如何从小白进阶成Go语言专家？

孔令飞、腾讯云技术专家

我是谁？

自我介绍：

- 工作经验：
  - Red Hat、联想云：Xen、KVM研发工程师
  - 目前在腾讯云：云函数SCF、容器服务TKE、分布式云中心TDCC等项目的设计和研发
- 多年Go项目开发经验，拥有大规模 Kubernetes 集群、微服务的研发和架构经验，对微服务、虚拟化、Kubernetes/Docker、Serverless等云计算相关技术有深层次的理解。目前专注于云原生分布式云领域的相关开发。
- 出版物
  - 图书：《从零构建企业级Go项目》
  - 极客时间专栏：《Go语言项目开发实战》
  - 掘金小册：《基于 Go 语言构建企业级的 RESTful API 服务》

目录

- Go语言概述
- 如何学习Go语言？
- 如何开发优秀的Go项目？
- 一个优秀的Go实战项目
- 基于声明式编程范式的软件架构
- 关于35岁焦虑的一点思考

Go语言概述

Go语言是什么？

Go出自名门Google公司，是一门支持并发 、垃圾回收的编译型高级编程语言。Go兼具静态编译语言的高性能以及动态语言的高开发效率。

该项目的三位领导者均是著名的语言学家：

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 Plan9，Inferno 操作系统和 Limbo 编程语言

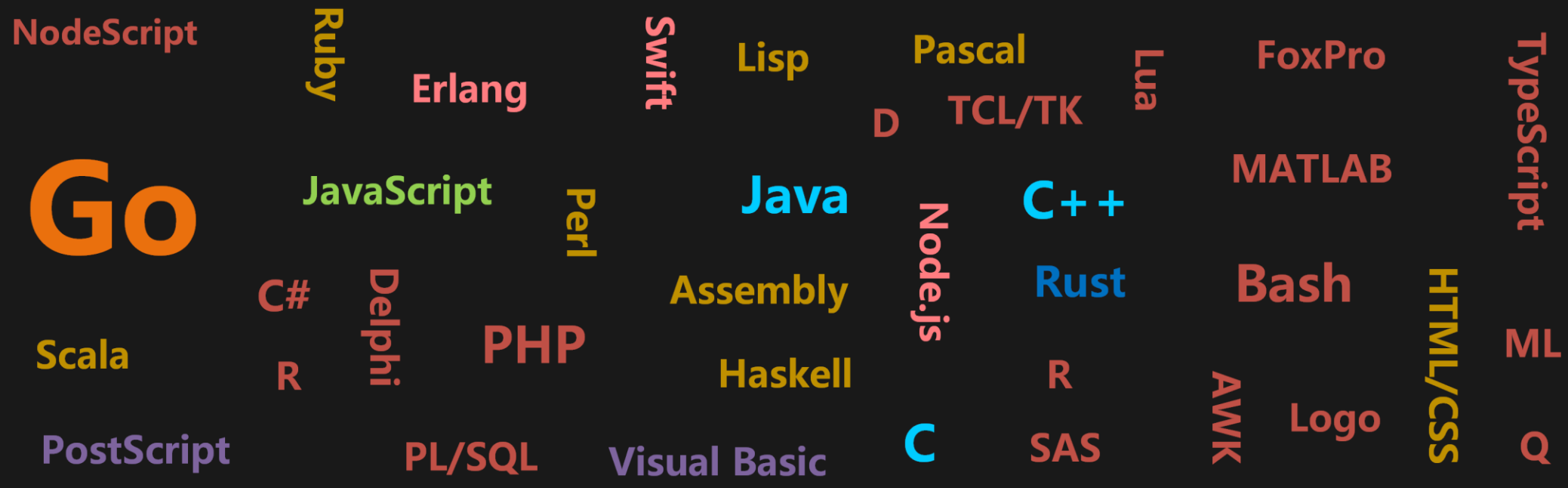
Ken Thompson：

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、 V8 Javascript engine

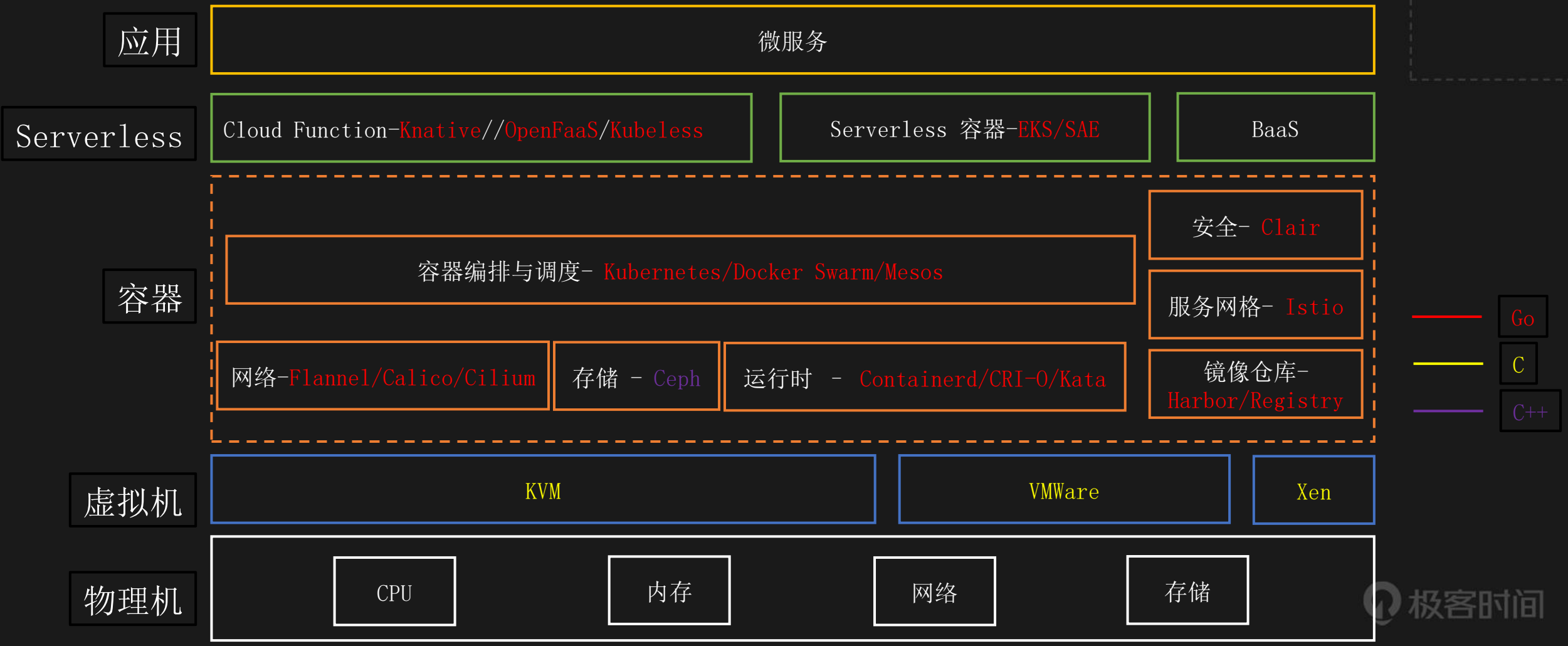
为什么选 Go?


NodeScript
Ruby
Erlang
Swift
Lisp
Pascal
Lua
FoxPro
TypeScript
Go
JavaScript
Perl
Java
D
TCL/TK
C++
MATLAB
Node.js
C#
Delphi
Assembly
Rust
Bash
HTML/CSS
ML
Scala
PHP
Haskell
R
AWK
Logo
R
C
SAS
Q
PostScript
PL/SQL
Visual Basic

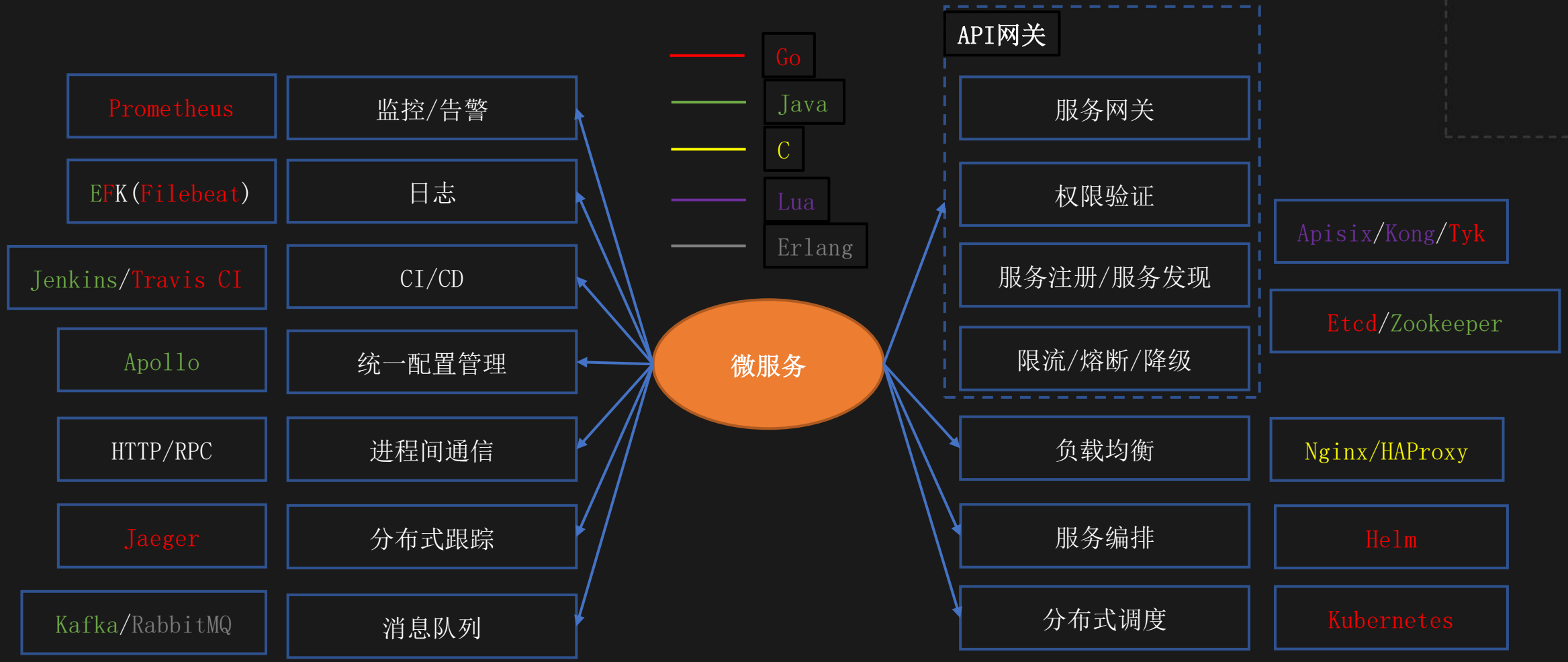
为什么选 Go?

- 语言简单、开发效率高
- 高效的垃圾回收机制
- 支持多返回值
- 更丰富的内置类型：map、slice、channel、interface
- 语言层面支持并发编程
- 编译型语言，编译即测试
- 跨平台编译

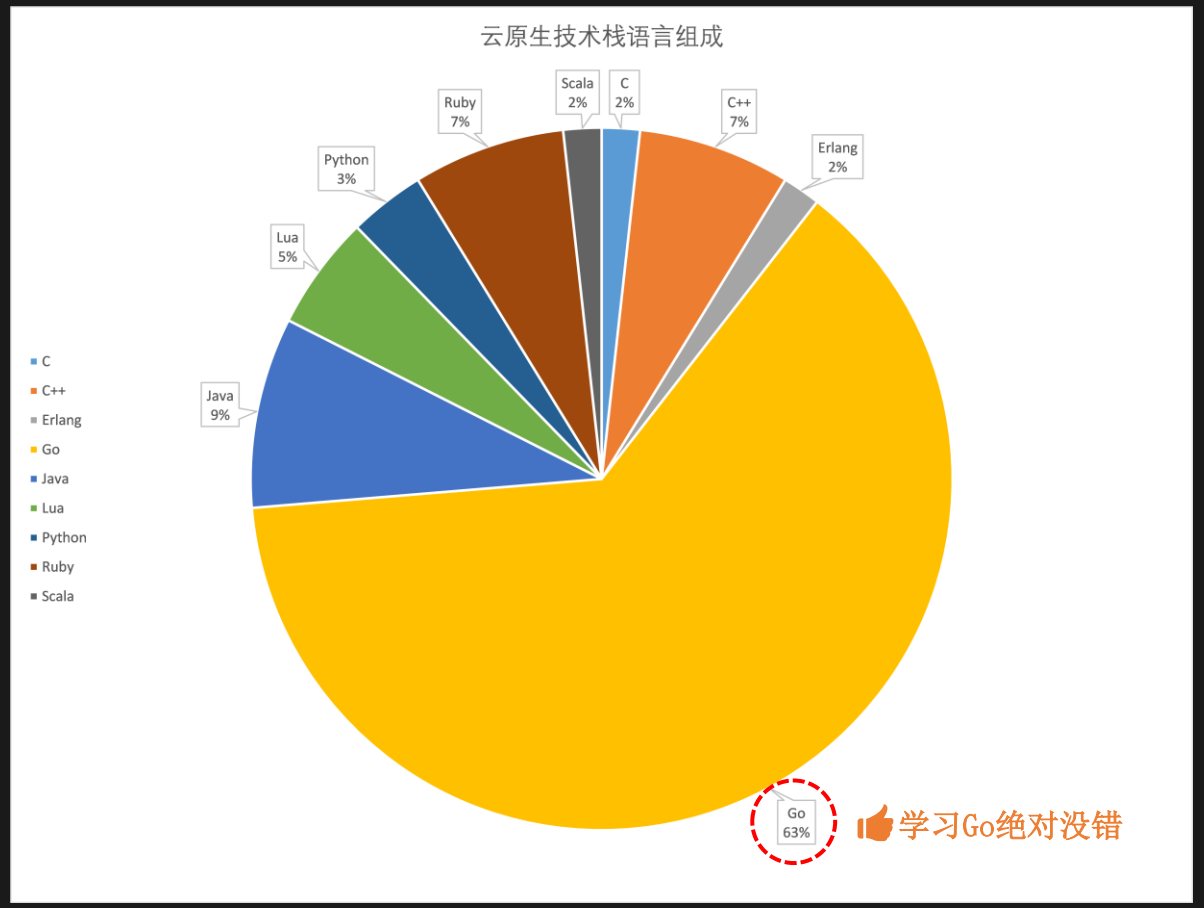
Go是云时代的语言



Go是云时代的语言



Go是云时代的语言



- **云原生：**是一种构建和运行应用程序的方法，是一套技术体系和方法论。云原生中包含了 3 个概念，分别是技术体系、方法论和云原生应用。整个云原生技术栈是围绕着 Kubernetes 来构建的。
- 云原生技术体系：七大技术板块，22 个技术技术栈，近百个开源项目

Go语言现状

美国程序员薪资全球最高，Go在招聘中最吃香，安全工程师薪资涨幅最高 | 软件工程师年度报告出炉

CSDN 已认证帐号 ＋ 关注

整理 | 于轩

出品 | 程序人生 （ID：coder _life）

近日，招聘公司Hired发布了《2022年软件工程师状况》报告。Hired发现，随着人才竞争的加剧，招聘市场对软件工程师的需求持续增长，Hired网站上软件工程师在2021年收到的面试请求是2020年的两倍多。

在这份报告中，Hired分析了市场上企业和软件工程师之间超36.6万次的互动数据。为了揭示该领域的最新趋势，报告还调查了Hired上2000多名软件工程师，研究了行业最紧缺的技能和职位、不同市场的薪资、远程招聘的变化，以及软件工程师在考虑新工作时最看重的因素等方面。

Hired希望这份报告能够为软件工程师提供参考和资源，促进他们的职业发展或在该领域开始职业生涯；同时为企业雇主提供如何吸引和保留技术人才的建议。以下是一些关键的发现：

- Go连续第二年成为最需要的技能，精通Go的软件工程师收到的面试请求比市场平均水平高出1.8倍。
- 网络安全人才短缺，网络攻击的兴起导致更多公司优先考虑其安全战略。因此，安全工程师的需求量很大，目前在所有软件工程角色中，安全工程师的平均工资最高。
- 公司正在招聘更多的全栈工程师。他们是需求量最大的职位，与其他软件工程职位相比，他们的面试请求增幅最大。公司寻求最大限度地提高他们工程团队的效率，并在这个动荡的就业市场和

如何学习Go语言？

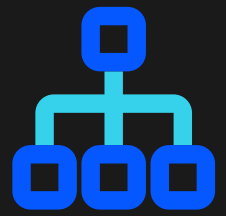
如何学习Go语言?

Go基础语法

学习方法

Go项目开发

详细介绍如何做



软件架构能力

学习路线

Go语言能力级别划分

初级

- 已经学习完Go基础语法课程，能够编写一些简单Go代码段；
- 或者借助于Google/Baidu能够编写相对复杂的Go代码段；
- 这个阶段开发者基本具备阅读Go项目代码的能力。

中级

- 能够独立编写完整的Go程序，例如：功能简单的Go工具等；
- 或者借助Google/Baidu能够开发一个完整、简单的Go项；
- 此外，对于项目中涉及到的其他中间件，也能知道怎么使用Go语言进行交互；
- 在这个阶段，开发者也能够二次开发一个相对复杂的Go项目。

高级

- 不仅能够熟练掌握Go基础语法，还能使用Go语言高级特性，例如channel、interface、并发编程等；
- 也能使用面向对象的编程思想去开发一个相对复杂的Go项目。

资深

- 熟练掌握Go语言编程技能与编程哲学，能够独立编写符合Go编程哲学的复杂项目；
- 同时，开发者需要对Go语言生态也有比较好的掌握，具备较好的软件架构能力。

专家

- 精通Go语言及其生态，能够独立开发大型、高质量的Go项目，编程过程中较少依赖搜索引擎，且对Go语言编程有自己的理解和方法论；
- 除此之外，还要具有优秀的软件架构能力，能够设计、并部署一套高可用、可伸缩的Go应用；
- 这个级别的开发者应该是团队的技术领军人物，能够把控技术方向、攻克技术难点，解决各种疑难杂症。

Go语言进阶方法（初级）

看书

实战

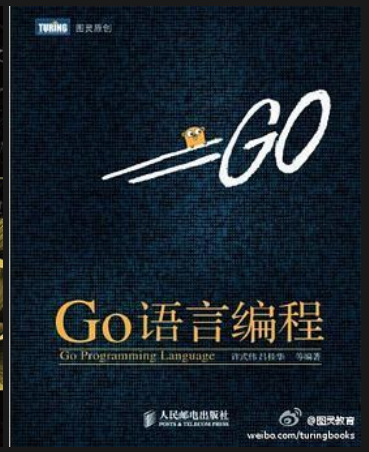
- 读2本经典书籍
- 动手编写示例代码段

- 以需求为驱动力
- 调研优秀开源项目
- 魔改优秀开源项目

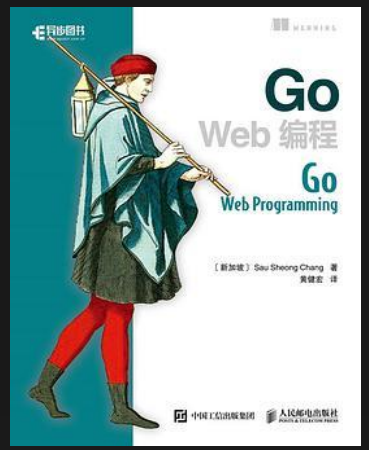
“抄” + 改

- 知识传播(文档、分享)

Go语言进阶方法（初级）



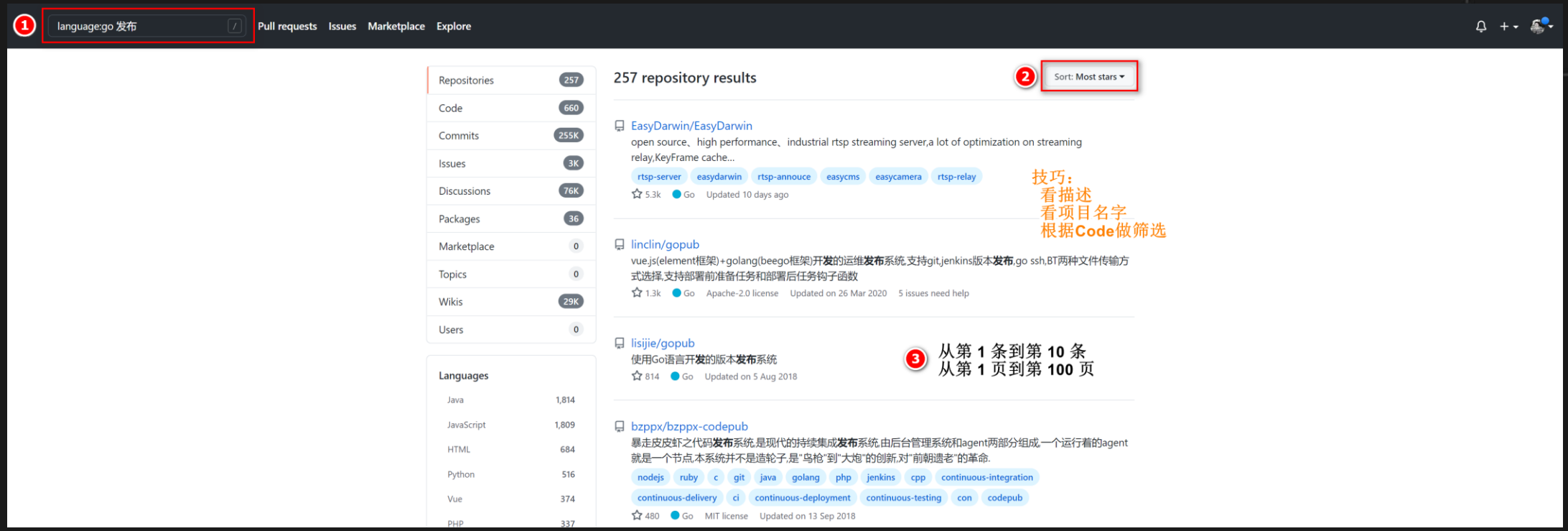
必读，优先读：Go程序设计语言



场景化编程，二选一

Go语言进阶方法（初级）

需求：开发一个版本发布系统



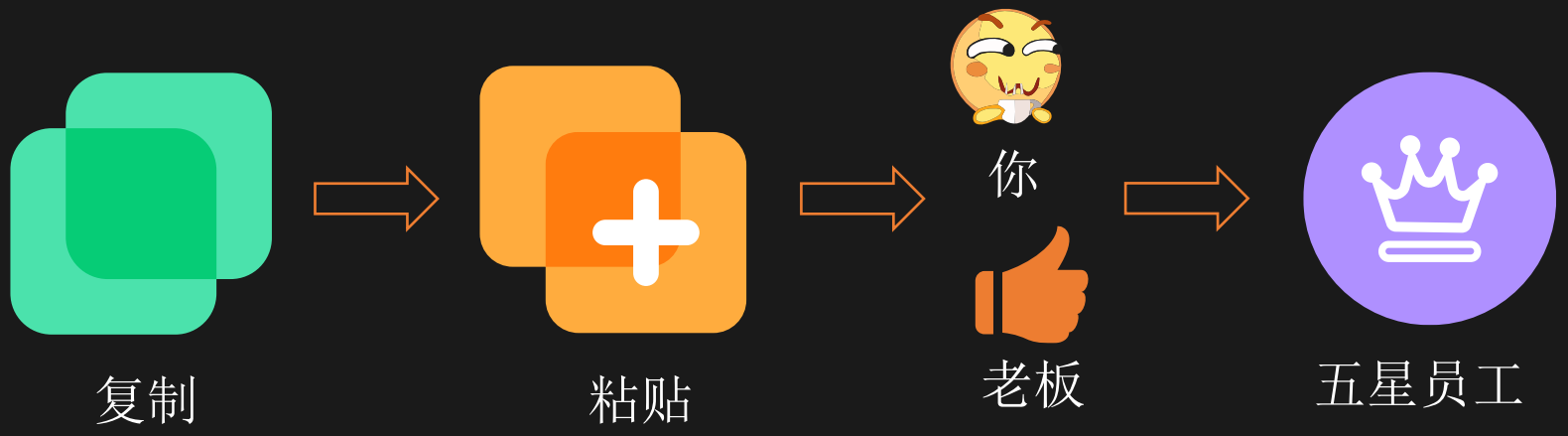
哪里调研Go开源项目：

- https://libs.garden/go
- https://github.com/avelino/awesome-go
- https://golangrepo.com/
- https://go.libhunt.com

Go语言进阶方法（初级）

调研、学习、魔改优秀开源项目很重要！

- 最优解：能够很有底气的跟老板说：这个方案在这个类别就是业界No. 1
- 高效：提高开发效率、提高学习效率
- 产出：知识收获、工作产出
- 知识积累：为今后的开发生涯积累代码库，量变到质变。

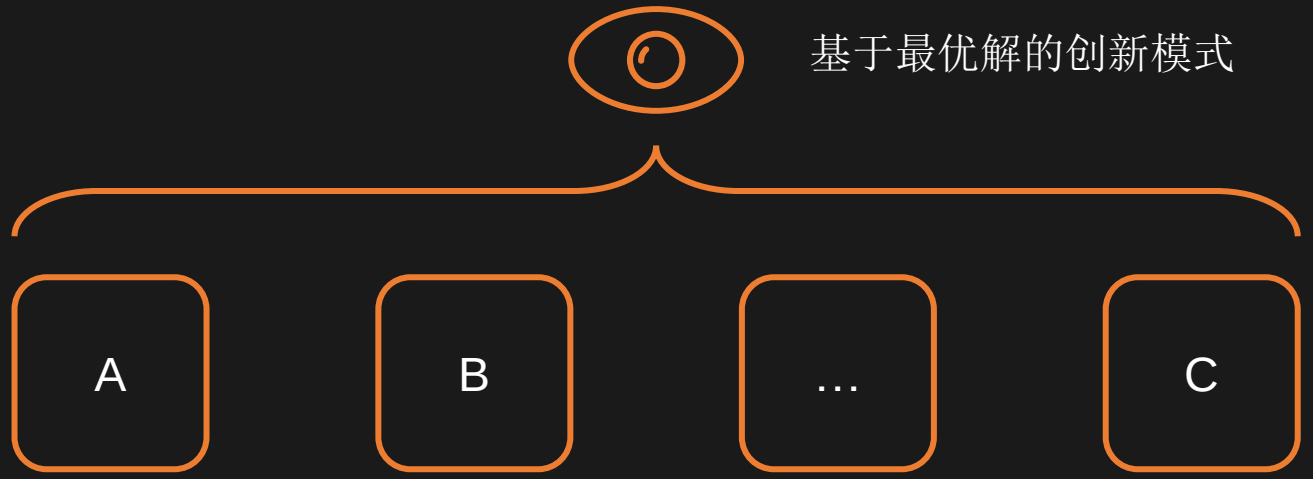


Go语言进阶方法（初级）

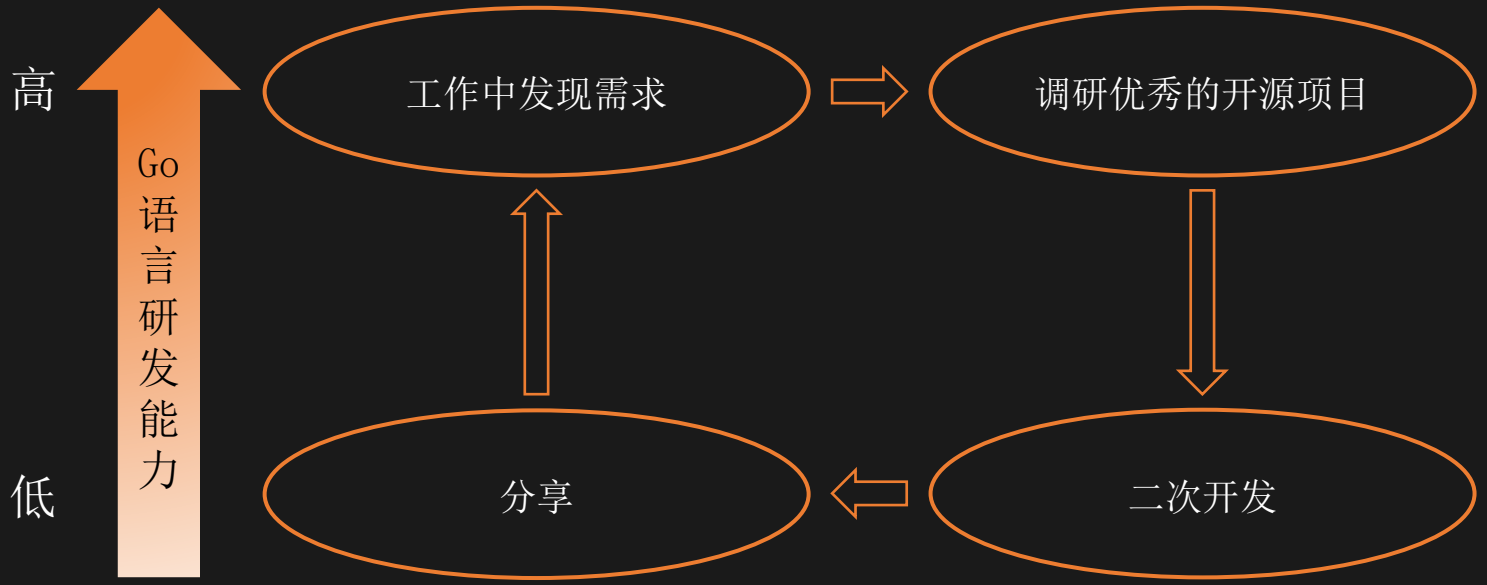
创新

≠

闭门造车



Go语言进阶方法（中级/高级）



在这个过程中需要强化的点：

- 减少对搜索引擎的依赖
- 优先使用Go的高级语法channel、interface等
- 结合面向对象编程的思想
- 注重程序结构和代码质量

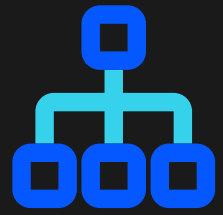
Go语言进阶方法（资深）

精通Go语言：

- 基础语法
- 高级特性
- 重要原理等

学会Go项目开发

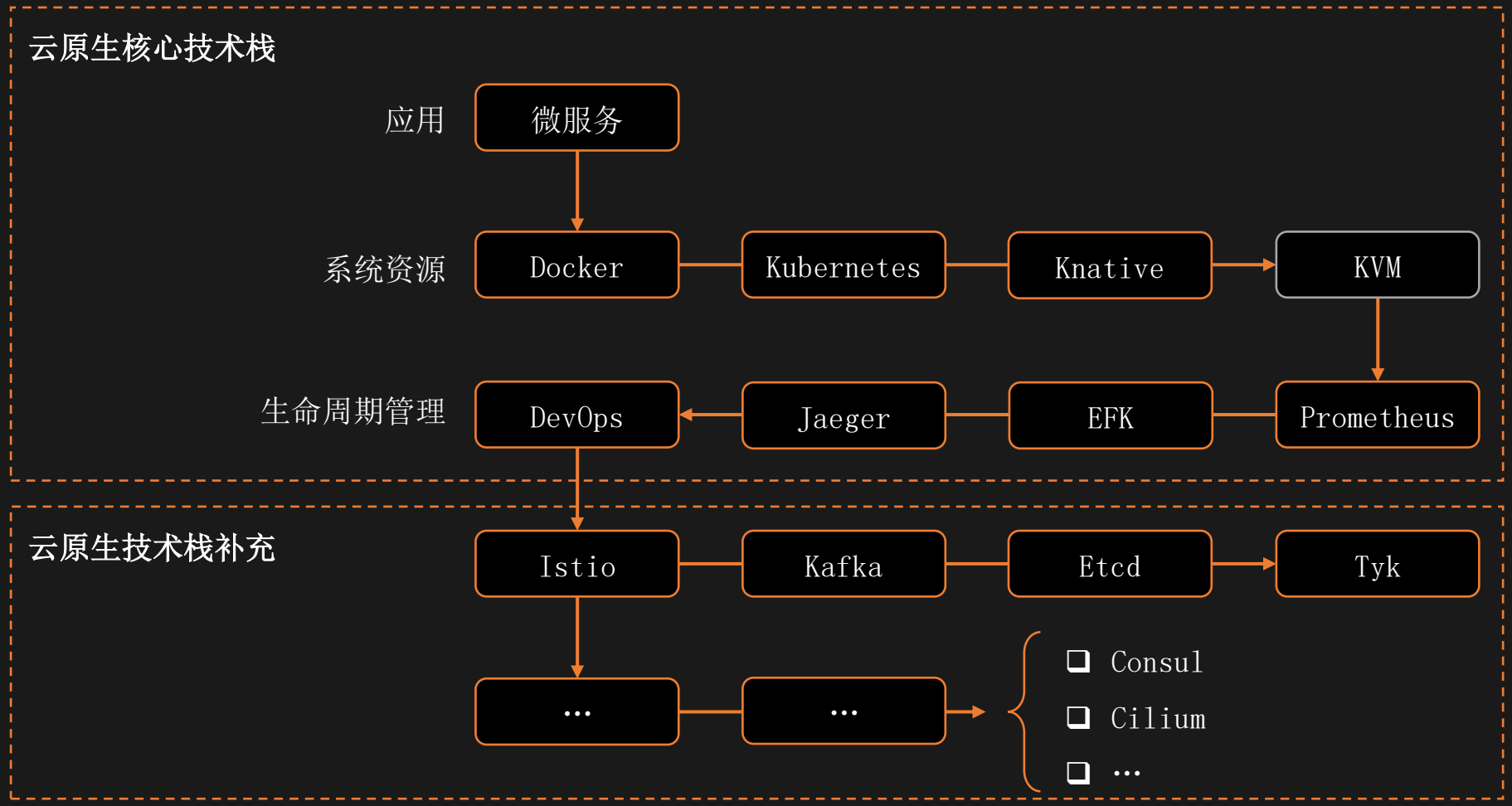
PPT后半部分介绍



建立架构能力：

- 学架构，先从当前业务开始
- 学完当前业务架构，再学云原生架构
- 工作中有架构层面的内容，踊跃参与
- 关注一些优质的架构相关的公众号
- …

Go语言进阶方法（资深）



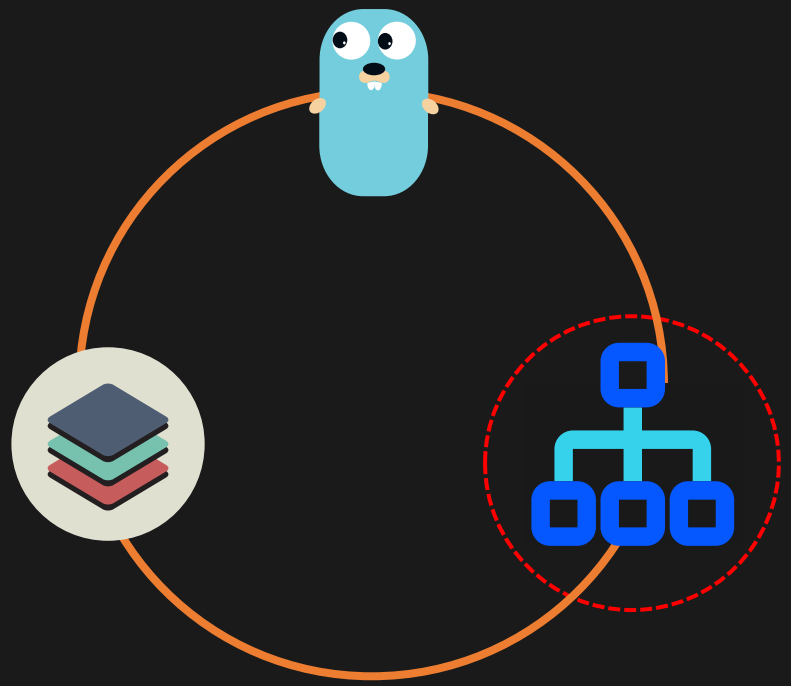
Go语言进阶方法（资深）

高清PDF格式电子书：https://github.com/marmotedu/awesome-books

云原生技术栈阅读材料推荐

类别	书名
微服务	[微服务设计](https://book.douban.com/subject/26772677
Docker	《Docker 技术入门与实战》（第 3 版）、《Docker容器与容器云》（第 2 版）
Kubernetes	Kubernetes权威指南：从Docker到Kubernetes实践全接触（第4版）
Serverless	Knative Documentation
KVM	KVM 虚拟化技术 : 实战与原理解析
监控告警	Prometheus Documentation
调用链	Jaeger Documentation
存储	Etcd
网关	Tyk Open Source
消息队列	Apache Kafka实战
ServiceMesh	云原生服务网格Istio：原理、实践、架构与源码解析
服务发现	Consul Documentation
网络	Cilium Documentation

Go语言进阶方法（专家）



没有天花板、没有标准答案：

- 精通Go语言、项目开发、架构
- Imitator -> Creator
- 把控技术方向、攻克技术难点，解决各种疑难杂症，团队中发挥更大价值

如何开发优秀的Go项目

项目 VS 应用

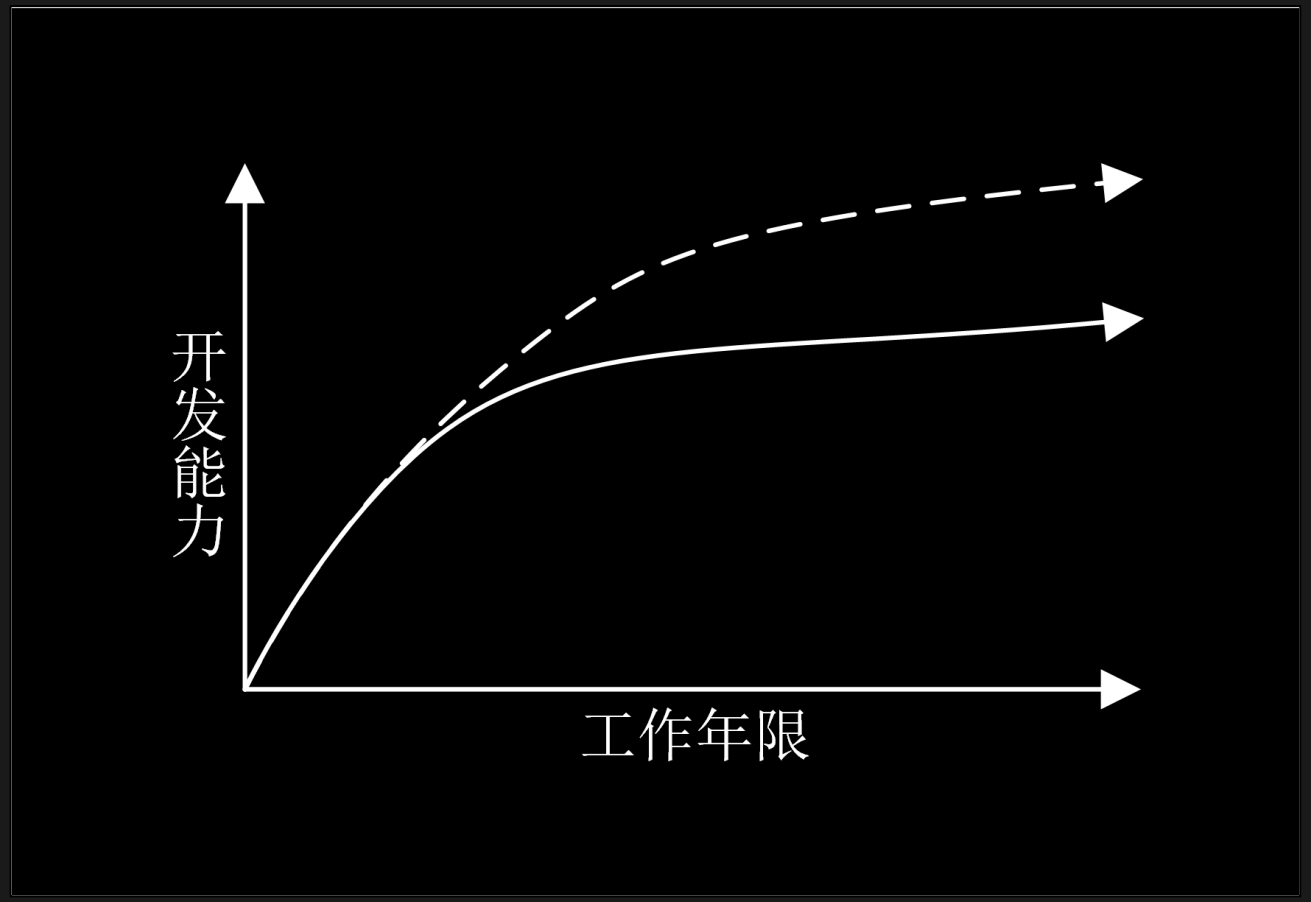
Go 项目 工程化

Go 应用

项目管理

文档

学习Go项目开发的重要性



工作年限和开发能力之间是一个抛物线关系：

- 刚开始，开发能力增长很快
- 到一定年限，开发能力会减缓
- 到一定年限，可能更多是反复使用已有的知识和经验

如何才能提高开发能力的天花板呢？

精研如何构建一个优秀的Go项目。一次学习，终生受益

如何学习Go项目开发？

开发规范

- 代码规范
- 目录规范
- 日志规范
- 错误码规范
- 提交规范
- 版本规范
- 文档规范

项目生命周期管理

- 开发流程
- 代码管理模式
- 开发模式
- DevOps
- …

功能构建

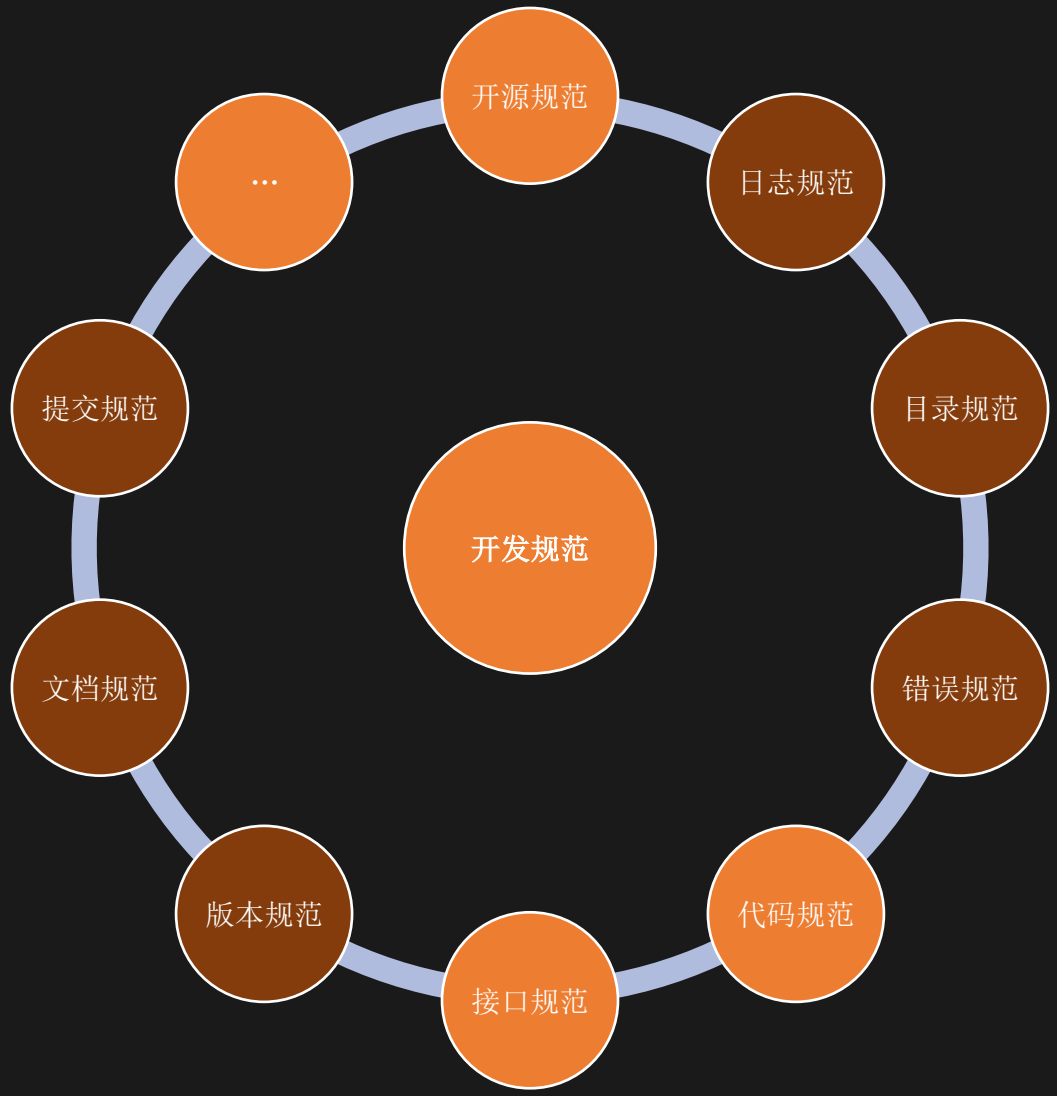
- 应用管理
- 基础功能
- 应用功能
- Web服务
- 应用测试
- 应用部署
- …

技巧/方法

- 模块拆分
- 面向接口编程
- 编写可测代码
- Go设计模式
- Go项目设计哲学
- …

Go项目开发：开发规范

Go项目开发规范有哪些？



开发规范：目录规范

目录结构通常是指我们的代码由哪些目录组成，每个目录下存放什么文件、实现什么功能，以及各个目录间的依赖关系等。

https://github.com/golang-standards/project-layout

```
[colin@dev iam]$ ls
api                 docs          LICENSE     SECURITY.md
build               examples      Makefile    test
CHANGELOG           githooks      _output     third_party
cmd                 go.mod        OWNERS      tools
configs             go.sum        pkg
CONTRIBUTING.md     init          README.md
deployments         internal      scripts
```

如何规范目录？

- ❑ 命名清晰
- ❑ 功能明确
- ❑ 全面性
- ❑ 可扩展性

开发规范：日志规范

日志规范：

- 在何处打印日志？
- 在哪个日志级别打印日志？
- 如何记录日志内容？

在需要的地方记录日志：

- 在分支语句处打印日志
- 写操作处打印日志
- 在循环中打印日志要慎重
- 在错误产生的最原始位置打印日志

在合适的级别记录日志：

- Debug 级别
- Info 级别
- Warn 级别
- Error 级别
- Panic 级别
- Fatal 级别

合理的记录日志：

- 在记录日志时，不要输出一些敏感信息，例如密码、密钥等
- 统一日志记录格式。例如：
  - 成功日志一律为<动词> + <一些事>
  - 失败日志一律为Fail to <动词> + <一些事>
  - 日志内容以大写字母开头，例如 log.Info("Update user function called")
- 根据需要，日志最好包含三个个信息：请求ID（RequestID）、用户（User）、行为（Action）
- 不要将日志记录在错误的日志级别上

开发规范：日志规范

其他建议：

- 开发调试、现网故障排障时，不要遗忘一件事情：根据排障的过程优化日志打印
- 打印日志要“不多不少”，避免打印没有作用的日志，也不要遗漏关键的日志信息
- 支持动态开关 Debug 日志
- 总是将日志记录在本地文件
- 集中化日志存储处理
- 结构化日志记录

开发规范：错误码规范

期望错误码实现的功能：

- 有业务 Code 码标识
- 考虑到安全，希望能够对外对内分别展示不同的错误信息

常见的错误码设计方式：

```
{
  "error": {
     "message": "Syntax error \"Field picture specified more than once. This is only possible before version 2.1\" at character 23: id,name,picture,picture",
     "type": "OAuthException",
     "code": 2500,
     "fbtrace_id": "xxxxxxxxxxx"
  }
}
```

开发规范：错误码规范

期望错误码实现的功能：

- 有业务 Code 码标识
- 考虑到安全，希望能够对外对内分别展示不同的错误信息

常见的错误码设计方式：

```
HTTP/1.1 400 Bad Request
x-connection-hash: xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
set-cookie: guest_id=xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
Date: Thu, 01 Jun 2017 03:04:23 GMT
Content-Length: 62
x-response-time: 5
strict-transport-security: max-age=631138519
Connection: keep-alive
Content-Type: application/json; charset=utf-8
Server: tsa_b

{"errors":[{"code":215,"message":"Bad Authentication data."}]}
```

开发规范：错误码规范

期望错误码实现的功能：

- 有业务 Code 码标识
- 考虑到安全，希望能够对外对内分别展示不同的错误信息

常见的错误码设计方式：

```
HTTP/1.1 400
Date: Thu, 01 Jun 2017 03:40:55 GMT
Content-Length: 276
Connection: keep-alive
Content-Type: application/json; charset=utf-8
Server: Microsoft-IIS/10.0
X-Content-Type-Options: nosniff

{
  "SearchResponse": {
    "Version": "2.2",
    "Query": {
      "SearchTerms": "api error codes"
    },
    "Errors": [
      {
        "Code": 1001,
        "Message": "Required parameter is missing.",
        "Parameter": "SearchRequest.AppId",
        "HelpUrl": "http://msdn.microsoft.com/en-us/library/dd251042.aspx"
      }
    ]
  }
}
```

开发规范：错误码规范

一种错误码设计

```
HTTP/1.1 400
Date: Thu, 01 Jun 2017 03:40:55 GMT
Content-Length: 276
Connection: keep-alive
Content-Type: application/json; charset=utf-8
Server: Microsoft-IIS/10.0
X-Content-Type-Options: nosniff


{
  "code": 100101,
  "message": "Database error",
  "reference": "https://github.com/marmotedu/iam/tree/master/docs/guide/zh-CN/faq/iam-apiserver"
}
```

标号	代表说明
10	服务
01	某个服务下的某个模块
01	模块下的错误码序号，每个模块可以注册100个错误

通过100101可以知道这个错误是服务A，数据库模块下的记录没有找到错误

HTTP Status Code建议

HTTP Status Code	代表说明
200	表示请求成功执行
400	表示客户端出问题
500	表示服务端出问题

HTTP Status Code	代表说明
401	表示认证失败
403	表示授权失败
404	表示资源找不到，这里的资源可以是 URL 或者 RESTful 资源

开发规范：如何返回错误信息

- 在肯定要求时用must，在否定要求时用must not。
  - must be greater than 0
  - must not contain '..'
  - ~~should be greater than 0~~
  - ~~should not contain '..'~~
- 指定不等式时，请使用文字而不是符号。例如：must be less than 256，must be greater than or equal to 0
- 当引用另一个字段名称时，在反引号中指明名称。例如：must be greater than `request`
- 引用文字字符串值时，请在单引号中指明文字。例如：must not contain '..'
- …

开发规范：提交规范

jQuery

Component: Short Description

Ember

[FEATURE name]

AngularJS

<type>(<scope>): <subject>

JSHint

[[TYPE]] <Short description>

Karma

Angular

Conventional Commits

<type>[optional scope]: <description>

```
<type>[optional scope]: <description>
// 空行
[optional body]
// 空行
[optional footer(s)]
```

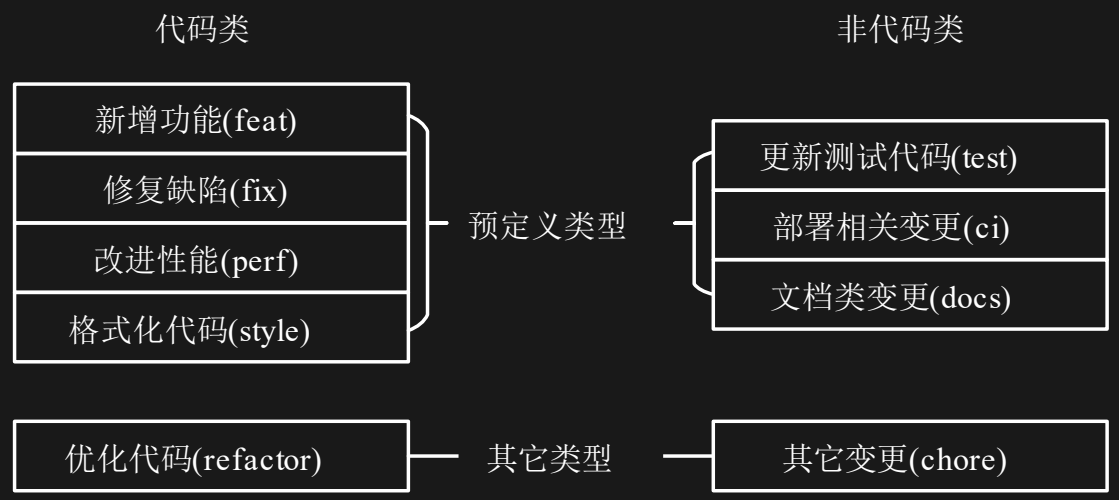
```
fix($compile): couple of unit tests for IE9
# Please enter the Commit Message for your changes. Lines starting
# with '#' will be ignored, and an empty message aborts the commit.
# On branch master
# Changes to be committed:
# ...

Older IEs serialize html uppercased, but IE9 does not...
Would be better to expect case insensitive, unfortunately jasmine does
not allow to user regexps for throw expectations.

Closes #392
Breaks foo.bar api, foo.baz should be used instead
```

开发规范：提交规范

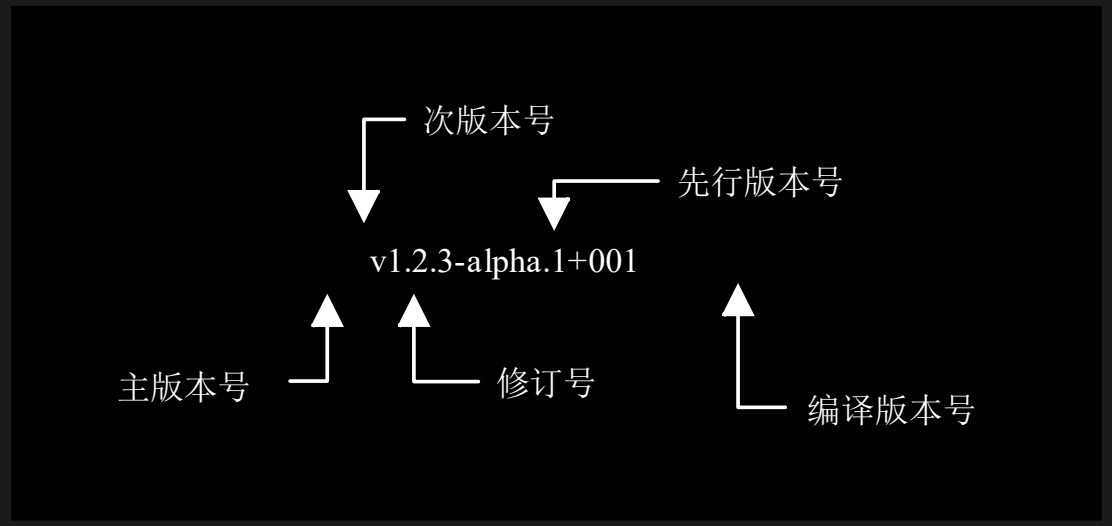
类型	类别	说明
feat	Production	新增功能
fix	Production	Bug 修复
perf	Production	提高代码性能的变更
style	Development	代码格式类的变更，比如用 gofmt 格式化代码、删除空行等
refactor	Production	其他代码类的变更，这些变更不属于 feat、fix、perf 和 style，例如简化代码、重命名变量、删除冗余代码等
test	Development	新增测试用例或是更新现有测试用例
ci	Development	持续集成和部署相关的改动，比如修改 Jenkins、GitLab CI 等 CI 配置文件或者更新 systemd unit 文件
docs	Development	文档类的更新，包括修改用户文档或者开发文档等
chore	Development	其他类型，比如构建流程、依赖管理或者辅助工具的变动等



开发规范：版本规范

语义化版本规范(SemVer)

语义化版本格式为：主版本号.次版本号.修订号（X.Y.Z），其中 X、Y 和 Z 为非负的整数，且禁止在数字前方补零。



示例：

1.0.0-alpha+001

1.0.0+20130313144700

1.0.0-beta+exp.sha.5114f85

版本号可按以下规则递增：

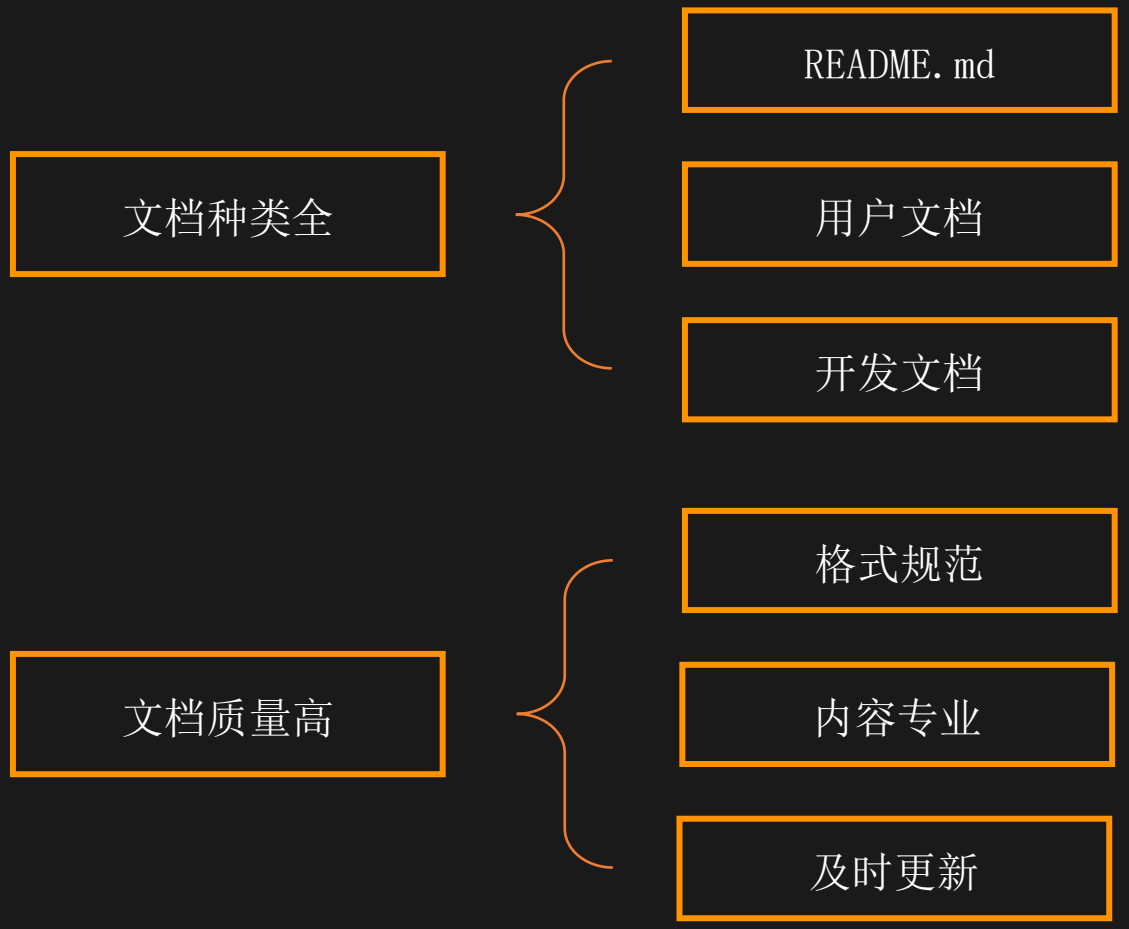
- 主版本号（MAJOR）：当做了不兼容的 API 修改。
- 次版本号（MINOR）：当做了向下兼容的功能性新增及修改。这里有个不成文的约定需要你注意，偶数为稳定版本，奇数为开发版本。
- 修订号（PATCH）：当做了向下兼容的问题修正。
- 先行版本号：意味着该版本不稳定，可能存在兼容性问题，格式为：X.Y.Z-[一连串以句点分隔的标识符]
- 编译版本号：一般是编译器在编译过程中自动生成的，我们只定义其格式，并不进行人为控制。

开发规范：版本规范

如何确定版本号？

- 在实际开发的时候，建议使用 0.1.0 作为第一个开发版本号，并在后续的每次发行时递增次版本号。
- 当我们的版本是一个稳定的版本，并且第一次对外发布时，版本号可以定为 1.0.0。
- 当我们严格按照 Angular commit message 规范提交代码时，版本号可以这么来确定：
  - fix 类型的 commit 可以将修订号 +1。
  - feat 类型的 commit 可以将次版本号 +1。
  - 带有 BREAKING CHANGE 的 commit 可以将主版本号 +1。

开发规范：文档规范



- API文档
- SDK文档
- 安装文档
- 功能介绍文档
- 操作指南
- …

Go项目开发：生命周期管理

如何管理Go项目生命周期？

开发流程

代码管理模式

- 集中式工作流
- 功能分支工作流
- Gitflow工作流
- Forking工作流

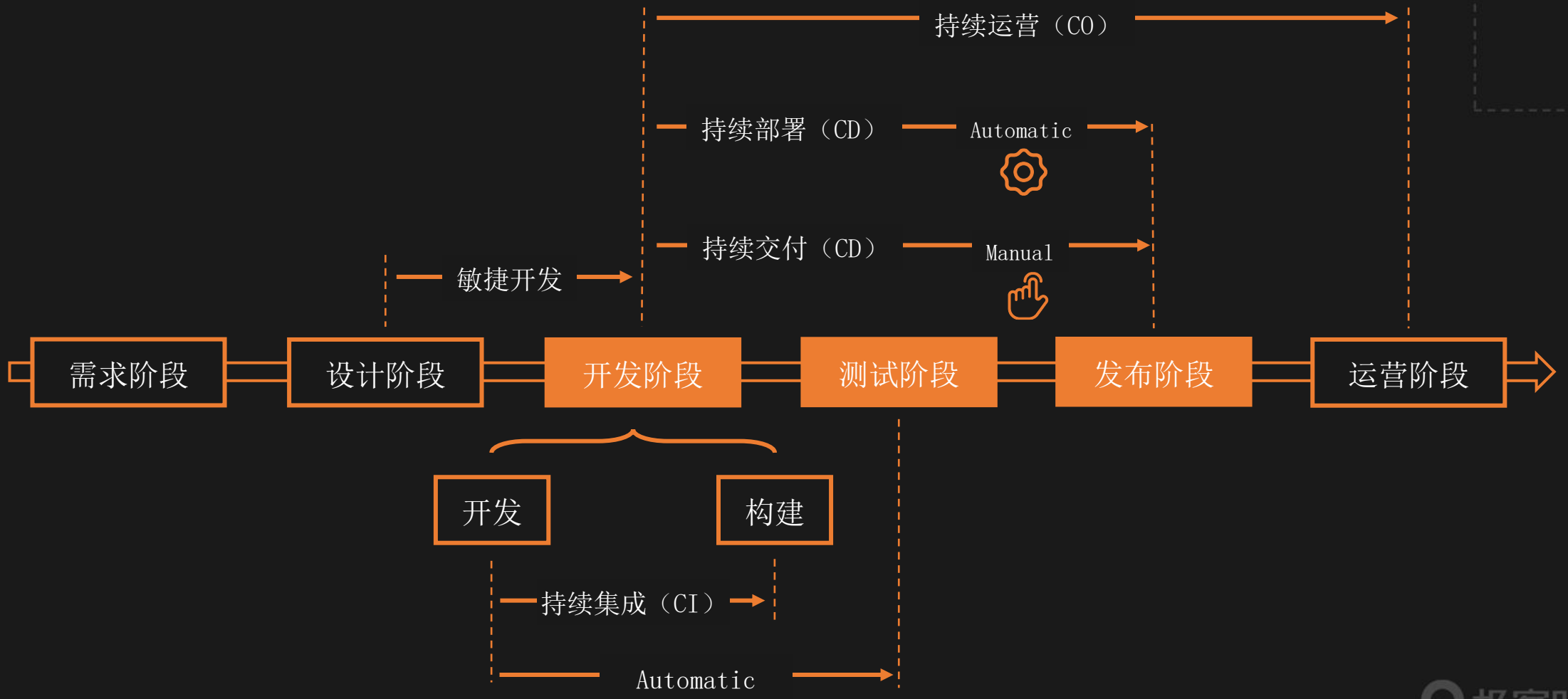
开发模式

- 瀑布模式
- 迭代模式
- 敏捷模式

DevOps

- CICD
- AIOps
- ChatOps
- GitOps
- NoOps

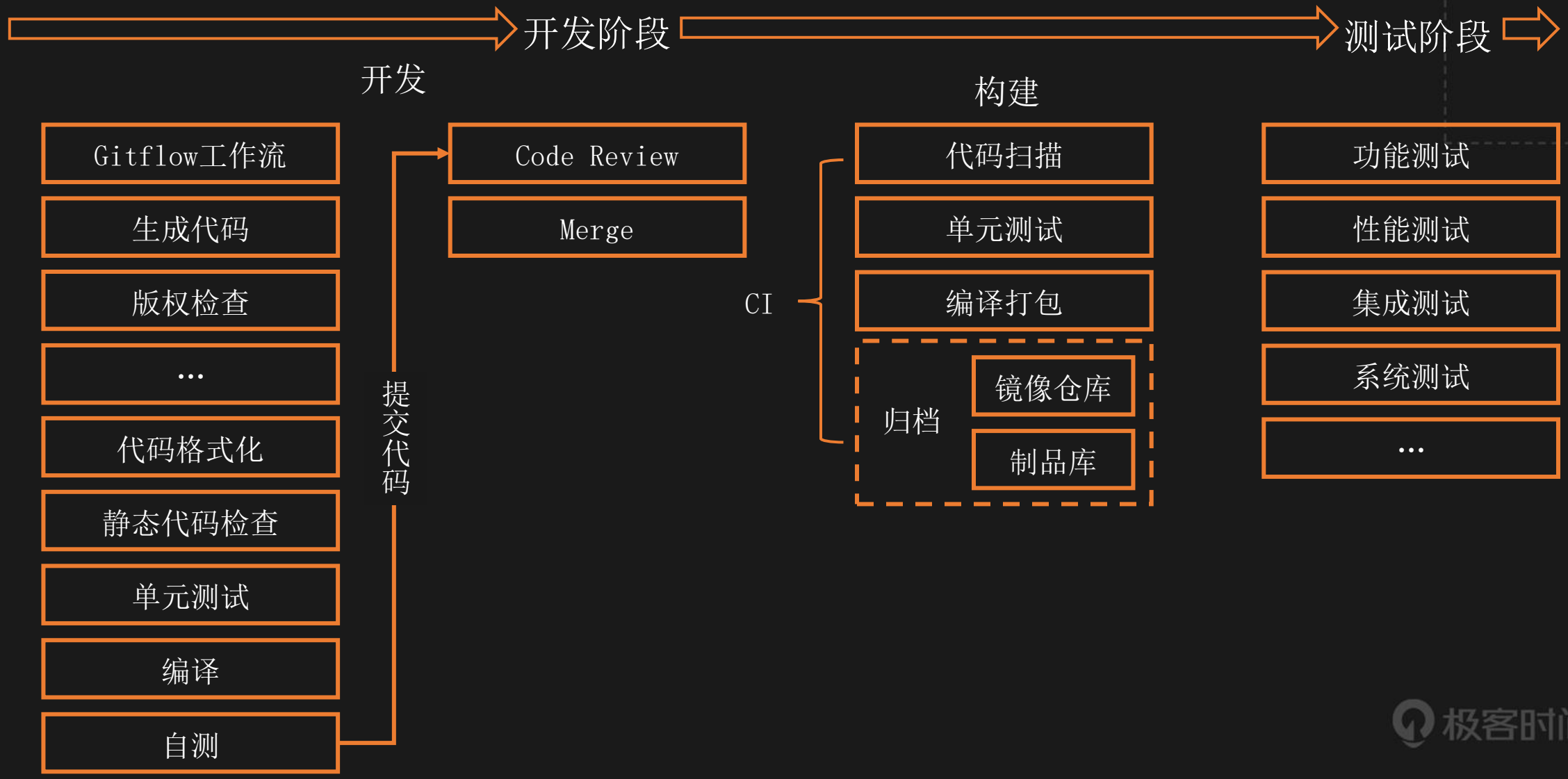
Go项目开发流程



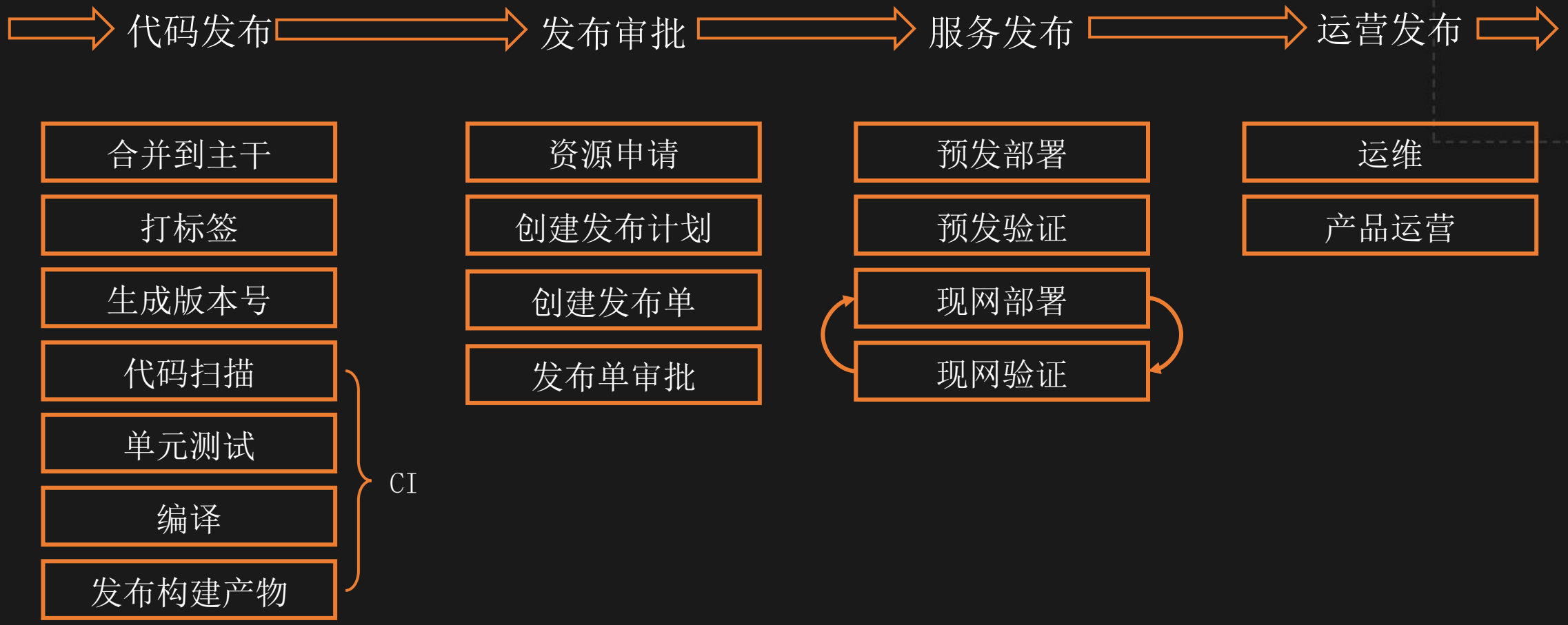
开发流程：设计阶段

产品设计	由产品人员负责，设计出产品的形态、功能和交互，并输出详细的产品设计文档
交互设计	由交互设计师负责，参照产品设计文档，将产品用原型图和交互流程的形式展现出来
视觉设计	根据需求文档和交互设计原型设计出产品视觉界面
技术设计	技术设计，包括技术架构、实现方法等
技术评审	所有的技术设计，都需要经过技术评审，评审通过后，才能进行实际的开发
需求排期	项目经理会组织产研人员对需求进行细化和排期，需求排期尽可能细分，这样才能更好的评估工作量和风险

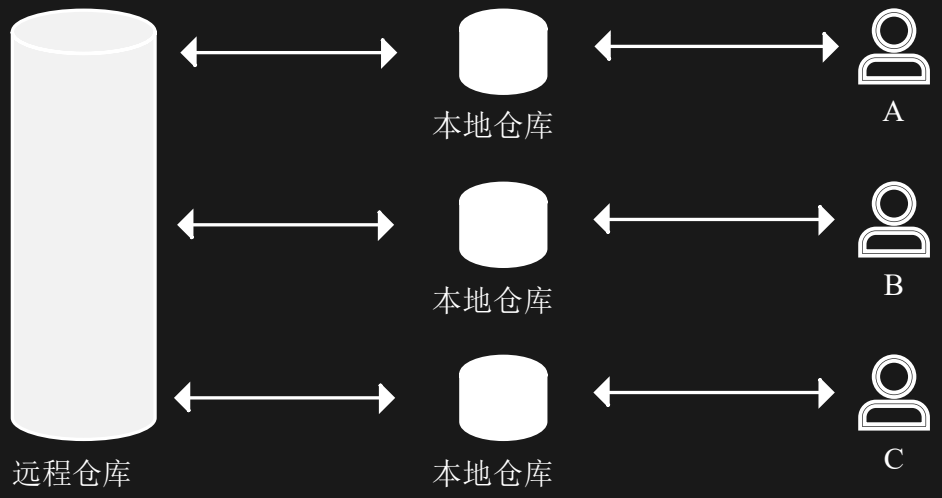
开发流程：开发 & 测试阶段



开发流程：发布 & 运营阶段



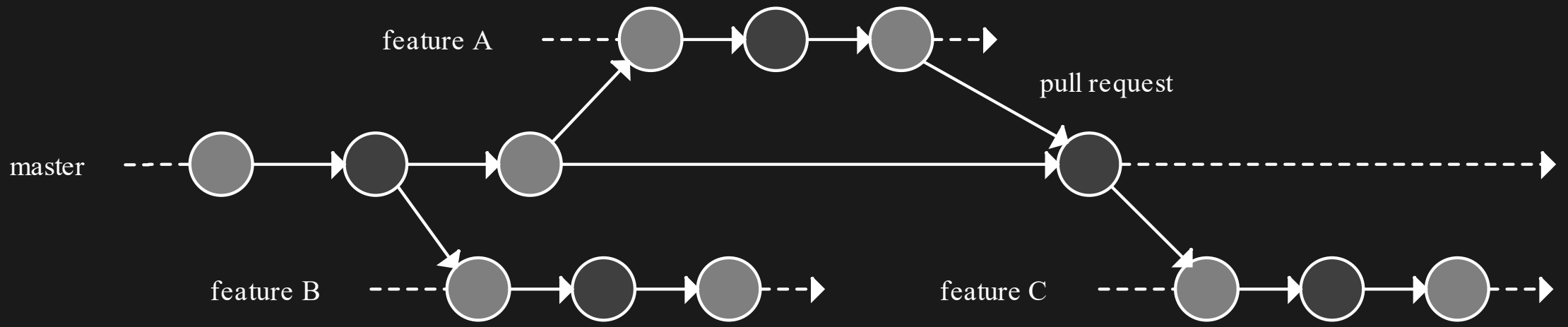
代码管理模式：集中式工作流



A、B、C 为 3 位开发者，每位开发者都在本地有一份远程仓库的拷贝：本地仓库。A、B、C 在本地的 master 分支开发完代码之后，将修改后的代码 commit 到远程仓库，如果有冲突就先解决本地的冲突再提交。

一段时间的开发之后，远程仓库 master 分支的日志

代码管理模式：功能分支工作流



功能分支工作流基于集中式工作流演进而来。在开发新功能时，基于 master 分支新建一个功能分支，在功能分支上进行开发，而不是直接在本地的 master 分支开发，开发完成之后合并到 master 分支

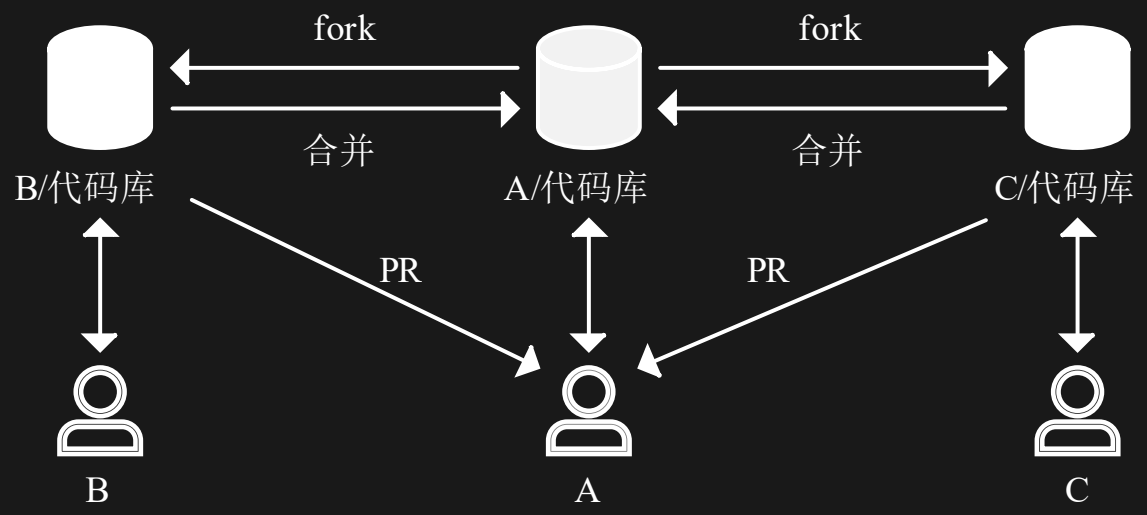
代码管理模式：GitFlow工作流

GitFlow 工作流是一个非常成熟的方案，也是非开源项目中最常用到的工作流。它定义了一个围绕项目发布的严格分支模型，通过为代码开发、发布和维护分配独立的分支来让项目的迭代流程更加顺畅，比较适合大型的项目或者迭代速度快的项目

5 种分支：

分支名	描述
master	该分支上的最新代码永远是发布状态，不能直接在该分支上开发。master 分支每合并一个 hotfix/release 分支，都会打一个版本标签
develop	该分支上的代码是开发中的最新代码，该分支只做合并操作，不能直接在该分支上开发
feature	在研发阶段用来做功能开发。一个新功能会基于 develop 分支新建一个 feature 分支，分支名建议命名为：feature/xxxx-xxx。功能开发完成之后，会合并到 develop 分支并删除。这里有一点需要你注意，feature 分支在申请合并之前，最好是先 pull 一下 develop 分支，看一下有没有冲突，如果有就先解决冲突后再申请合并
release	在发布阶段用作版本发布的预发布分支，基于 develop 分支创建，分支名建议命名为：release/xxxx-xxx。例如：v1.0.0 版本的功能全部开发测试完成后，提交到 develop 分支，然后基于 develop 分支创建 release/1.0.0 分支，并提交测试，测试中遇到的问题在 release 分支修改。最终通过测试后，将 release 分支合并到 master 和 develop，并在 master 分支打上 v1.0.0 的版本标签，最后删除 release/1.0.0 分支
hotfix	在维护阶段用作紧急 bug 修复分支，在 master 分支上创建，修复完成后合并到 master。分支名建议命名为hotfix/xxxx-xxx。例如：当线上某个版本出现 Bug 后，从 master 检出对应版本的代码，创建 hotfix 分支，并在 hotfix 分支修复问题。问题修复后，将 hotfix 分支合并到 master 和 develop分支，并在 master 分支打上修复后的版本标签，最后删除 hotfix 分支

代码管理模式：Forking工作流



开发者需要先fork目标仓库，然后基于fork的仓库开发代码，并将代码变更push到fork的仓库。最后通过提交Pull Request的方式，将fork仓库的代码变更，应用到目标仓库。

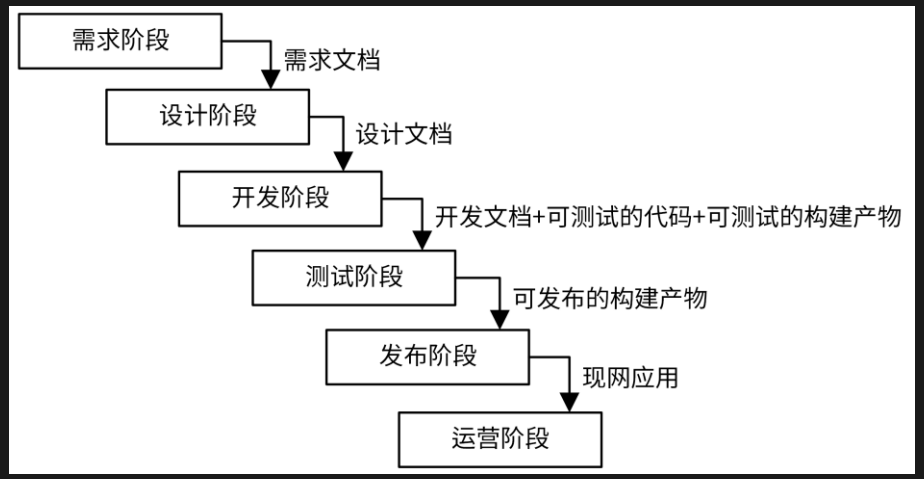
Forking 工作流中，项目远程仓库和开发者远程仓库完全独立，开发者通过提交 Pull Request 的方式给远程仓库贡献代码，项目维护者选择性地接受任何开发者的提交，通过这种方式，可以避免授予开发者远程仓库的权限，从而提高项目远程仓库的安全性，这也使得任意开发者都可以参与项目的开发。

如何选择代码管理模式？

工作流	优点	缺点	使用场景
集中式工作流	上手最简单	代码管理较混乱，容易出问题	团队人数少，开发不频繁，不需要同时维护多个版本的小项目
功能分支工作流	上手比较简单，支持并行开发，支持Code Review	无法给分支分配明确的目的，不利于团队配合	开发团队相对固定、规模较小的项目
Git Flow工作流	每个分支分工明确，这可以最大程度减少它们之间的相互影响，可以并行开发，支持Code Review	一定的上手难度	比较适合开发团队相对固定，规模较大的项目
Forking工作流	完全解耦个人远端仓库和项目远端仓库，最大程度上保证远端仓库的安全	对于职能分工明确且不对外开源的项目优势不大	比较适用于开源项目中，或者开发者有衍生出自己的衍生版的需求，或者开发者不固定，可能是任意一个能访问到项目的开发者

- 非开源项目采用 Git Flow 工作流
- 开源项目采用 Forking 工作流

开发模式：瀑布模式



瀑布模式：

按照预先规划好的研发阶段来串行推进研发进度。每个阶段完美完成之后，才会进入到下一阶段，阶段之间通过文档进行交付。

优点：

- 简单，严格按照研发阶段来推进研发进度，流程清晰

缺点：

- 后期变更代价大
- 研发周期比较长，很难适应互联网时代对产品快速迭代的诉求

开发模式：迭代模式

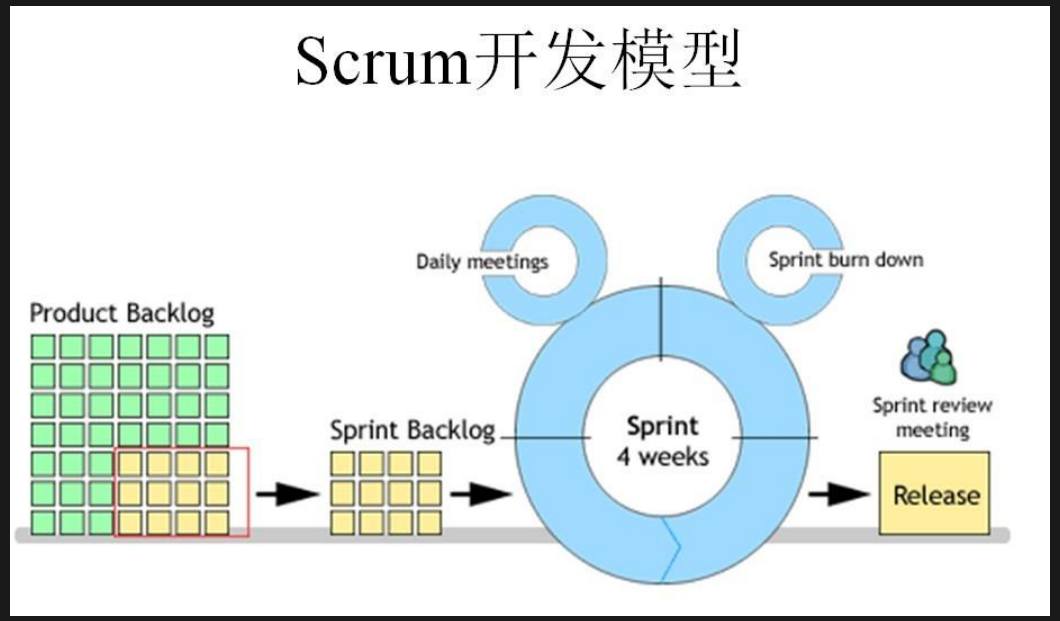
迭代模式：

是一种与瀑布式模式完全相反的开发过程：研发任务被切分为一系列轮次，每一个轮次都是一个迭代，每一次迭代都是一个从设计到实现的完整过程。它不要求每一个阶段的任务都做到最完美，而是先把主要功能搭建起来，然后再通过客户的反馈信息不断完善。

优点：迭代开发可以帮助产品改进和把控进度，它的灵活性极大地提升了适应需求变化的能力，克服了高风险、难变更、复用性低的特点。

缺点：比较专注于开发过程，很少从项目管理的视角去加速和优化项目开发过程

开发模式：敏捷模式



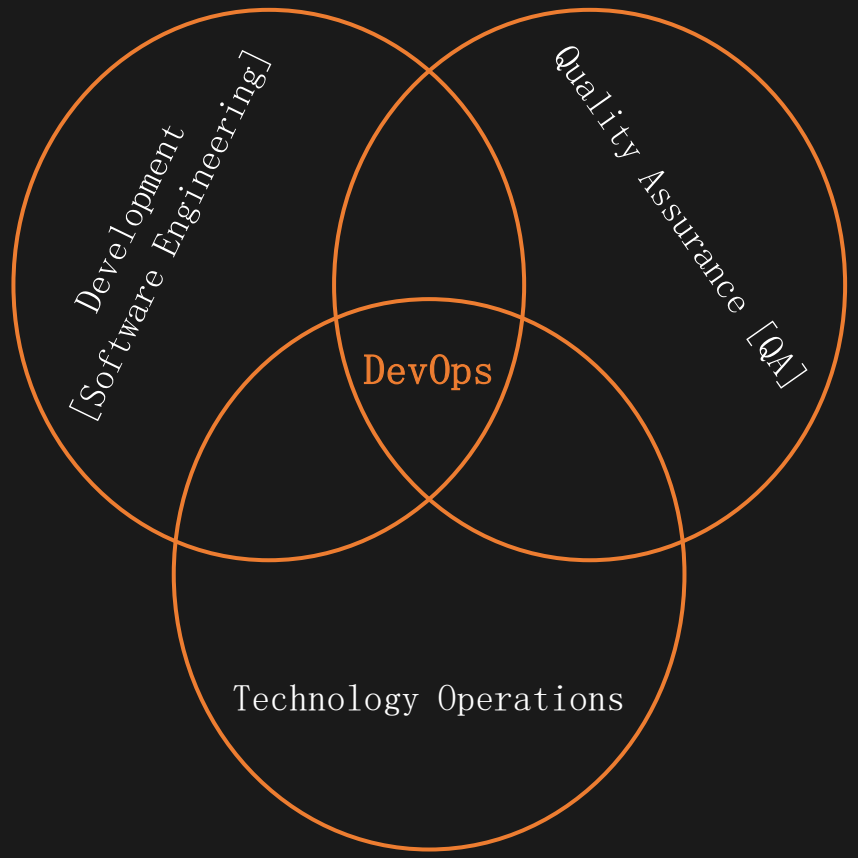
敏捷模式：

把一个大的需求分成多个、可分阶段完成的小迭代，每个迭代交付的都是一个可使用的软件。在开发过程中，软件要一直处于可使用状态。

- 快速迭代，拥抱变化
- 软件快速上线、更新，适应互联网时代

DevOps：什么是DevOps

DevOps: Development & Operations （研发运维一体化）



- 是一组过程、方法与系统的统称，用于促进开发、技术运营和质量保障（QA）部门之间的沟通、协作与整合
- 目标：提高软件质量、快速发布软件

要实现 DevOps，需要一些工具或者流程的支持，CI/CD 可以很好地支持 DevOps 这种软件开发模式，如果没有 CI/CD 自动化的工具和流程，DevOps 就是没有意义的。

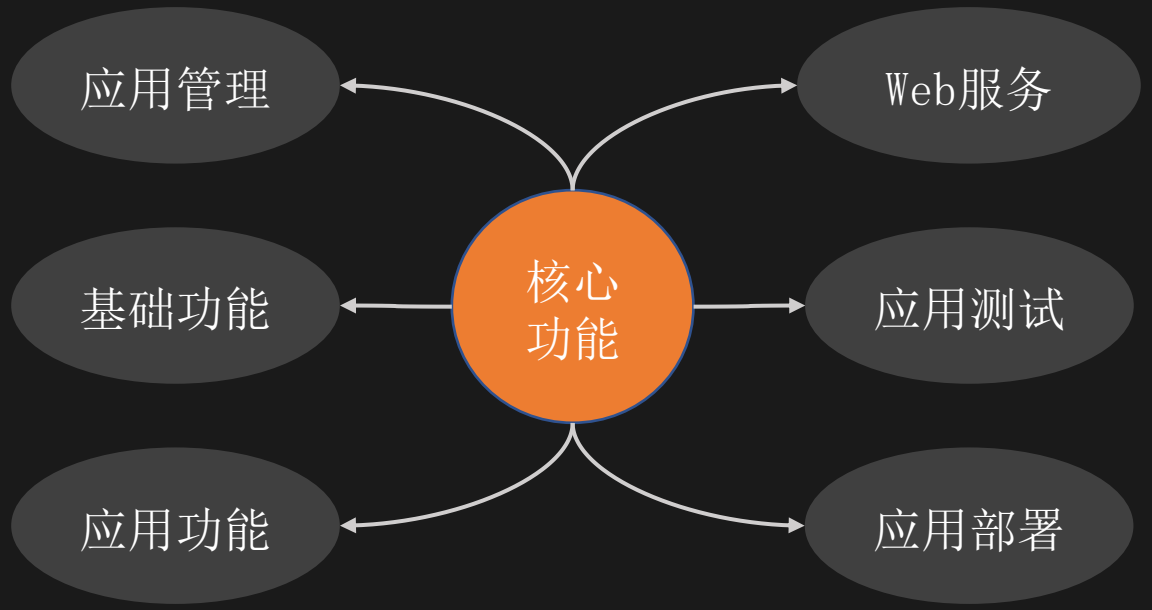
DevOps ！= CI/CD。DevOps 是一组过程、方法和系统的统称，而 CI/CD 只是一种软件构建和发布的技术

DevOps：Ops

- AIOps：智能运维
- ChatOps：聊着天就把事情给办了
- GitOps：一种实现云原生的持续交付模型
- NoOps：无运维

Go项目开发：功能构建

Go项目开发核心功能设计



应用管理

Makefile

- 代码格式化
- 镜像制作
- 版权声明
- 代码部署
- 生成Swagger文档
- 工具安装
- 编译
- 单元测试
- 代码生成
- CA证书制作
- 静态代码检查
- …

基础功能

日志包设计	主要功能：排障、数据分析。注重功能、性能、规范、日志告警
错误包设计	在大型项目中，需要和业务码适配，并具有trace功能
错误码设计	需要考虑到业务码、HTTP Status Code、内外Message、Reference
应用框架构建	框架的通用性、易用性。通常包括：命令行程序、配置文件解析、命令行参数解析
数据库	稳定性、易用性、功能性
…	…

应用功能

API 服务

- 以 RESTful 或者 RPC 接口的方式对外提供服务
- 管控流通常为 RESTful 接口
- 数据流通常为 RPC 接口

Go SDK

- 通常包含了跟软件相关的库、文档、使用示例、封装好的 API 接口调用和工具
- 可以极大的提高开发者的开发效率和体验
- 对于一个企业级的项目，最好提供 SDK

命令行工具

- 一般为xxxctl，例如：kubectl、etcdctl、istiotl
- 实现自动化、提高调用效率

分布式作业

- 异步任务处理

异步数据处理

- 通常用于数据分析、监控告警、运营数据展示等。
- 对采集效率要求很高
- 分为同步采集和异步采集

Web服务

RESTful	gRPC	参数解析
JSON	Protobuf	参数校验
路由	中间件	逻辑处理
RequestID	跨域	返回结果
HTTP/HTTPS	优雅关停	
认证/授权	…	

应用测试

单元测试

性能分析

测试框架

Testify

GoConvey

性能测试

覆盖率

Mock工具

sqlmock

httpmock

bouk/monkey

golang/mock

...

应用部署

裸金属部署

容器化部署

服务编排

CICD

高可用

负载均衡

弹性伸缩

安全

...

Go项目开发：技巧/方法

如何写出优雅的Go项目？

如何写出优雅的Go项目？

Go 项目 工程化

Go 应用

项目管理

文档

高质量的Go代码

高效的项目管理

高质量的文档

易阅读、可维护、可扩展的高质量 Go 项目

高质量的Go代码

代码结构	代码规范	编程哲学		代码质量	设计模式
目录结构	编码规范	面向接口编程	单元测试	编写可测试的代码	设计模式
按功能拆分模块	最佳实践	面相组合编程		高单元测试覆盖率	遵循SOLID原则
				Code Review	

按功能拆分模块

按层拆分

```
$ tree --noreport -L 2 layers
layers
├── controllers
│   ├── billing
│   ├── order
│   └── user
├── models
│   ├── billing.go
│   ├── order.go
│   └── user.go
└── views
    └── layouts
```

缺点：功能共享差、循环引用

按功能拆分

```
$ tree pkg
$ tree --noreport -L 2 pkg
pkg
├── billing
├── order
│   └── order.go
└── user
```

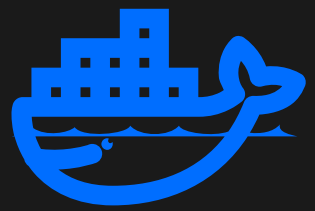
优点：

- 不同模块，功能单一，可实现高内聚低耦合的设计哲学
- 大大减少出现循环引用的概率

面相接口编程

Go 接口是一组方法的集合，任何类型，只要实现了该接口中方法集，那么就属于这个类型，也称为实现了该接口。

Kubernetes



Docker

Prometheus

接口是衡量代码质量高低的一个核心指标

面向接口编程

```go
package main

import "fmt"

// 定义了一个鸟类
type Bird interface {
    Fly()
    Type() string
}

// 让鸟类飞一下
func LetItFly(bird Bird) {
    fmt.Printf("Let %s Fly!\n", bird.Type())
    bird.Fly()
}

func main() {
    LetItFly(&Canary{"金丝雀"})
    LetItFly(&Crow{"乌鸦"})
}
```

```go
package main

import "fmt"

// 鸟类：金丝雀
type Canary struct {
    Name string
}

func (c *Canary) Fly() {
    fmt.Printf("我是%s，用黄色的翅膀飞\n", c.Name)
}
func (c *Canary) Type() string {
    return c.Name
}

// 鸟类：乌鸦
type Crow struct {
    Name string
}

func (c *Crow) Fly() {
    fmt.Printf("我是%s，我用黑色的翅膀飞\n", c.Name)
}

func (c *Crow) Type() string {
    return c.Name
}
```

面向接口编程

面向接口编程优点：

- 使代码扩展性更强
- 解耦上下游实现，不用关心具体实现细节
- 提高代码的可测性
- 使代码更健壮、稳定

编写可测试的代码

不可测试

```go
package testable

import "gorm.io/gorm"

type User struct {
    Name  string
    Phone string
}

// CreateUser 不可测试
func CreateUser(db *gorm.DB, user *User) error {
    return db.Create(user).Error
}
```

可测试

```go
package testable

type UserStore interface {
    Create(user *User) error
    Get(name string) (*User, error)
    List() ([]*User, error)
    Delete(name string) error
}

type User struct {
    Name    string
    Address string
}

func CreateUser(ds UserStore, user *User) error {
    return ds.Create(user).Error
}
```

编写可测试的代码

```go
package testable

type fake struct {
}

func NewFake() *fake {
    return &fake{}
}

func (f *fake) Create(user *User) error {
    return nil
}

func (f *fake) Get(name string) (*User, error) {
    return &User{"colin", "shenzhen"}
}

func (f *fake) List() ([]*User, error) {
    return []*User{&User{"colin", "shenzhen"}}, nil
}

func (f *fake) Delete(name string) error {
    return nil
}
```

```go
package testable

import (
    "testing"

    "github.com/stretchr/testify/assert"
)

func TestCreateUser(t *testing.T) {
    fake := NewFake()

    assert.Nil(t, fake.Create(&User{"colin", "shenzhen"}))
}
```

高单元测试覆盖率

如何提高单元测试覆盖率：

- 编写可测试的代码
- 借助工具：gotests、golang/mock、sqlmock、httpmock、bouk/monkey
- 加入 Makefile，集成进 CI 流程，设置质量红线

设计模式


创建型模式
行为型模式
结构型模式
单例模式
简单工厂模式
抽象工厂模式
工厂方法模式
建造者模式
原型模式
工厂模式
访问者模式
策略模式
模板模式
状态模式
观察者模式
备忘录模式
中介者模式
迭代器模式
解释器模式
命令模式
责任链模式
适配器模式
桥接模式
组合模式
装饰模式
外观模式
享元模式
代理模式
选项模式

遵循SOLID原则

简写	中文描述	介绍
SRP	单一功能原则	一个类或者模块只负责完成一个职责(或者功能)
OCP	开闭原则	软件实体应该对扩展开放、对修改关闭
LSP	里氏替换原则	如果S是T的子类型，则类型T的对象可以替换为类型S的对象，而不会破坏程序
DIP	依赖倒置原则	依赖于抽象而不是一个实例，其本质是要面向接口编程，不要面向实现编程
ISP	接口分离原则	客户端程序不应该依赖它不需要的方法

高效的项目管理

高效的项目管理

工具名称	功能
golangci-lint	静态代码检查工具
swagger	自动生成Go Swagger文档
gotests	根据Go代码自动生成单元测试模板
gsemver	根据git commit规范自动生成语义化版本
golines	格式化Go代码中的长行为短行
git-chglog	根据git commit自动生成CHANGELOG
go-mod-outdated	检查依赖包是否有更新
db2struct	将数据库表一键转换为go struct
rts	用于根据服务器的响应生成 Go 结构体
goimports	自动格式化 Go 代码并对所有引入的包进行管理

如何写出优雅的Go项目?

算法

一个优秀的Go实战项目

IAM 项目背景

过去学习Go的过程中，遇到的3个核心困难：

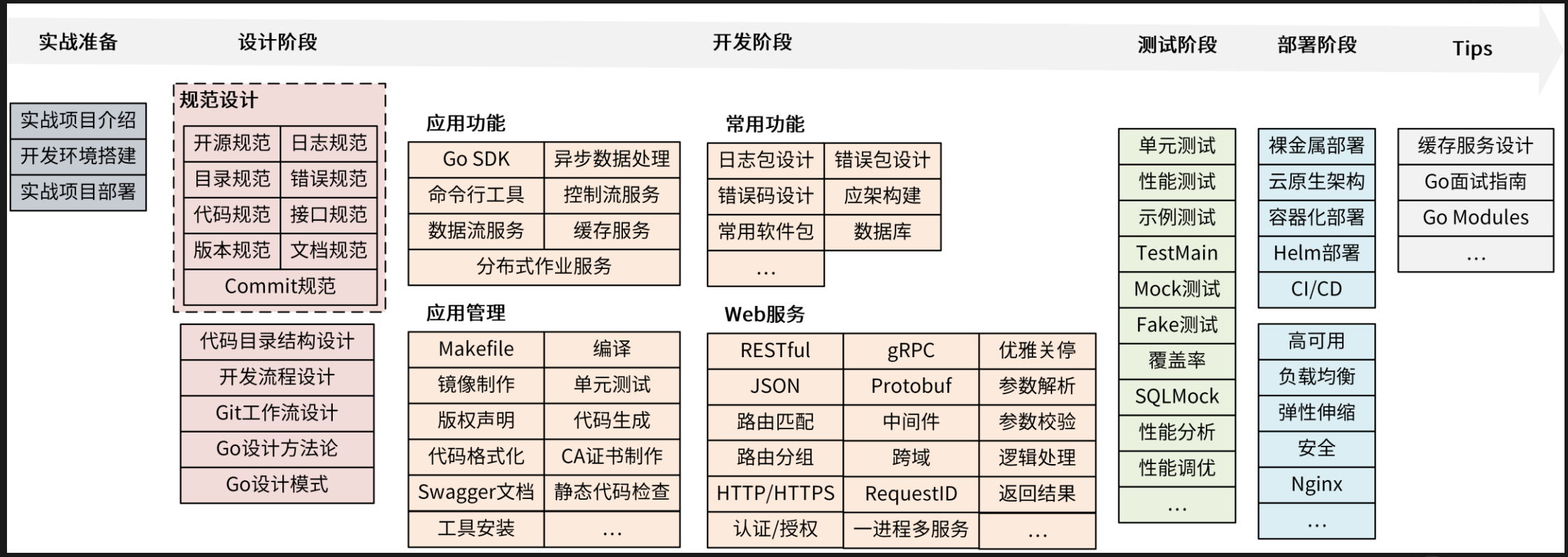
- 学不到最佳实践，能力提升有限：网上有很多介绍 Go 项目的构建方法，但大多并非最佳实践，学完也无法带来很大的提升；
- 不懂如何从 0 到 1 开发一个 Go 项目：对 Go 开发的知识点和构建方法，掌握不够全面和深入，无法建立完整的 Go 项目研发体系。
- 很多时候需要参考一个优秀的项目来学习、验证，但是发现缺少这样一个项目。

https://github.com/marmotedu/iam

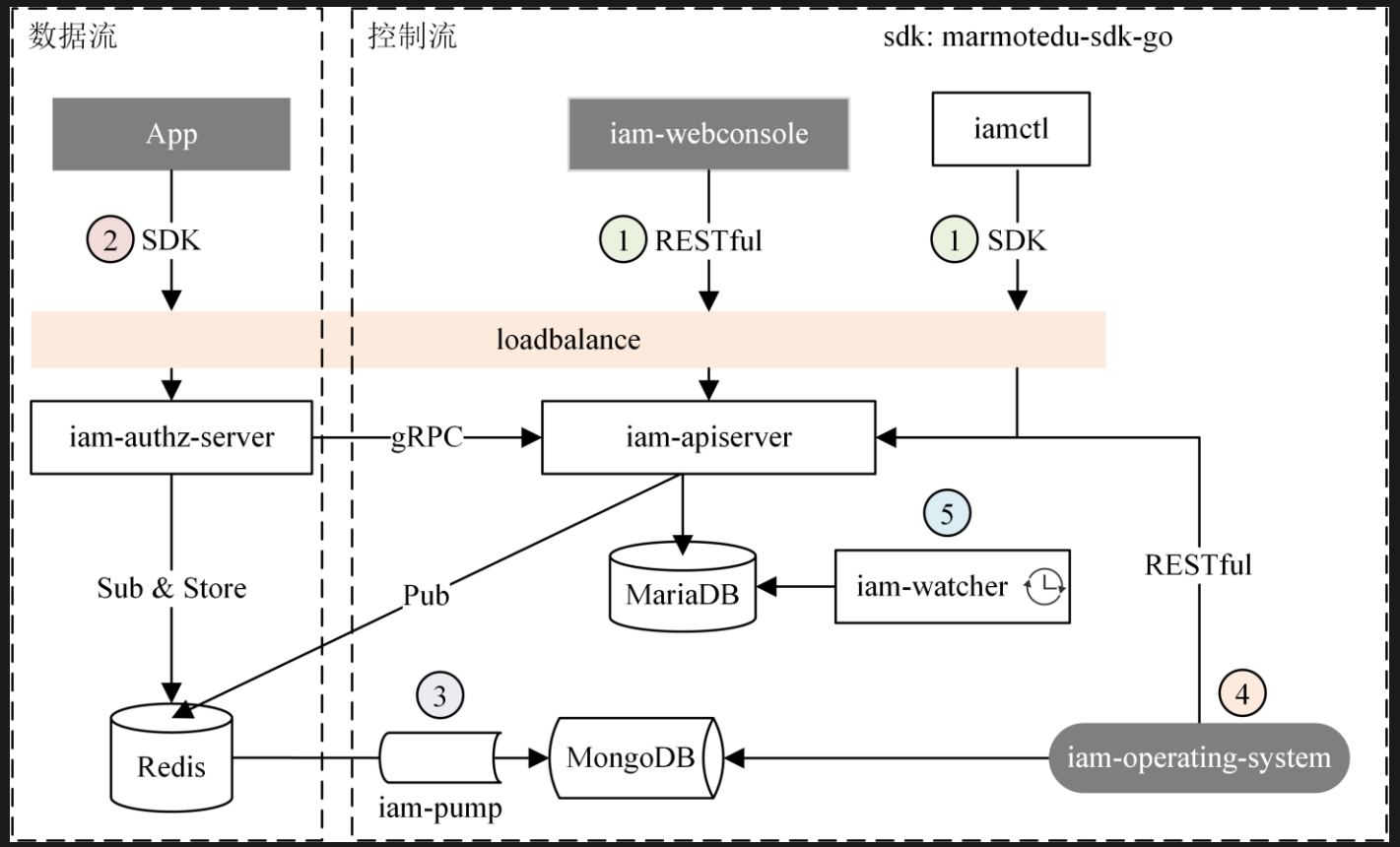
IAM 是一个基于 Go 语言开发的身份识别与访问管理系统，用于对资源访问进行授权。

覆盖绝大部分的开发技能

闭环Go项目开发：流程、功能、技能



IAM项目架构



1. 通过iam-webconsole、SDK、命令行工具iamctl请求iam-apiserver创建用户、密钥、策略，并保存在MariaDB中
2. 创建密钥、策略时会向Redis Channel中发布一条消息
3. iam-authz-server订阅这个Channel获取变更，并刷新密钥、策略缓存
4. 通过SDK调用iam-authz-server进行资源授权，并将授权信息保存在Redis中
5. 授权信息经过iam-pump组件处理后，保存在MongoDB中，供运营系统iam-operating-system展示
6. iam-watcher用来执行一些异步任务

基于声明式编程的软件架构

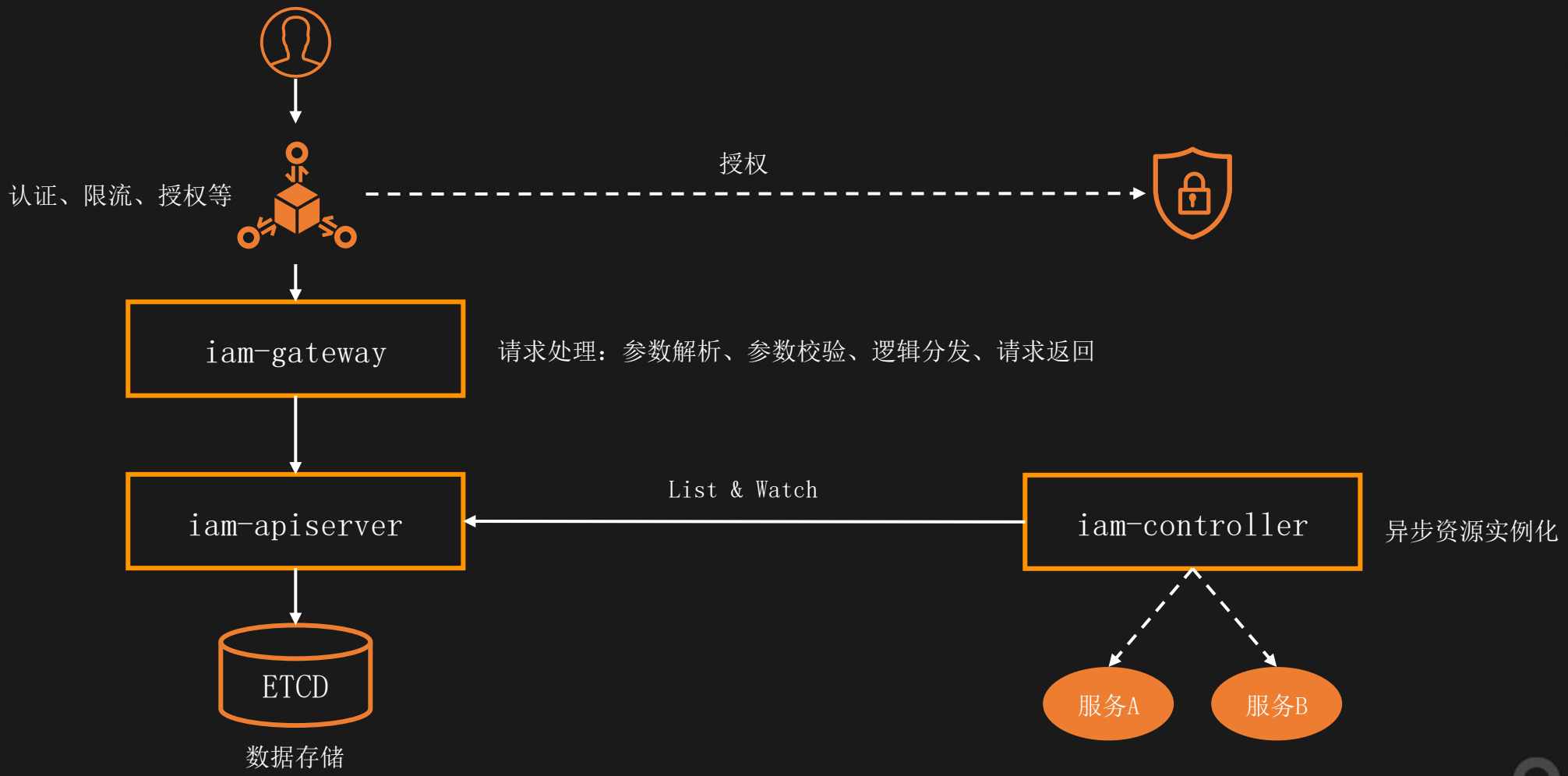
声明式编程 VS 命令式编程

命令式（Imperative）编程：命令“机器”如何去做事情(how)，这样不管你想要的是什么(what)，它都会按照你的命令实现。

声明式（Declarative）编程：告诉“机器”你想要的是什么(what)，让机器想出如何去做(how)。

- 声明式让程序逻辑变得更加简单，开发者只需要定义需要的应用，而不用去关注具体如何执行。
- 声明式更加直观，应用程序更加健壮（支持重试、自愈能力）；

一种声明式软件架构

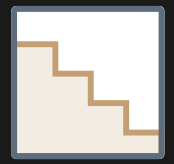


一种声明式软件架构

关于35岁焦虑的一点思考

我是如何解决35岁焦虑的

规划

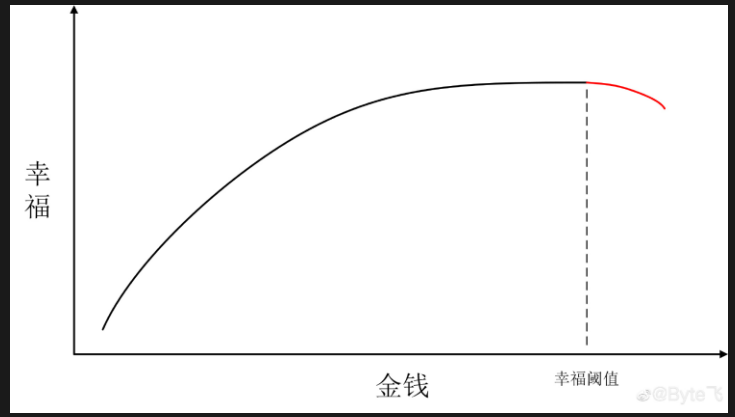


阶梯式规划

职业规划

人生规划

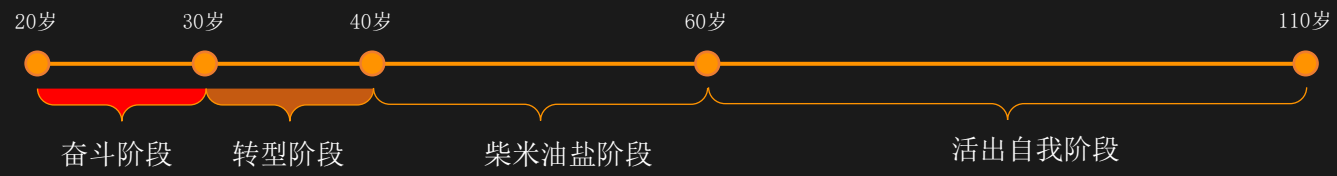
人生规划没有标准答案，只要你觉得成功的规划就是好的人生规划



钱够花就行

- 车
- 房
- 满足生活的经济收入

关于35岁焦虑的一点思考



- 20岁~30岁：要花很多时间去学习，夯实自己的能力，并建立核心竞争力，阶段你需要有一个小家庭，至少有个未来结婚的另一半吧；
- 30岁~40岁：
  - 最重要的任务是让20岁~30岁的积累产生最大的价值，要产生最大的价值，我觉得重点在于：选择+机遇；
  - 转型：慢慢的将重心从工作转移到家庭/生活上，花更多的时间去享受家庭、生活并教育好自己的孩子；
- 40~60岁：利用20岁~40岁的积累，工作上仍然能够有一个不错的收入，如果机遇好、选择好，还可以在事业上走得更远。工作之外，多陪伴家庭，享受生活；
- 60~100岁：但如果前面3个阶段做的好，60~100岁，怎么着都不会差。

在我看来关键点在 20岁~30岁 和 30岁~40岁 这2个阶段。

- 20岁~30岁：通过努力，具备幸福家庭最基本的物质条件。这些物质条件，可以通过职业规划去获取；
- 30岁~40岁：你需要具备一个良好的心态。通过20岁~30岁的努力，你已经具备了车、房和适量的存款。有了这些基本的物质保障，想要幸福并不难，只需要调整好自己的心态就行。比如：对爱情知足、对生活知足、对工作知足。

基本的物质条件+良好的心态，可以消除35岁焦虑

Thanks

《Go语言项目开发实战》专栏

- 专栏最终交付一套优秀、可运行的企业级代码
- 详解 Go 项目开发 5 大核心流程
- 带你一次掌握 Go 项目开发常用技能点
- 超多图谱辅助学习，老师社群高效答疑

专栏原价 ¥199，今日扫码
购买立减 ¥80，到手 ¥119

极客时间

入群方式

读者群二维码

若群二维码失效，也可以扫码添加老师个人微信，或搜索微信号“nightskong”添加。