API 设计原则

杨子凡

Jul 15, 2026

API 作为系统之间的契约，一旦对外发布，就意味着未来的每一次迭代都将与历史版本共同存在。Twitter 从 v1 到 v1.1 再到 v2 的三次大规模重构，耗费了数年时间，也让大量第三方应用不得不反复适配新的接口。这类案例反复提醒我们：接口设计不是一次性工程，而是需要长期维护的技术债务。
本文面向具备一到三年后端或全栈开发经验的工程师，目标是提供一套可直接用于 Code Review 的检查清单，帮助读者在内部服务或对外 API 中建立「经得起迭代」的接口规范。

1　核心设计原则

设计一个健壮的 API，首先要确立几条贯穿始终的原则。以消费者为中心，而不是以内部实现为中心，这意味着接口的命名、结构和行为都应贴近调用方的使用场景，而不是暴露服务端的表结构或方法名。最小惊讶原则要求接口的行为符合调用者的直觉，例如用名词表示资源、用 HTTP 方法表示动作，避免在不同端点出现语义相悖的返回结构。演进优于革命，意味着在不破坏已有契约的前提下，通过可选字段、默认值兼容等方式逐步添加能力。显式优于隐式，则强调把版本、错误、权限等关键信息放在请求或响应中可见的位置，而不是依靠约定或文档口头传递。幂等性与安全性默认开启，要求写操作接口在重试或并发场景下不会产生副作用，同时传输层默认使用 HTTPS 并开启 HSTS。

2　命名与资源建模

在 REST 风格的接口中，资源路径应使用名词的复数形式。`/getUserInfo` 或 `/createOrder` 这类把动作放在路径里的写法，会让调用者误以为服务器暴露了内部方法，而 `/users` 和 `/orders` 则清晰地表达了操作对象。
当需要表达复杂业务动作时，可以把动作作为子资源或自定义动作，例如 `POST /orders/{id}/cancel`，把取消视为订单的一种状态转换；或者 `POST /emails/{id}/send`，把发送视为邮件的一种操作。
命名的一致性同样重要。团队需要提前约定 snake_case 或 camelCase，并在整个代码库和文档中严格遵守。同时建立领域术语表，避免同一概念出现 user、member、account 等多种表述。

3　版本控制策略

版本控制主要有两种常见做法。URI 版本把版本号放在路径中，例如 `/v1/users`，优点是直观，缺点是每次升级都可能需要新的端点；Header 版本则通过 `Accept: application/vnd.company.v1+json` 传递版本信息，URL 保持不变，但调用方需要额外设置请求头。
无论采用哪种方式，向后兼容都需要遵循三条原则：不删除已有字段、不改变字段的语义、新增字段必须可选且

提供默认值。灰度发布时，可以让多个版本并存，通过路由或网关把不同客户端导向对应版本的服务实例。

4 请求与响应结构

请求参数的命名应保持统一。过滤使用 `filter`，分页使用 `page`，排序使用 `sort`，当查询条件变得复杂时，建议改用 `POST /search` 并在请求体中携带结构化条件，避免 GET 参数无限膨胀。
响应结构推荐采用统一的壳，例如包含 `data`、`error`、`meta` 三个顶级字段。错误信息则遵循 RFC 7807 Problem Details 规范，用 `type`、`title`、`status`、`detail` 等字段描述错误。HATEOAS 即超媒体驱动的 API，可以在需要客户端动态发现下一步操作时引入，但对大多数内部服务来说，维护成本高于收益。
数据传输格式上，JSON 是默认选择；Protobuf 或 gRPC 更适合高性能的内部服务；OpenAPI/Swagger 则用于自动生成契约文档，让接口定义与实现保持同步。

5 安全与权限

传输层必须强制使用 HTTPS，并通过 HSTS 防止降级攻击。认证与授权推荐采用 OAuth 2.0 或 OIDC 流程，JWT 适合无状态场景，而 opaque token 更适合需要实时撤销的场景。
权限模型从 RBAC 向 ABAC 演进，可以在策略中引入时间、IP、资源标签等属性，实现更细粒度的控制。限流方面，返回 429 状态码并携带 `Retry-After` 头，能让客户端按建议时间重试。幂等性则通过 `Idempotency-Key` 请求头实现，服务端对相同键的请求只执行一次副作用操作。

6 错误处理与可观测性

错误应区分客户端错误（4xx）和服务端错误（5xx）。HTTP 状态码用于快速判断大类，内部 `error_code` 用于精确定位，人类可读的 `message` 则用于日志和用户提示。
日志需包含结构化字段，例如 `trace_id`、`user_id`、`latency`，以便在分布式追踪系统中串联请求链路。OpenTelemetry 提供了标准的 SDK，可以在中间件中自动注入这些字段，并把指标、日志、追踪统一导出到后端存储。

7 性能与扩展

缓存头的使用能显著降低重复计算。`ETag` 配合条件请求可实现精确的缓存失效；`Cache-Control` 控制浏览器和 CDN 的缓存时长；CDN 则把静态或半静态响应推到边缘节点。
当响应体过大时，可以通过 `?fields=id,name` 实现字段过滤，让客户端只拉取必要数据。批量操作适合用 `POST /users/batch` 封装多个子请求，但需注意原子性和部分失败的处理。GraphQL 在灵活查询和减少往返次数上有优势，但 REST 在缓存和简单性上仍有不可替代的位置。

8 文档与开发者体验

文档即代码的理念要求接口定义与实现同步更新。OpenAPI 3.0 定义文件可以直接生成 Redoc 或 Swagger UI，让调用者在线浏览和调试。SDK 和 CLI 工具可通过代码生成器从 OpenAPI 文件自动产出，减少手动维护成本。

提供 Playground 或 Sandbox 环境，让开发者在不影响生产数据的前提下验证调用。变更日志应记录每个版本的破坏性变更、新增字段和废弃计划，模板通常包含版本号、日期、变更类型、影响范围和迁移指南。

9 演进与废弃策略

废弃接口需要提前公告。`Deprecation` 和 `Sunset` 响应头可以告诉客户端该接口将在何时下线，并给出替代端点。并行运行期通常设置为数月到一年，之后再强制下线。破坏性变更前，需检查所有契约测试是否覆盖新旧版本的兼容场景，并更新 SDK 和文档。

10 真实案例复盘

GitHub 从 REST v3 迁移到 GraphQL v4 的过程，展示了如何在保持向后兼容的同时引入更灵活的查询能力。Stripe 的版本策略则把版本号放在请求头中，每个版本独立维护，调用方可按需升级。内部 BFF 重构案例中，团队通过引入统一的响应壳和错误格式，把多个后端服务的差异屏蔽在 BFF 层，显著降低了前端的适配成本。

11 落地 checklist

资源路径是否符合 RESTful 规范，是否提供 OpenAPI 3.0 定义，错误格式是否统一，是否支持幂等与重试，是否在契约测试中覆盖破坏性变更，是否有明确的废弃公告流程，这些问题可以在每次 Code Review 时逐一核对。

API 设计是一门长期主义的技术债务管理艺术。把上述原则和 checklist 应用到下一个 PR 中，并把实际遇到的边界情况反馈到团队讨论中，才能持续迭代出真正经得起时间考验的接口。

12 附录

推荐阅读书单包括 JJ Geewax 撰写的《API Design Patterns》和 Mike Amundsen 撰写的《Design and Build Great Web APIs》。常用工具与模板仓库可从 OpenAPI 官方示例和 Redoc 社区获取。中英文术语对照表可帮助团队统一「resource」「endpoint」「deprecation」等概念的中文表述。