**Learning**

**Hertztools 生成的框架结构**

|  |
| --- |
| Go . ├── biz │ ├── config │ ├── dal │ ├── handler │ │ └── ping.go │ └── service ├── build.sh ├── conf │ ├── config.yaml │ └── hertz.config.yaml ├── go.mod ├── main.go ├── router.go ├── router\_gen.go └── script  └── bootstrap.sh |

biz代表business，业务，config下面的config.go是封装好的读取配置文件（yml）的，dal是数据库相关，也有其他语言、其他场景喜欢叫dao，handler对应java里的handler，即http请求的实际处理方法，其路径与方法的映射，在外层文件夹router里配置。service层是handler与dal的中间层，业务一般写在这里，dal层和handler层不建议写过于复杂的业务。conf文件夹就是配置文件啦~字节目前的配置方案是优先读取远程配置文件（ee/conf）读不到就会降级读取本地配置文件，配置文件不是热读取的，仅在服务启动的时候读取一次。router\_gen.go是自动生成的，不能改，也不用管。script文件夹下面是架构组同学提交帮忙写好的脚本，运行即可一键编译，如果自己想在本地goland单步调试的话，就要配置环境变量之类的了~

**Hertz 创建 RPC Client 调用下游服务**

HTTP Client (Client calls or curl) -> HTTP Server (Hertz server) && RPC Client (KiteX Client) -> RPC Server (KiteX Server)

* HTTP Client：curl
* HTTP Server & RPC Client：Hertz/Turorial/Part2搭建的服务端
* RPC Server：example\_item服务，在KiteX/Tutorial/Part2中

**KiteX 脚手架**

|  |
| --- |
| Go . ├── build.sh // 用来编译的脚本，一般情况下不需要更改 ├── conf // 用来存放配置文件的目录，一般情况下不需要更改 │ └── kitex.yml // 配置文件，可以通过修改里面的值来更改一些配置 ├── go.mod // go module 文件 ├── handler.go // 服务端的业务逻辑都放在这里，这也是我们需要更改和编写的文件 ├── idl // 示例 idl 存放的目录 │ ├── base.thrift │ └── kitex\_example\_item.thrift ├── kitex\_gen // kitex 生成代码存放的目录，不可更改，更改后重新生成会丢失改动 │ ├── base │ │ ├── base.go // 根据 IDL 生成的编解码文件 │ │ ├── k-base.go // kitex 专用的一些拓展内容 │ │ └── k-consts.go │ └── kitex │ └── example │ └── item │ ├── itemservice // kitex 封装代码主要在这里 │ │ ├── client.go │ │ ├── invoker.go │ │ ├── itemservice.go │ │ └── server.go │ ├── k-consts.go │ ├── k-kitex\_example\_item.go // kitex 专用的一些拓展内容 │ └── kitex\_example\_item.go // 根据 IDL 生成的编解码文件 ├── kx.yml // 如果使用 kx 的话会生成的文件 ├── main.go // go 中的 main 函数，一般在这里做一些资源初始化的工作，可以更改 └── script // runtime 标准环境所需的一些启动脚本，一般情况下不需要更改  ├── bootstrap.sh  └── settings.py |

**RPC protocols**

* Serialization of structured data is a key step to transmit information over networks or to store data because is a very cpu-intensive task. Actually, in many communication scenarios the **bottleneck** is the data serialization and deserialization.
* Developing a Serialization mechanism involves the design of a neutral format, platform independent, to allow data interchange over heterogeneous distributed systems.
* Middleware alternatives based on verbose serialization formats such as XML or JSON, used in Web Services and REST, expose very poor performance. The emergence of cloud computing and service integration in large distributed systems is driving companies and developers to consider again fast binary formats and lightweight Remote Procedure Call (RPC) frameworks. We compared Apache Thrift vs Protocol Buffers vs Fast Buffers.

[Protocol Buffers](https://code.google.com/p/protobuf/) is an alternative developed by Google and designed to be smaller and faster than XML. Protocol Buffers is the basis for a custom RPC engine used in nearly all inter-machine communication at Google.

[Apache Thrift](http://thrift.apache.org/) is an RPC framework developed at Facebook aiming “scalable cross-language services development”. Facebook uses Apache Thrift internally for service composition.

**Thrift related**

**Processor class**

Processor class is generated by the thrift. The Processor does two simple things. Reads data from an input stream and writes data to an output stream. The processor reads data from the input, processes the data (actually uses the handler specified by the user to process the data) and writes the processed data to the output.

**Example**

https://thrift-tutorial.readthedocs.io/en/latest/usage-example.html

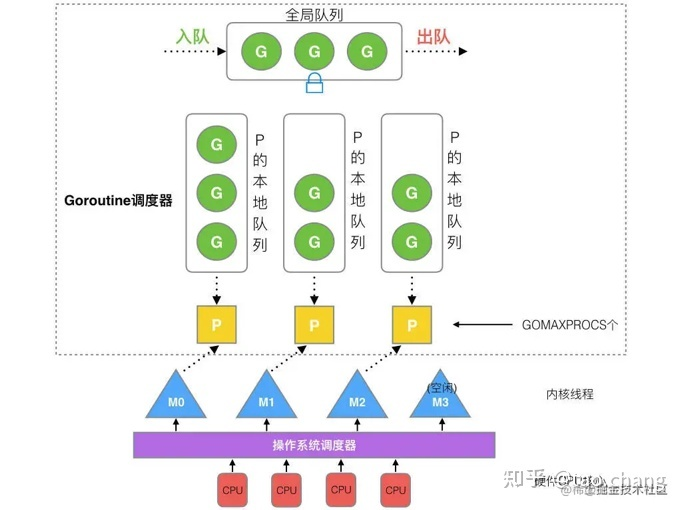
**Go Context**

https://go.dev/blog/context

* Context type carrys deadlines, cancellation signals, and other request-scoped values across API boundaries and between processes.
* Incoming requests to a server should create a Context, and outgoing calls to servers should accept a Context.

**GMP model**

G：goroutine 协程的概念，M：machine 机器线程的概念，P：processor协程的处理器或者更准确的说是调度器。



**写作习惯**

很多中文的文档包括产品需求的表述存在一个问题：没有对于概念/对象/需求的详细描述，描述抽象且句子中的信息高度浓缩。有系统英文写作经验的人会觉得作者并没有站在读者的角度上思考，导致作者写出来的东西很主观，很抽象，很浓缩。一些英文技术文档会对一个技术概念进行定义，解释定义，举例（比喻，类比，对比）来使一个zero prior knowledge的读者了解这个技术概念。但在一众中文文档中，你看不到这样的写作方式。英文文档一眼扫过去再加上稍加思考就可以获得信息，但是中文文档一眼扫过去要付出比前者更多的思考才可以获得同等的信息。这种现象会导致开发者的学习效率变低，团队间的书面沟通效率变低。当然这是从我的角度出发的（中文母语，大学内英文教学），但是可以确定的是我们缺少“如何写有逻辑，有层次，说的清楚的文章”的语文教育。故而每个中文母语者都要小心。

**Thrift API**

1. Data struct abstraction

AItemListRequest (common, filterConditions)

AItemListResponse (AItemListData, errmsg, errno)

* AItemListData (list<AItem> data, i64 count)
* AItem (detailed fields)

1. MySQL charset choose `utf8mb4`
2. 一些数据可以存成 plaintext，然后业务逻辑里面json解析
3. 少用foreign key, 不方便同步