GO语言开发规范

| 序号 | 规则级别 | 分类 | 规范内容 | 示例/说明 |
| --- | --- | --- | --- | --- |
| 1 | 强制 | 格式与风格 | 所有源代码必须使用 gofmt 格式化 | 自动统一缩进、空格、换行风格 |
| 2 | 强制 | 格式与风格 | 使用驼峰命名法，禁止下划线命名 | maxLength ✅，max\_length ❌ |
| 3 | 推荐 | 格式与风格 | 函数命名应简洁，不重复包名或类型上下文 | yamlconfig.Parse() 而非 yamlconfig.ParseYAMLConfig() |
| 4 | 推荐 | 格式与风格 | import 语句应三段分组：标准库、第三方、本地包，并使用空行隔开 | 可用 goimports 工具自动完成 |
| 5 | 允许 | 格式与风格 | \_ 导入仅限在 main 或测试文件中使用 | 如注册数据库驱动：\_ "github.com/lib/pq" |
| 6 | 推荐 | 格式与风格 | 启用 go vet / staticcheck 等工具确保代码质量 | 检查死代码、无效判断等 |
| 7 | 推荐 | 格式与风格 | 函数名避免冗余前缀 Get，已知上下文信息不再重复 | User.Name() 优于 User.GetName() |
| 8 | 推荐 | 格式与风格 | 长行不强制换行，优先重构逻辑而非换行 | 函数应职责单一 |
| 9 | 推荐 | 包与模块组织 | 包名应有语义，避免 util / common / helper | 比如 jsonfmt 比 helper 更具体 |
| 10 | 推荐 | 格式与风格 | 局部变量应语义清晰，仅在特殊情形下允许 i / t / r / w 等短名 | 例如 HTTP 中 w http.ResponseWriter 合理 |
| 11 | 推荐 | 包与模块组织 | 使用 Go Modules (go.mod) 进行依赖管理 | 模块路径建议使用 github.com/your/repo 形式 |
| 12 | 推荐 | 包与模块组织 | 可执行程序放在 cmd/<app> 目录下，仅包含引导逻辑 | 复杂逻辑应放入 internal/ 或 pkg/ |
| 13 | 推荐 | 包与模块组织 | 内部使用代码放入 internal/，外部复用代码放入 pkg/ | 避免滥用公共 API |
| 14 | 推荐 | 包与模块组织 | 包名避免与标准库冲突，如 json, url, time | 防止引起导入混淆 |
| 15 | 强制 | 可维护性 | 禁止使用全局可变变量 | 推荐使用依赖注入或封装 |
| 16 | 推荐 | 接口设计 | 接口类型应以值传递，不使用 \*Interface 指针 | 接口本身已包含对内部类型的引用 |
| 17 | 推荐 | 接口设计 | 倾向于小接口（Narrow Interface），只暴露最小功能集 | io.Reader ✅，避免胖接口 UserManager❌ |
| 18 | 强制 | 错误处理 | 所有 error 必须显式检查或处理，禁止忽略 | \_, err := f(); if err != nil { ... } |
| 19 | 推荐 | 错误处理 | 避免用特殊值如 -1、空字符串表示错误，建议返回 (value, error) 或 (value, ok) | GetUserByID(id) (User, error) |
| 20 | 推荐 | 并发 | 并发任务可使用 errgroup 管理错误和统一取消 | errgroup.WithContext() |
| 21 | 强制 | 错误处理 | 库代码中不得直接 panic，必须返回错误 | panic 只适用于断言失败等不可恢复场景 |
| 22 | 推荐 | 错误处理 | panic 仅限内部 bug 检查，不能越过包边界 | 如：索引越界、nil 解引用等内部断言 |
| 23 | 推荐 | 错误处理 | 使用 context.WithCancel/Timeout 后应立即 defer cancel() 释放资源 | go<br>ctx, cancel := context.WithTimeout(...); defer cancel() |
| 24 | 推荐 | 并发安全 | 使用 sync.Mutex 推荐嵌入结构体中直接用值，不用指针 | go<br>type Foo struct { sync.Mutex } |
| 25 | 推荐 | 并发安全 | Lock() / Close() 等资源应使用 defer 释放 | 推荐写法：mu.Lock(); defer mu.Unlock() |
| 26 | 推荐 | 并发安全 | 若保存切片/Map，需拷贝副本以避免数据竞争 | d.trips = append([]Trip(nil), trips...) |
| 27 | 推荐 | 并发安全 | chan 缓冲区大小优先设为 1 或无缓冲，其他值需说明理由 | 官方建议：make(chan Job, 1) |
| 28 | 强制 | 可维护性 | 所有导出类型/函数/变量都必须有文档注释，且以名称开头 | // Parse 解析配置内容... |
| 29 | 推荐 | 可测试性 | 使用表驱动测试 + t.Run() 子测试组织用例 | go<br>for \_, tc := range cases { t.Run(tc.name, func(t \*testing.T) { ... }) } |
| 30 | 推荐 | 测试 | 测试覆盖率应覆盖关键路径，建议保持在 80% 以上 | 通过 go test -cover 工具检查 |
| 31 | 强制 | 测试 | 所有公共 API（导出函数）必须至少有一个单元测试 | 特别是业务核心逻辑必须测试覆盖 |
| 32 | 推荐 | 日志 | 日志记录应使用结构化日志，推荐使用 zap、logrus 等库 | 避免 fmt.Println()，使用字段型日志记录 |
| 33 | 强制 | 日志 | 禁止将敏感信息如密码、Token、身份证号记录进日志 | 日志中仅记录 userID、匿名数据 |
| 34 | 推荐 | 安全 | 所有网络请求必须有超时时间设定 | http.Client{Timeout: 5 \* time.Second} |
| 35 | 推荐 | 安全 | 不允许在业务逻辑中出现硬编码密钥或配置 | 推荐使用配置文件或环境变量注入 |
| 36 | 推荐 | 安全 | 所有配置加载应支持默认值与覆盖 | 使用 viper, envconfig, flag 等工具 |
| 37 | 推荐 | 构建与部署 | 推荐使用 Makefile 或 Taskfile.yml 管理构建流程 | 构建、测试、打包、CI 集成保持一致性 |
| 38 | 推荐 | 构建与部署 | 所有项目应支持通过环境变量进行配置 | 如 PORT, LOG\_LEVEL, DB\_DSN 等 |
| 39 | 推荐 | 文档 | 每个包应有 doc.go 文件，说明包的用途与设计思路 | GoDoc 与 pkg.go.dev 可自动识别 |
| 40 | 推荐 | 接口设计 | 不为测试而定义接口（Interface） | 接口应有稳定的消费场景，否则使用具体结构体 |
| 41 | 推荐 | 性能优化 | 避免在热路径中频繁申请大对象，应使用对象池 | 使用 sync.Pool 优化临时对象 |
| 42 | 推荐 | 性能优化 | 批量操作尽可能避免循环内多次网络/数据库调用 | 使用 batch SQL、事务处理、缓存预热 |
| 43 | 推荐 | 数据结构 | 若结构体存在嵌套，应控制层级避免嵌套超过 3 层 | 简化结构便于阅读与序列化 |
| 44 | 推荐 | 序列化 | 使用 json 标签时，应明确所有字段，禁用不使用的字段 | 使用 json:"-" 忽略导出字段 |
| 45 | 推荐 | 协程管理 | 启动的 goroutine 应有退出机制，避免内存泄漏 | 可结合 context.Context 控制 |
| 46 | 推荐 | 协程管理 | 批量启动协程应控制数量，推荐使用协程池 | 推荐使用 ants, tunny, 或自建池 |
| 47 | 推荐 | CI/CD | 推送主干前必须通过 go test、go vet、golangci-lint 检查 | 作为 GitHub Action / GitLab CI 步骤配置 |
| 48 | 推荐 | 可维护性 | 单个 .go 文件内逻辑不宜超过 300 行，建议按职责拆分 | 有利于阅读与测试 |
| 49 | 推荐 | 包与模块组织 | 避免跨包引用非导出成员（例如通过 unsafe、反射方式访问） | 提高封装性与模块独立性 |
| 50 | 强制 | 模块边界 | 禁止循环依赖（cyclic import），应通过中间层或接口解耦 | package A 依赖 B，B 也依赖 A 为严重设计缺陷 |
| 51 | 推荐 | 第三方依赖 | 所有依赖应显式固定版本（不允许 latest 或浮动版本） | go.mod 中明确指定版本 |
| 52 | 推荐 | 第三方依赖 | 优先选择社区活跃、文档完善、贡献者多的开源库 | 避免使用无人维护的小众库 |
| 53 | 推荐 | 微服务 | 每个微服务应暴露健康检查 /healthz 接口 | 支持 Kubernetes readiness / liveness probe |
| 54 | 推荐 | 微服务 | 微服务间通信建议统一为 gRPC 或 REST，避免多种协议混用 | 有利于统一治理与网关代理 |
| 55 | 推荐 | 微服务 | 推荐使用统一的错误码、日志字段、链路追踪规范 | 便于集中监控与排障 |
| 56 | 推荐 | 监控与可观测性 | 所有服务应提供基础指标，如 QPS、ErrorRate、延迟分布，支持 Prometheus 导出 | 可使用 promhttp.Handler() |
| 57 | 推荐 | 监控与可观测性 | 所有入参请求建议打日志（敏感信息除外），所有错误必须打日志 | 便于排查与复现 |
| 58 | 推荐 | DevOps | 推荐使用 Dockerfile 构建镜像，并使用 .dockerignore 优化上下文 | 减少构建镜像大小与上下文体积 |
| 59 | 推荐 | DevOps | 所有服务必须支持优雅停止（graceful shutdown） | 使用 signal.NotifyContext 或 http.Server.Shutdown |
| 60 | 允许 | 性能调优 | 可在关键路径使用 unsafe 或 atomic，但必须附详细注释并经团队评审 | 如 atomic.AddInt64() 操作热点计数器 |