API 设计原则

1. **充分原则** 不是随便一个功能就要有个接口，也不是随便一个需求就要加个接口。每新建一个接口，要有充分的理由和考虑，即这个接口的存在是十分有意义和价值的。无意义的接口不仅增加了维护的难度，更重要是对于程序的可控性的大大降低，接口也会十分臃肿。
2. **单一视角原则** 设计接口时，分析的角度要统一。否则会造成接口结构的混乱。例如：不要一会以角色的角度设计，一会儿就要以功能的角度设计。推荐：以”属性对象 + 行为”的视角定义API。
3. **单一功能原则** 每个API接口应该只专注一件事，并做好。产品概念简单、关系清楚。功能模棱两可，诸多特殊逻辑的API肯定不是个优雅的API，且会造成功能类似重复的API。 注：如果API它很难命名，那么这或许是个不好的征兆，好的名称（概念）可以驱动开发、并且只需拆分与合并模块即可。 功能大而全的API在灵活性、简单性方面肯定捉襟见肘。定义API的粒度之前，建议先将业务分领域、划边界，以此来提取业务对象，然后再根据业务对象用例来设计单一功能的API。 比如：查询会员，可能除了查询会员表外还要获取该会员的其他必要信息，但不要在查询会员的同时还有修改权限等类似的其他业务功能，应该分成两个接口执行。
4. **简单原则** 接口设计简单、清晰。API执行的功能可以很丰富、很强大，但API声明和用法一定要尽量的简单，不能将功能的丰富通过复杂的用法来实现，这会导致API功能不单一，演进不可控。最终的评审要看API的简单易用程度。 你写的例子，能不能让你的代码看起来更简单？ 你是不是强迫调用方关注/提供他们不在乎的选项/配置？ 有没有毫无价值的额外步骤？ 编写的代码一定要易于读、易于理解，这样别人才会欣赏，也能够给你提出合理化的建议。相反，若是繁杂难解的程序，其他人总是会避而远之的。
5. **抽象原则** API的入参、出参所述的对象、属性，一定是按业务特性进行抽象后的实体。误将底层数据模型概念如实的反应到API上。抽象API、抽象对象实体更宏观，具有更好的适用性、兼容性、扩展性。
6. **兼容扩展原则** 对扩展开放，对修改关闭。保证API的向后兼容。扩展参数应当是便利的，保证后续类似的需求，可以在已有的API上通过兼容扩展的方式实现。
7. **最小惊讶原则** 代码应该尽可能减少让读者惊喜。业务API只需根据需求来设计即可，不需要刻意去设计一下复杂无用、华而不实的API，以免弄巧成拙。
8. **低耦合原则** API应该减少对其他业务代码的依赖关系。低耦合往往是完美结构系统和优秀设计的标志。 耦合的种类：
    - 代码实现业务逆向调用。
    - 条件逻辑依赖耦合。例如：调用某个API之后（或之前）必须再调用另外一个API才行。
    - 耦合API功能设计之外的无关业务行为。
9. **正交原则** 正交性是指改变某个特性而不会影响到其他的特性。API之间的功能应该成正交性，无功能重合，无相互影响。API之间应该是相互独立、相互补充的关系。

10. **统一原则** API 要具备统一的命名、统一的入/出参规范、统一的异常处理规范、统一的结果码规范、统一的版本规范等。 统一规范的API优点：
    - 易于被框架集成、处理。
    - 有助于API调用方和API提供方按照统一模式复用经验、避免误用。

优秀 API 的特质：

- **自解释**
    - 从API本身一眼就能看懂 API 是干什么的，适用的场景，支持的用法，内部流程、异常如何处理等。
- **易学习**
    - 有完善的文档，以及提供尽可能多的示例和可 copy－paste 的代码。
- **易使用**
    - 功能强大，但使用简单。不增加调用方的使用成本（例如要求业务方用API时需要额外的配置和依赖），不暴露复杂的细节、冗长的使用流程给调用方感知。调用方只做最小的感知和最少的传参。
- **难误用**
    - 优秀的API可以使有经验的开发直接使用 API 而不需要阅读文档。
    - 明确的参数定义、充分的静态检查、动态校验、显式的异常说明、有效的错误提示。
- **易测试**
    - 对于API调用者而言，API应该是可被测试且易于被测试的。测试API不需要依赖额外的环境、容器、配置、公共服务等。
    - 对可测试友好的API也是可被有效集成测试的前提。