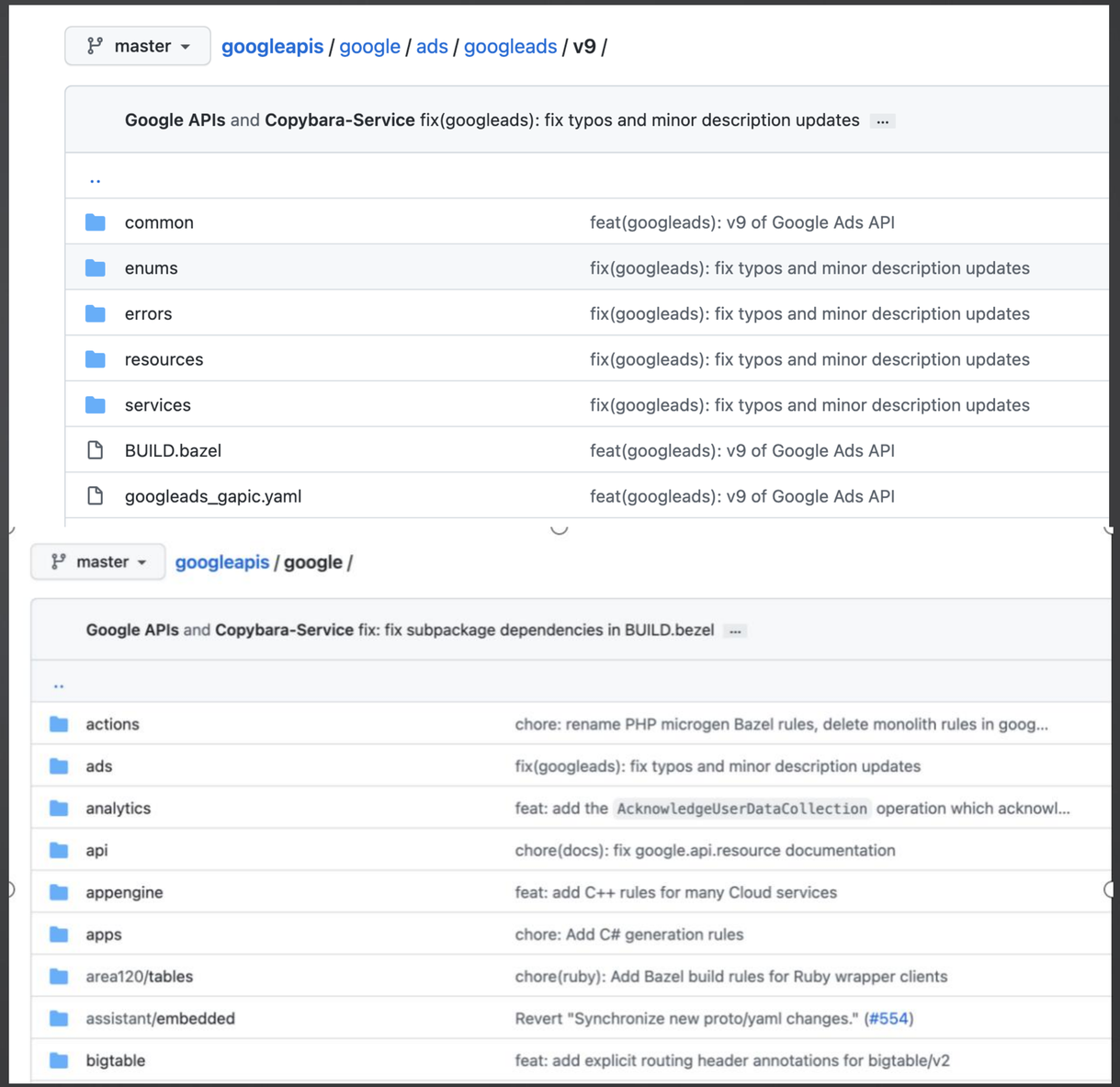
目录

Proto Project Layout

在统一仓库中管理 proto，以仓库为包名根目录:

- 目录结构和 package 对齐;
- 复杂业务的功能目录区分;
- 公共业务功能：api、rpc、type;

```
|____api // 服务API定义
| |____kratos
| | |____demo
| | | |____v1
| | | | |____demo.proto
|____annotations // 注解定义options
|____third_party // 第三方引用
```



目录

Proto Errors

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;
};
```

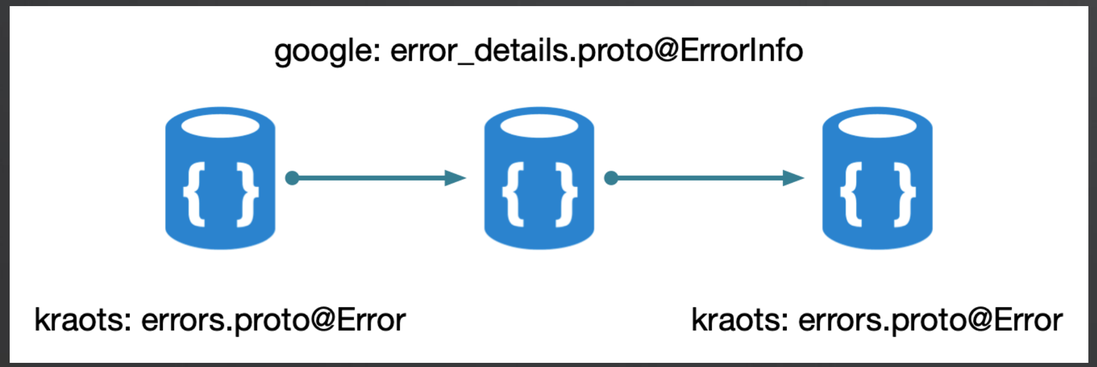
```
{
    // 错误码，跟 http-status 一致，并且在 grpc 中可以转换成 grpc-status
    "code": 500,
    // 错误原因，定义为业务判定错误码
    "reason": "USER_NOT_FOUND",
    // 错误信息，为用户可读的信息，可作为用户提示内容
    "message": "invalid argument error",
    // 错误元信息，为错误添加附加可扩展信息
    "metadata": {}
}
```

Proto Errors

- 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。*

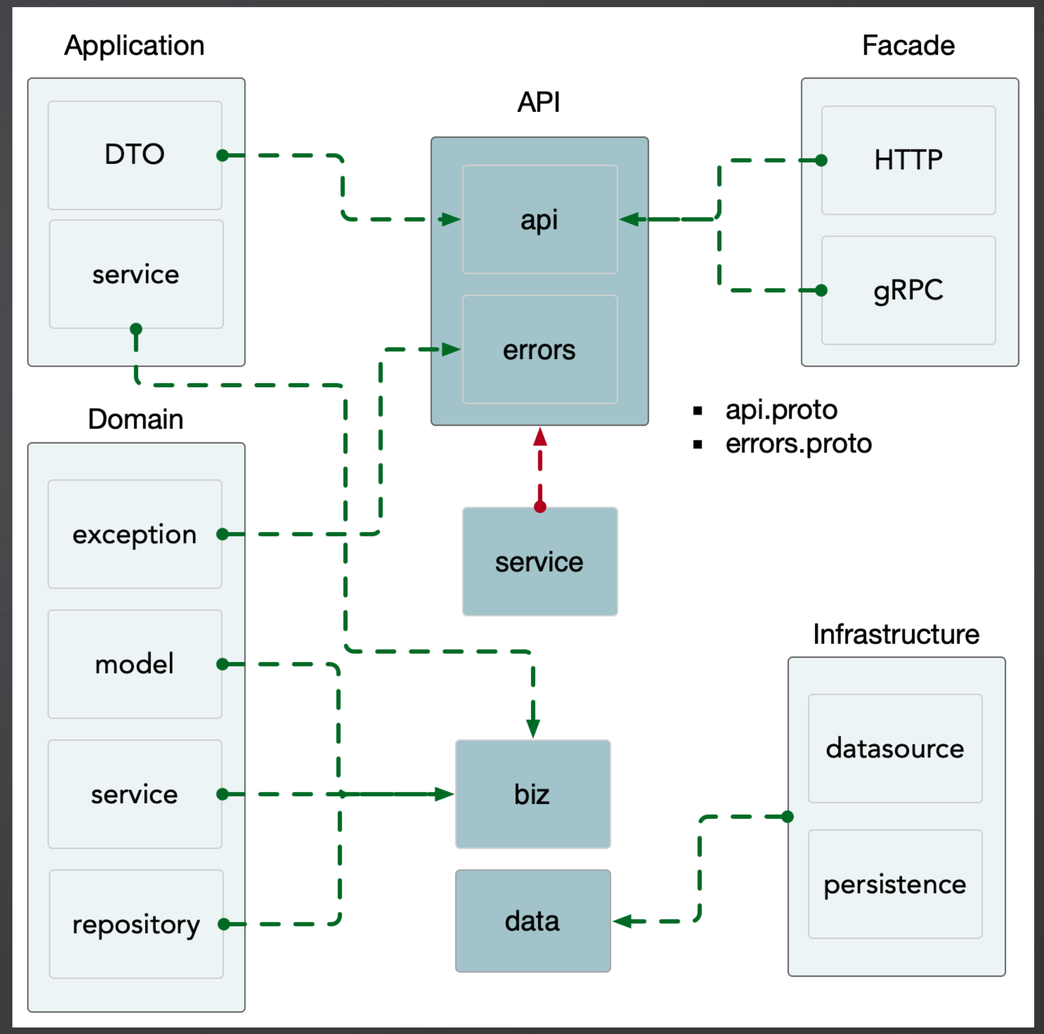



```
// Example of an error that is returned when attempting to create a Spanner
// instance in a region that is out of stock:
//
//     { "reason": "STOCKOUT"
//       "domain": "spanner.googleapis.com",
//       "metadata": {
//         "availableRegions": "us-central1,us-east2"
//       }
//     }
message ErrorInfo {
  // The reason of the error. This is a constant value that identifies the
  // proximate cause of the error. Error reasons are unique within a particular
  // domain of errors. This should be at most 63 characters and match
  // /[A-Z0-9_]+/.
  string reason = 1;

  // The logical grouping to which the "reason" belongs. The error domain
  // is typically the registered service name of the tool or product that
  // generates the error. Example: "pubsub.googleapis.com". If the error is
  // generated by some common infrastructure, the error domain must be a
  // globally unique value that identifies the infrastructure. For Google API
  // infrastructure, the error domain is "googleapis.com".
  string domain = 2;

  // Additional structured details about this error.
  //
  // Keys should match /[a-zA-Z0-9-_]/ and be limited to 64 characters in
  // length. When identifying the current value of an exceeded limit, the units
  // should be contained in the key, not the value.  For example, rather than
  // {"instanceLimit": "100/request"}, should be returned as,
  // {"instanceLimitPerRequest": "100"}, if the client exceeds the number of
  // instances that can be created in a single (batch) request.
  map<string, string> metadata = 3;
}
```

Proto Errors: Server

errors.proto 定义了 Business Domain Error 原型，使用最基础的 Protobuf Enum，将生成的源码放在 biz 大目录下，例如 biz/errors

- *biz 目录中核心维护 Domain，可以直接依赖 errors enum 类型定义；*
- *data 依赖并实现了 biz 的 Reporisty/ACL，也可以直接使用 errors enum 类型定义；*
- *TODO: kratos errors 需要支持 cause 保存，支持 Unwrap();*



Proto Errors: Client

从 Client 消费端只能看到 api.proto 和 errors.proto 文件，相应的生成的代码，就是调用测的 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 errors.As)快速判定；*

```
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

Proto Errors: Client

基于 openapi 插件 + IDL Protobuf 注释（IDL 即定义，IDL 即代码，IDL 即文档），最终可以在 Makefile 中使用 make api 生成 openapi.yaml，可以在 gitlab/vscode 插件直接查看。

- *API Metadata 元信息用于微服务治理、调试、测试等；*

```
.PHONY: api
# generate api proto
api:
        protoc --proto_path=. \
               --proto_path=./third_party \
               --go_out=paths=source_relative:. \
               --go-http_out=paths=source_relative:. \
               --go-grpc_out=paths=source_relative:. \
               --openapi_out==paths=source_relative:. \
               $(API_PROTO_FILES)
```


```
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
```

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

极客时间 训练营

THANKS