API设计与实现

主讲人：陈长兵

目 录

CONTENTS

1 绪论 1#

2 API设计概论 1#

3 API设计规范 2#

4 API设计模式 8#

5 API安全 8#

6 API技术实现 12#

3 API设计规范

- API设计评估
- 通用设计原则
- API设计原则

3.1什么是好的API

3.1什么是好的API

Operational

- Operational：可行，可用
- 功能需求：提供用户真实想要的操作
- 非功能需求（性能）：耗时，准确性等

3.1什么是好的API

Expressive

- Expressive：富有表现，表达能力强
- 能够便捷准确表示和达成用户的诉求
  - 功能和性能方面都可以准确的表达
- 隐藏功能
  - TranslateText vs DetectLanguage

3.1什么是好的API

Simple

- Simple：简单，简洁，直接
- 简单并不一定少
  - 比如通过一个方法执行所有操作
  - 复杂被转移了，方法多 -> 配置多
- 常用情况 & 高级版本
  - 常用情况简单容易，同时为高级版本留有可能性
  - 为高级版本扩展功能，但不要增加常用情况复杂程度
  - TranslateText，高级模型

3.1什么是好的API

Predictable

- Predictable：可预测的
- 各API的接口定义和行为一致性
  - TranslateText入参text，DetectLanguage入参content？
  - 类比，一致性，延续性
- 基于模式的认知
  - 观察局部，建立模式，推演验证，预期管理
  - 易于学习，使用时降低心智负担

3.2通用设计原则

3.2通用设计原则

简单原则

- 声明直观清晰
- 功能可见性
- 简单易用程度
- 单一职责

3.2通用设计原则

统一原则

- 统一的命名规范
- 统一的入/出参规范
- 统一的错误码规范
- 统一的版本规范

3.2通用设计原则

抽象原则

- 按业务特性抽象实体
- 最小信息，接口粒度
- 恰当的抽象与具体
- 更好的适用性、扩展性

3.2通用设计原则

正交原则

- 覆盖全业务流程
- MECE原则，相互独立&完全穷尽
- 高内聚低耦合

3.2通用设计原则

兼容扩展原则

- 开闭原则
- 保证API的后向兼容
- 扩展参数便利

3.2通用设计原则

安全原则

- 认证授权，访问控制
- API自身保护
- 传输数据保护

3.3API设计原则

API
构成

- 操作
- 参数
- 组织结构

1 命令规则

2 数据类型和默认值

3 资源范围和层级

3.3API设计原则-命名规则

好的命名特点

- Expressive
- Simple
- Predictable

3.3API设计原则-命名规则

命名的机制

- 语言
  - 美式英语 & 英式英语
  - 用词，电影movie & film，饼干cookie & biscuit
  - 拼写，颜色color & colour，识别recognize &

3.3API设计原则-命名规则

命名的机制

- 语法
  - 命令式动词
  - 介词
  - 复数

3.3API设计原则-命名规则

命名的机制

- 词法
  - 大小写
  - 关键词

3.3API设计原则-命名规则

四种命名方法

- 帕斯卡命名法
  - 单词之间不以空断开，不以连接号、下划线连接
  - 第一个单词首字母大写，后续单词首字母也采用大写
  - eg：ShowMessage

3.3API设计原则-命名规则

四种命名方法

- 驼峰命名法
  - 小驼峰法，除第一单词外，其他单词首字母大写
    - 一般用于变量命名
  - 大驼峰法，即帕斯卡命名法
    - 常用于类名、命名空间等

3.3API设计原则-命名规则

四种命名方法

- 下划线命名法
  - 单词一般都采用小写，单词之间用下划线连接
  - 与帕斯卡和驼峰命名的差异，就是单词之间逻辑断点的连接符不同

3.3API设计原则-命名规则

四种命名方法

- 匈牙利命名法
  - 构成=属性+类型+对象描述
  - 属性：g_全局变量，c_常量，m_成员变量，s_静态变量
  - 类型：a数组，p指针，fn函数，v无效，h句柄，b布尔，f浮点等
  - 描述：Max最大，Init初始化，Src源对象，Dest目的对象等
  - eg1：hwnd，h是类型，wnd是描述，表示窗口句柄
  - eg2：pfnEatApple，pfn是类型（函数指针），EatApple是描述

3.3API设计原则-命名规则

其他考虑因素

- 上下文Context
  - 上下文携带信息，影响名字的含义
  - 限定含义，或赋予特有含义
  - eg1：book，图书管理API，航班销售API
  - eg2：record，录音API

3.3API设计原则-命名规则

其他考虑因素

- 数据类型和单位
  - 数据没有单位会引起混乱
  - eg1：图片的大小size字段，是字节大小sizeBytes，还是尺寸大小sizeMegapixels？
  - 引入丰富的数据类型：

```
interface Image {
    content: string;
    sizeBytes: number;
    dimensions: Dimensions;
}
interface Dimensions {
    lengthPixels: number;
    widthPixels: number;
}
```

3.3API设计原则-数据类型和默认值

数据类型介绍

- 数据类型是每个编程语言的重要组成部分
  - API使用者可能涉及多种编程语言
  - 布尔、数值、字符、枚举、列表、Map等
- 动态类型，Dynamic Typing
- 序列化 & 反序列化，信息丢失
- 缺失与空值

3.3API设计原则-数据类型和默认值

布尔类型

- 一定应用场景
  - eg1：archived: boolean, 是否已归档
  - eg2：allowChatbots: boolean，是否允许聊天机器人
- 二值表达力有限，可能有更好的方案
  - allowChatbots, allowModerators, allowChildren, allowAnonymousUsers
- 避免double negative
  - disallowChatbots + false = 允许

3.3API设计原则-数据类型和默认值

数值类型

- 存在算术计算诉求的场景
  - eg1：计数类viewCount，大小类itemWeight，金额priceUsd
- 不适用于纯数值类标识符
  - eg1：手机号，银行卡号
  - 更适合适用字符类型
- 整型int，长整型long，单精度浮点float，双精度double等

3.3API设计原则-数据类型和默认值

数值类型

- 边界
  - char，单字节，-127~127
  - integer，4字节，-2^31~2^31-1，约正负21亿
  - long，8字节，-2^63-2^63-1，约十进制19位
  - float，单精度浮点，4字节，7位精度
  - double，双精度，8字节，16位精度
  - decimal，精准十进制类型，可变长度
  - 各编程语言存在差异，基本类型 & 对象类型

3.3API设计原则-数据类型和默认值

数值类型

- 默认值
  - 0或0.0？
  - 空值null，缺失Missing

3.3API设计原则-数据类型和默认值

数值类型

- 序列化
  - 很长的数值，25位十进制，9999999999999999999999998
  - 精度，0.1
  - 使用字符类型进行序列化

3.3API设计原则-数据类型和默认值

字符类型

- 最通用的数据类型
  - 名称、地址、长文本
  - 二进制的字符表示，Base64
  - 数值类标识符，数值的字符形式
- 编码，byte & char & Character
  - ASCII字符集
  - Unicode字符集，utf-8

3.3API设计原则-数据类型和默认值

字符类型

- 边界
  - 大小，长度 & 存储空间？
  - 应用层characters，存储层bytes
  - 超出如何处理，truncate or reject

3.3API设计原则-数据类型和默认值

字符类型

- 默认值
  - null值，空字符串
  - 特殊字符”default”

3.3API设计原则-数据类型和默认值

字符类型

◆ 序列化

- 字符串是由一组字节构成
- 字符集：字符的逻辑表示，区位码&国标码&(机)内码

ASCII，Latin-1/ISO-8859-1，Unicode，GB2312/GBK/GBK18030

- 编码格式：字符的字节存储

UTF-8/UTF-16/UTF-32，GB2312/GBK/GBK18030

- 大端BigEndian & 小端LittleEndian，字节序
- 符号，半角 & 全角

汉的表示

```
Unicode  : \u6C49
UTF-8    : E6B189
GB2312   : BABA
```

3.3API设计原则-数据类型和默认值

字符类型

```
Java
Normalizer.normalize("方", Form.NFKC);
```

- Unicode Normalization
  - 如何支持注音符号，emoji符号
  - 有些字符 与 其他字符或者一组字符序列，在含义是等价的
  - 标准等价，≠(\u2260) 与 =/(\u003d\u0338)
  - 兼容等价，㋋(\u328b) 与 火(\u706b)
  - 完全合成，完全分解
  - NFC：标准等价组合，NFD：标准等价分解
  - NFKC：兼容等价组合，NFKD：兼容等价分解

3.3API设计原则-数据类型和默认值

枚举类型

- Name(字符) + Value(数值)
- 优势：校验，压缩
- 不足：依赖码表，变更同步
- 在API场景传输时，优先使用字符形式

3.3API设计原则-数据类型和默认值

列表类型

- 基本类型的集合，也可以是其他列表或Map类型
  - 通过下标，来引用指定元素
  - 序列化框架的支持
  - 存储系统并不总是支持

3.3API设计原则-数据类型和默认值

列表类型

- 原子性
  - 列表类型的字段，变更：更新第5个元素？还是整体覆盖
  - 部分更新严重依赖元素的顺序
  - 整体覆盖，保持原子性
  - 列表各元素的数据类型是相同的
  - 混合时可使用字符类型，保持同类型

3.3API设计原则-数据类型和默认值

列表类型

- 边界
  - 列表大小
  - 两个方面：元素数量 & 元素大小
  - 超出大小，拒绝 优于 截断
  - 超大列表，可使用分页方式

3.3API设计原则-数据类型和默认值

列表类型

- 默认值
  - 空值null，空列表[]
  - 什么是默认值？初始值？未知值？
  - 代表一种信息或状态，{初始值，未知值} 与 {潜在合理值} 最好是可区分的

3.3API设计原则-数据类型和默认值

Map类型

- key+value，可分两类
  - 静态schema，固定的key集合 和 value类型
  - 动态key-value，key集合可灵活变更
- 数据平铺 -> 信息分组
  - 静态schema，已知字段进行分组管理
  - 动态key-value，定义API时字段未知
  - 食品店的产品成分，ingredientAmounts: Map<string, string>;

3.3API设计原则-数据类型和默认值

Map类型

- 边界
  - 主要针对动态key-value类型
  - key 的数量
  - 整体大小
- 默认值
  - 空值null，空Map{}

3.3API设计原则-资源范围和层级

资源布局

- 什么是资源布局？
  - API中有很多资源，它们是如何组织的
  - 是一种实体关系模型
  - 实体，构成实体的元素，以及实体之间的关系

3.3API设计原则-资源范围和层级

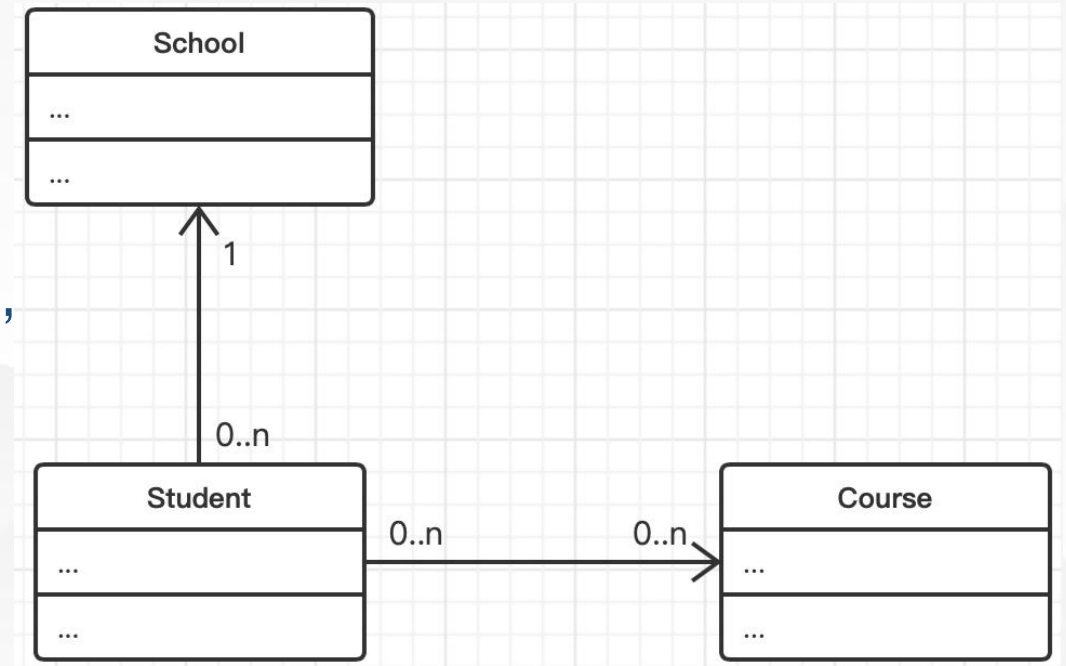
资源布局

- 关系的种类
  - 引用关系，Message A 的author是 User B
  - Many-to-One关系，多个User 参与 ChatRoom C
  - 自引用关系，Employee A 是 Employee B的经理
  - 层级关系，与引用关系相比，更关注归属，比如ChatRoom A下的Messages

3.3API设计原则-资源范围和层级

资源布局

- ER图
  - Entity Relationship Diagrams
  - UML，Unified Modeling Language，统一建模语言



3.3API设计原则-资源范围和层级

选择正确的关系

- 是否需要一个关系？
  - 建立关系是有代价的，性能变差，维护复杂
  - 关系也是有价值的，比如社交网络
  - 新增关系 一定与 某个重要的业务需求 挂钩

3.3API设计原则-资源范围和层级

选择正确的关系

- 使用引用Reference 还是 内嵌in-line？
  - 引用方式：通过指针找到真实对象
  - 内嵌方式：存储对象的全部数据
  - 差异：交互次数，传输数据大小
  - 考虑因素：具体场景具体看，需求意图，底层存储方式

3.3API设计原则-资源范围和层级

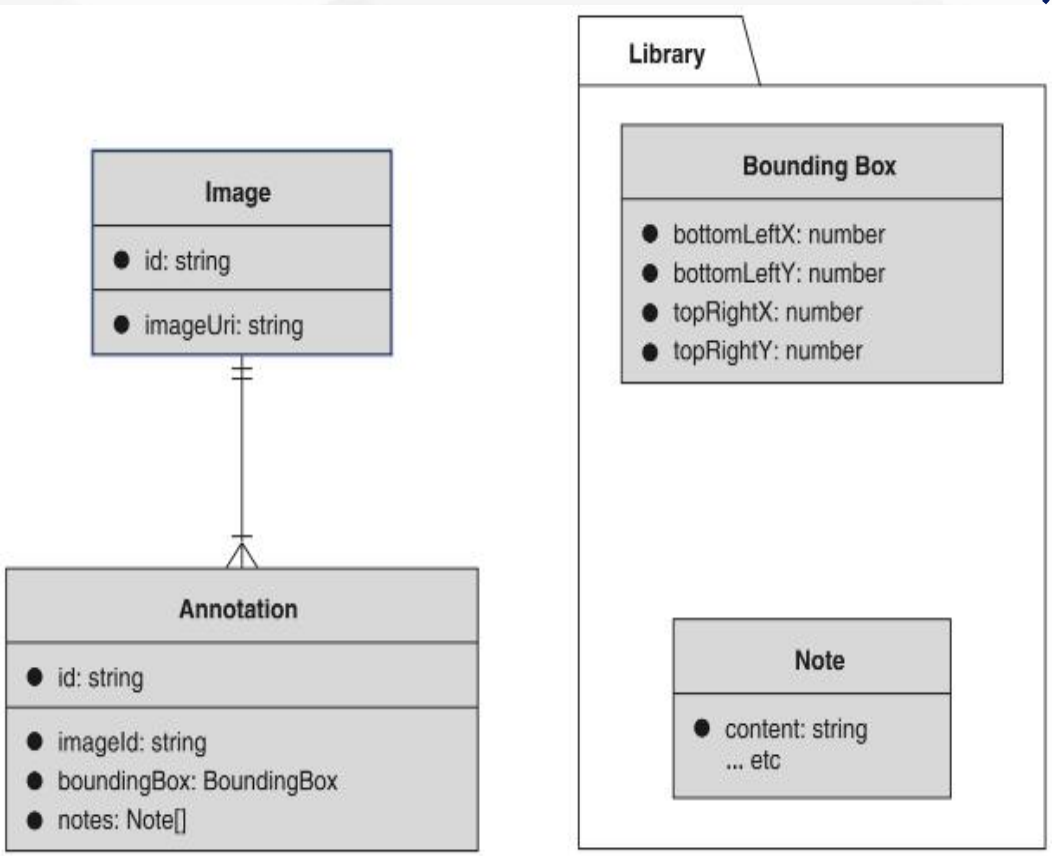
选择正确的关系

- 层级
  - 层级关系 or 引用关系？
  - 层级关系特点：操作的级联性
  - eg：ChatRoom & Message，删除 & 权限的级联性

3.3API设计原则-资源范围和层级

一些不好的模式

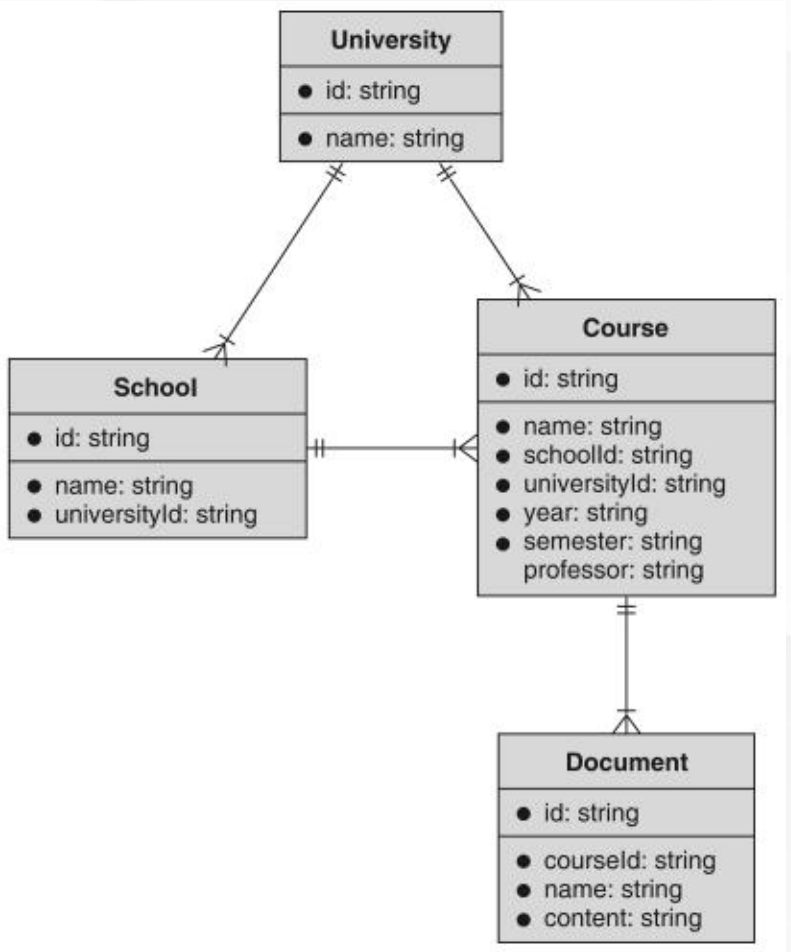
- 所有都是资源
  - 把任意的概念都看成资源
  - eg：图片标注API，概念：图片、标注、边界框、笔记
  - 是否需要独立交互？
  - 是否元素非常小且不是很大的集合？
  - 能否使用内嵌方法？



3.3API设计原则-资源范围和层级

一些不好的模式

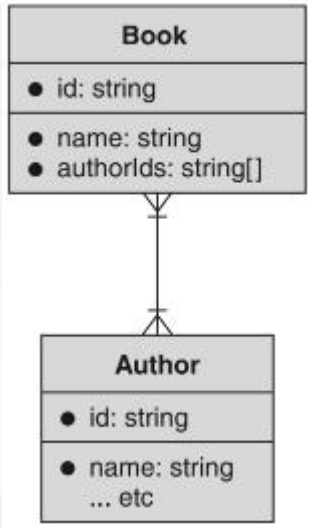
- 非常深的层级
  - 功能虽然强大，但不能过度使用
  - 不依赖过深的层级，也有有效的办法
  - eg：学校课程API，概念：学校，学院，课程，学期，教案
  - 分析关键概念，比如：学校，课程，教案？
  - 分析业务操作，比如：学期增减？根据学期查询课程？

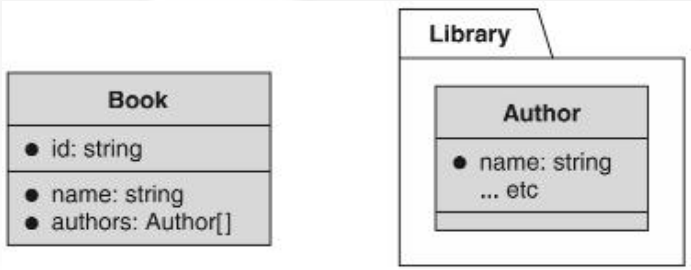


3.3API设计原则-资源范围和层级

一些不好的模式

- 都采用内嵌方式
  - 减少关联，去范式化
  - eg1：图书管理API，概念：图书，作者
  - 作者信息更新？
  - eg2：商品物流API，概念：用户，商品，订单，地址
  - 用户地址更新？订单中的地址是否需要更新？





QA