

Tencent 腾讯

# 如何写好文档-谷歌的文档

kuo@实践 凯

2022.05.27

# 目录


- What
- Why
- How
- 总结

主要节选自[\*Software Engineering at Google\*](#)第十章，结合我自身在谷歌的经验



你需要调研一个你不太熟悉的系统

# 你打开内部search敲入关键词...



XXX系统

×

搜索

XXX系统思考

XXXXXXXXXXXXXXXXXXXX

XXX系统设计与实现

XXXXXXXXXXXXXXXXXXXX

XXX系统2.0

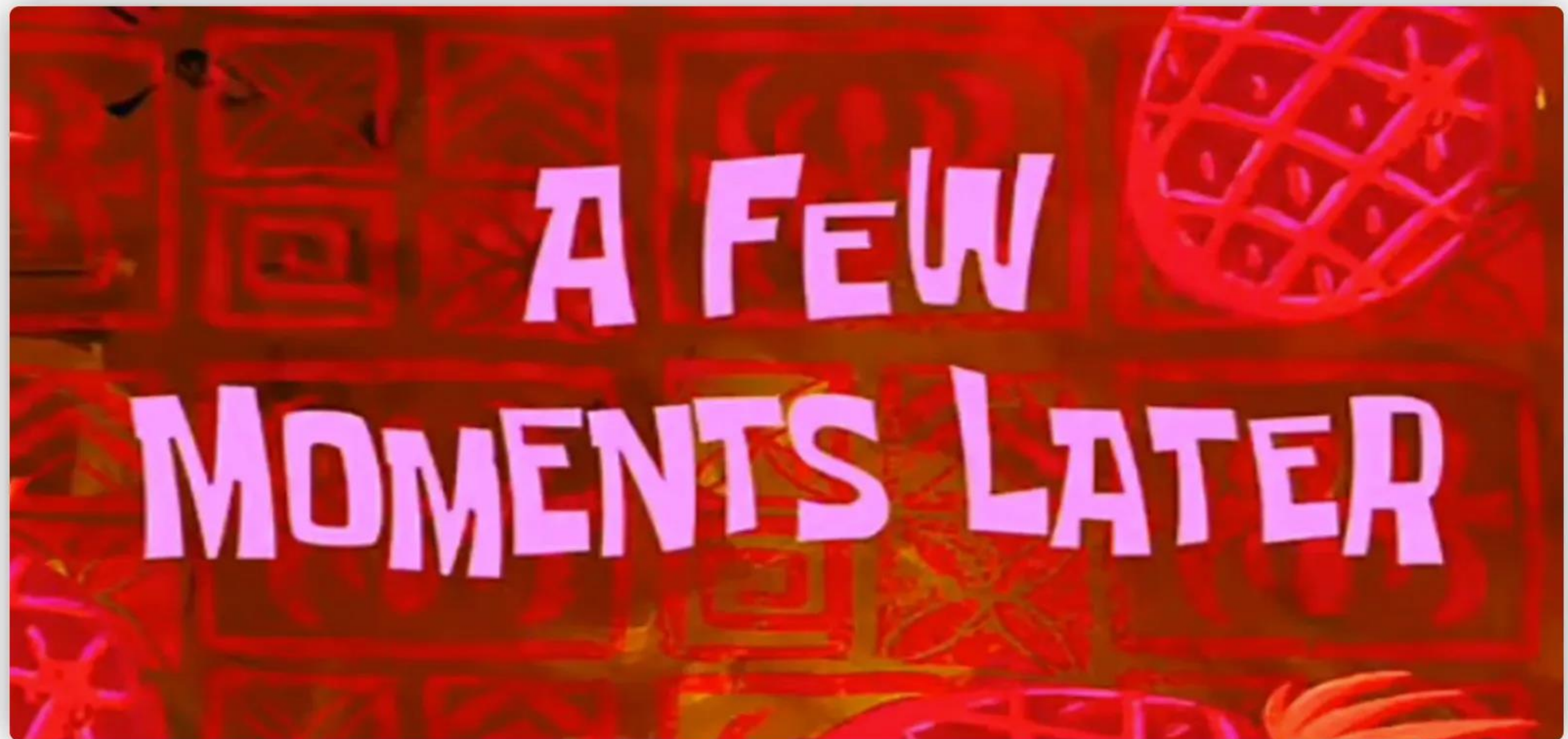
XXXXXXXXXXXXXXXXXXXX

分布式XXX系统

XXXXXXXXXXXXXXXXXXXX

...





你发现每个文档讲的都不太一样，  
没有一个权威的说法

保持笑容





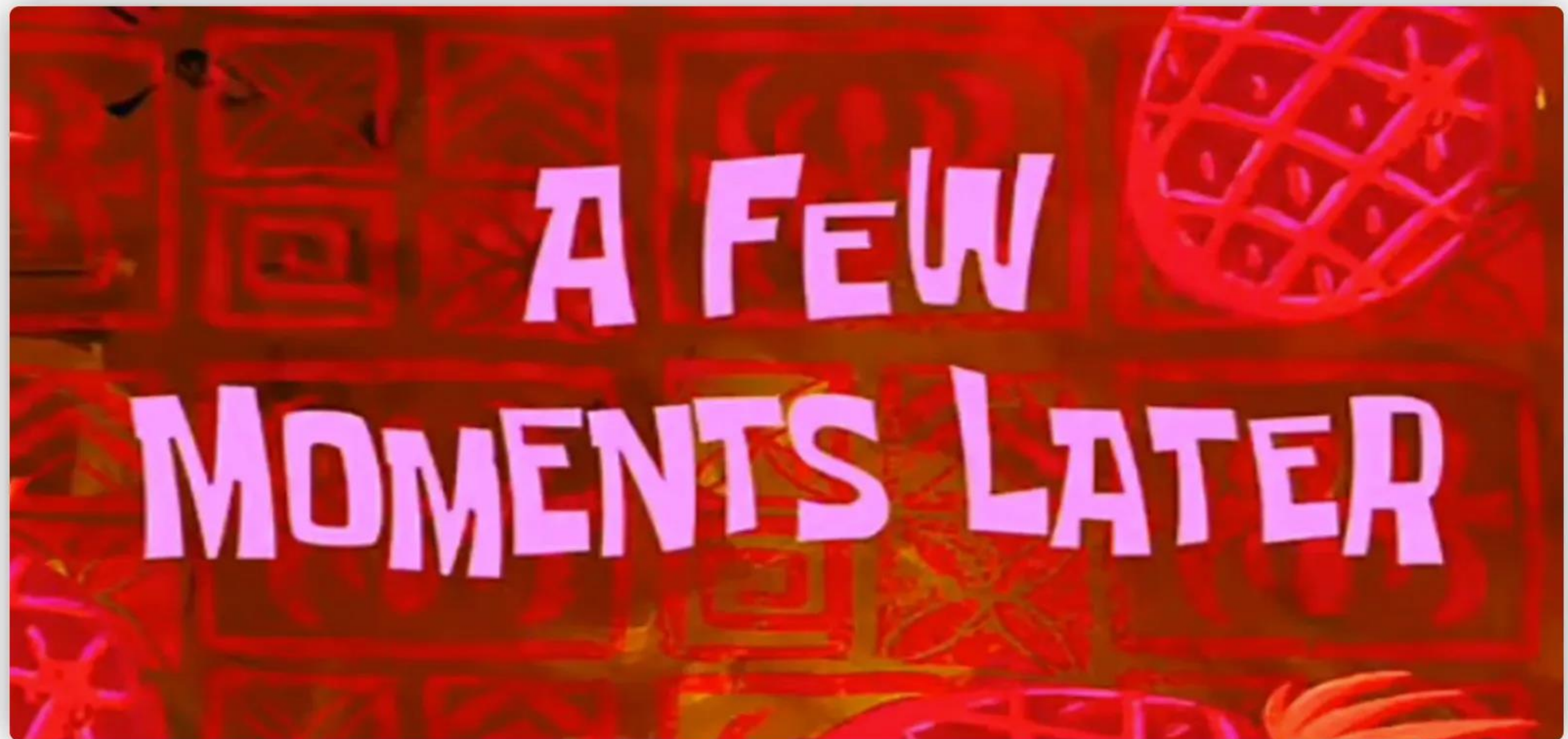
你好不容易联系上了相关负责人开会对齐，  
一开就是三小时



但有些细节你决定还是看看代码







结果你发现具体的实现跟文档里写的截然不同

笑容渐渐消失





# 1 What?

# 什么算作文档？

- 当我们提到“文档”时，我们谈论的是工程师为完成工作需要编写的每一个补充文本
- 不光是独立的文档（比如设计文档，接口文档）
- 还有代码注释（事实上，谷歌的工程师所写的大部分文档都是以代码注释的形式出现的）
- MR描述
- ...

## 媒资多租户技术设计文档

|            |   |
|------------|---|
| 作者         | @lanceduan(段海鹏) @zolachen(陈中良)  |
| 状态         | 草稿   审核中   通过   |
| 审核人 / 评审状态 | 组长: @zolachen(陈中良), 通过 / 未通过<br>TL: @kluo(KaiLuo) @yuankuisun(孙元魁), 通过 / 未通过  |
| iwiki链接    | <a href="https://iwiki.woa.com/pages/viewpage.action?pageId=760473609">https://iwiki.woa.com/pages/viewpage.action?pageId=760473609</a> |
| 最后更新日期     | 2022/05/19  |

### 目标

- 实现贯穿媒资Paas能力的多租户系统，提供租户管理、pass应用管理等能力
- 实现媒资字段、数据存储、数据访问、数据分发、数据加工多租户隔离
- 尽量平滑、对外协议兼容的方式实现多租户系统，避免额外的推广成本

### 背景

媒资系统依托于强大的数据管理、数据分发等能力，为ovbu乃至公司大量的业务提供支持，目前接入媒资的各类服务多达1650个，日均请求逾1亿次；

媒资的数据存储实现了实体级别的数据隔离，然而，随着业务的发展，腾讯视频媒资当前不仅管理了腾讯视频生产的vid，同时对来自其他业务生产vid内容也进行了管理，比如街舞社区，新闻，中视频，体育视频，NBA视频，素材视频等。对于同一实体而言，如何管理多个业务对同一实体数据做运营管理，同时避免相互之间产生影响成为了迫切需要解决的问题。



# 什么算作文档？

```
194 // ClassifyUpdateReqBySourceId 按照数据源id分类数据适配层更新请求
195 func ClassifyUpdateReqBySourceId(adaptorRepMap map[int32]*adaptor.SetFieldInfosRequest, req *access.MediaUpdateRequest,
196     retInfo *protocol.CommRetInfo, traceInfo string) error {
197     // 从缓存中拉取该appId在指定数据集下可读写字段列表
198     appFieldInfo, ok := cache.GetTableItem(item.TBAppField, "c_appid|c_data_set", req.AuthInfo.AppId,
199         req.DataSetId).(item.AppFieldInfo)
200     if !ok {
201         log.Errorf("AppInfo is not exist, appId:%s, dataSetId:%d.", req.AuthInfo.AppId, req.DataSetId)
202         return NewAccessErr(protocol.EnumMediaErrorCode_RetNoAppField)
203     }
204
205     // 判断appId对字段是否有写权限，有则透传到数据适配层请求，没有则放入相应的FailList当中
206     for _, updateField := range req.UpdateFieldInfos {
207         field := updateField.FieldInfo
208         // 1. 检查是否有修改权限，*权限单独处理
209         if "*" != appFieldInfo.WriteListText && !item.CheckWPermission(req.AuthInfo.AppId, field.FieldName,
210             int(req.DataSetId)) {
211             retInfo.FailList[field.FieldName] = protocol.EnumMediaErrorCode_RetNoPermission
212             continue
213         }
214
215         // 2. 检查变更字段是否存在
216         fieldInfoMap, ok := cache.GetTableItem(item.TBFieldInfo, "c_interface_name|c_data_set",
217             field.FieldName, req.DataSetId).(item.FieldInfo)
218         if !ok {
219             retInfo.FailList[field.FieldName] = protocol.EnumMediaErrorCode_RetFieldInfoNotExist
220             continue
221         }
```

1 week ago

--story=857329939

【直播】直播聊天区视觉优化+轻互动自动飘

直播聊天区域增加自动飘心逻辑，每1秒触发一下飘心动画

- 将飘心的方法修改为传入播放时间，可以随机动画时长，手动点击时还是按照2.5秒的动画时长
- 由于一直会播放动画，可能会产生性能问题，目前采取初步的策略，viewWillDisappear 时停止播放动画，后续如果遇到性能瓶颈再尝试进行深度的优化

腾讯iWiki

Dashboard

Repositories

Explore

Help

Add

saas化项目

summer资料系统操作说明

标准化中台系统

统一接入平台使用指南

多图文素材

媒资分发系统接入手册

数据分发

排播系统手册

对外接口文档

视频接口

上架组排播

C端排播

行业排播

全网排播

中视频专辑字段添加及展示

中视频链路

待整理为服务运维文档

待整理为技术文档

待整理为知识库内容

待整理为规范

待改为 QA 手册（挪入对外...

不确定或功能性不强的文档

数据总线设计文档

对外接口文档

Created by samszhong(钟山), last modified by joyyhe(何俊艺) on 04-27 17:20

目录

1. 数据字段详情

2. 排播服务

2.1 获取排播信息接口

2.2 根据合集ID和集数获取对应排播时间

2.3 获取PC端每日排播数据

2.4 获取每日上新数据

2.5 获取APP端每日排播数据（首页专用）

2.5.1 六大品类（电影、电视剧、动漫、纪录片、综艺、少儿）数据

2.5.2 直播数据

2.6 获取专辑排播信息

3. 排播数据tdw表

1. 数据字段详情

相关字段及其对应取值详情：请参照文档<https://iwiki.woa.com/pages/viewpage.action?pageId=914470422>

2. 排播服务

北极星名字：trpc.video\_play.play\_plan.PlayPlan

Namespce:

测试：Development

开发：Development

13



# 2 Why?



## 为什么需要文档？

- 让代码和API更好理解，减少错误
- 明确的设计能提高实现的效率和质量
- 当步骤被清楚地列出时，流程规范更容易被遵循，减少出错（服务上线流程，迁移流程，错误排查流程，...）
- 经验的总结和传承（oncall手册，故障复盘）
- 新成员上手会节省很多时间
- ...

## 为什么我需要写文档？

高质量的文档对一个工程组织有巨大的好处，但他的好处是延后的，而且通常不会给作者本身带来直观的好处



## 为什么我需要写文档？

### 是一种长期投资

- 对文档的投入会随着时间的推移而得到回报
- 文档只用写一次，但会被读千百次
- 沉淀下来好的设计和经验，供后人借鉴，提升个人影响力
- 是一个正向循环，你也会用到别人的文档（写给未来的自己）

### 提供了维护路线图和历史记录

- 为什么做出这些设计决策？
- 为什么我们要以这种方式实现这段代码？
- 如果你两年后再看自己的代码，我为什么要以这种方式实现这些代码？

### 有助于做出更好的设计

- 把想法落实到文本上是对思维的一种辅助（讲不清楚就画个图）
- 有助于制定API。编写文档是确定API是否合理的最可靠方法之一，因为需要你去解释它，定义它。如果做不到这些那么你可能设计得不够好。
- 提供一个介质可以让大家充分讨论各种方案，集思广益



## 为什么写不好文档？

- 对作者的收益不够“直观”
- 文档被看作是一个额外的负担——需要维护的其他东西——而不是能使他们现有的代码维护更容易的东西
- 业务催的急，没时间写文档
- 认为写作是一种独立于编程的技能，自己的写作能力不够好（其实不然，你只需要跳出自己视角，从读者的角度看问题）
- 有限的工具支持，没有很好的集成到开发流程中
- 缺乏有效的激励机制（谷歌的升职委员会很看重候选人的设计文档，文档写的不好或太多 open question 可能会被驳回）



# 3 How?



## 把文档当作代码

- 就像我们需要学习不同的编程语言解决问题一样，文档没有什么不同：它是一种工具，用不同的语言编写，用于完成特定任务
- 与编程语言一样，它有规则、特定语法和格式，通常用于实现与代码中类似的目的：加强一致性、提高清晰度和避免（理解）错误
- 像代码一样，多份文档可能有重复的现象。这里需要指定规范文档，避免重复。（谷歌有个“go/links”的命名短链工具，便于确立权威的标准来源）
- 文档通常与代码紧密相连，所以应该尽可能地把它当作代码来对待
  - 置于源代码控制（source control）之下（谷歌使用g3doc的格式跟代码一起checkin）
  - 有明确的所有权，负责维护文档
  - 对修改进行审查（并与它所记录的代码一起变更）
  - 定期评估（谷歌里有个自动流程会定期file bug让owner更新文档）
  - ...

# 把文档当作代码

## Google g3doc

```
1 # Tutorials
2
3
4 ## MNIST For ML Beginners
5
6 If you're new to machine learning, we recommend starting here. You'll learn
7 about a classic problem, handwritten digit classification (MNIST), and get a
8 gentle introduction to multiclass classification.
9
10 [View Tutorial](../tutorials/mnist/beginners/index.md)
11
12
13 ## Deep MNIST for Experts
14
15 If you're already familiar with other deep learning software packages, and are
16 already familiar with MNIST, this tutorial will give you a very brief primer on
17 TensorFlow.
18
19 [View Tutorial](../tutorials/mnist/pros/index.md)
20
```

## 腾讯 Cherry Markdown

 Cherry Markdown

### 目标

规范媒资技术文档的编写和评审流程，提升文档质量，提高评审效率，促进信息的流通和分

### 流程规范

1. 按照媒资文档编写[规范](<https://iwiki.woa.com/pages/viewpage.action?https://iwiki.woa.com/pages/viewpage.action?pageId=760473609>)里。
2. 文档编写完成后找自己的组长进行初审。组长应在此环节把好质量关，确保文档的\*\*完  
档作者选定上下游相关系统的评审人加入当文档header里。
3. 将设计文档发给上个环节选定的相关评审人和TL进行线下批注审核（如果有特定的审核  
闲的时间能完整阅读文档并评论，同时敦促作者把需覆盖的内容落实到文档里，避免开会口  
避免带着未回答的问题进入最终评审。
4. 组长和TL根据设计的\*\*重要度、复杂度、项目的大小等\*\*决定是否需要大范围评审  
行开工排期。对于中大型或重要的设计，每周book一个线下会议室（可以找TPM帮忙，时长  
论，会上踊跃发言，集思广益，促进中心知识的流通和沉淀。
5. 评审会若通过，可以为文档打上【通过】的标签并开工。如果对设计仍有疑问，可以根



## 了解你的受众

- 工程师在写文档时犯的最主要的错误之一是只为自己写
  - 会认为读者跟你有相同的技能和背景知识，做出某些假设
- 开始写作之前，应该确定你的文件针对的受众和要满足目的
  - 说服决策者（High level文档）
  - 提供代码库说明（README）
  - 提供接口使用说明
  - 帮助半夜被page醒的oncall排查问题
  - ...
- 好的文档不需要润色或“完美”，而是以听众期望的声音和风格向他们写作
- 你的受众就站在你曾经站过的地方，但没有你的新领域知识。



## 受众类型

- 根据以下一个或多个标准，你可能需要面对多个受众
  - 经验水平（专家级程序员，或者甚至不熟悉语言的新手）
  - 领域知识（团队成员，或组织中只熟悉API端点的其他工程师）
  - 目的（需要用API来完成特定任务，或负责维护复杂实现的核心的软件专家）
- 需要以一种尽可能广泛地适用于不同受众群体的方式进行写作
  - 寻求一个平衡，没有silver bullet
  - 从受众的角度去思考
  - 精简，精简，精简



## 受众类型

另一个重要的受众区分是基于用户如何使用文档

- **寻求者 (seekers)**：知道他们想要什么，并且想知道他们所看到的是否符合要求
  - 对于这些受众来说，最关键的是**一致性**
  - 比如技术文档评审时某个对接模块的负责人就是寻求者，他们希望看到你的设计是否符合他们模块的要求，这里就需要上下文是一致的
  - 有经验的code reviewer会从代码里查找通用（设计）问题，这时候代码风格（比如命名）的一致性就很重要
- **浏览者 (stumblers)**：可能不知道他们到底想要什么，对正在使用的东西只有一个模糊的概念
  - 对于这些受众来说，最关键的是**清晰**
  - 提供概述或背景介绍（例如，在文件的顶部）
  - 确定文档何时不适合受众也很有用
    - 谷歌的很多文件都以 “TL;DR” (Too Long; Didn't Read) 开始
    - 如 “TL;DR: 如果你的工作不涉及到写技术文档，你现在可以停止阅读。”

## 了解文档类型

了解不同的类型，不要混合类型。

一个文档应该有一个单一的用途。



## 文档类型

软件工程师经常需要写的文档主要有几种类型：

1. 参考文档（包括代码注释）
2. 设计文档
3. 教程/手册
4. 概念文档（比如总体架构图）
5. 登录页

## 参考文档

负责解释清楚代码、API、CL等等的用途

- 代码注释
  - 解释清楚代码为什么这么写
  - 包注释
  - 类型注释
  - 函数注释
- API注释
  - 不需要涉及实现细节，只需要讲清楚怎么用
- CL描述
  - 摘要：概括变更内容点
  - 背景：此次变更的背景说明
  - 说明：补充说明原因和备注可能产生的影响

参考[Golang代码规范](#)，[如何做好Code Review](#)

## 设计文档

- 负责把一个设计讲清楚，并达成团队共识
- 一个好的设计文档应该包括
  - 设计目标 (goal)
  - 需求、背景和限制条件 (requirements & constraints)
  - 关键的设计决策，和各自的利弊权衡 (tradeoffs)
- 遵循最简原则 (less is more)
- 规范化评审流程
  - 基础质量关：初稿完成之后跟组长过审
  - 关联系统关：组长评审通过后跟相关领域的负责人过审
  - 更广受众关：最后设计评审会上跟更大的受众和主要负责人过审

参考[媒资文档编写规范](#)，[如何编写软件工程设计文档](#)，[媒资技术文档评审流程](#)



## 设计文档的反面模式 (Anti-pattern)

- 上来就提供“最优解”，没有背景介绍也没选项对比（所有的设计都有tradeoff，不存在绝对的最优）
- 目标跟具体设计内容不一致
- 对于自己的文档有太多情感投入，添加不必要的内容（文档是给受众写的，不是用来自我欣赏的）
- 为了显示自己的设计足够复杂刻意把文档写的很长
- 文档里直接贴实现代码（proto除外）
- 机械的去套模板，而不是根据实际情况更合理的组织文档，方便读者理解

## 教程/手册

如果我是个新人，我能否一步步走通？

### 示例：糟糕的教程

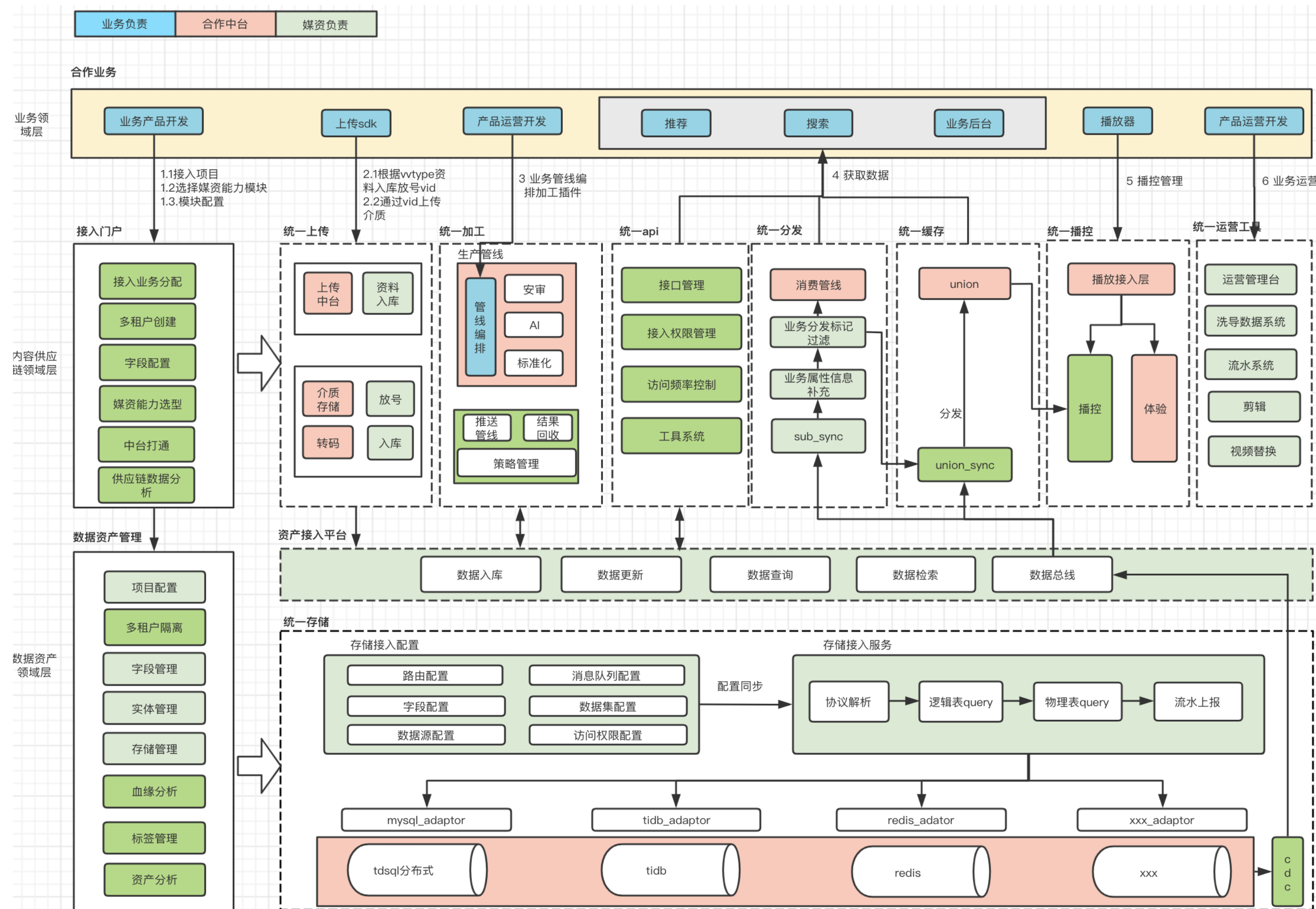
1. 从我们的服务器下载软件包，网址为 `http://example.com`
2. 将shell脚本复制到主目录
3. 执行shell脚本
4. foobar系统将与认证系统通信
5. 经过身份验证后，foobar将引导一个名为“baz”的新数据库
6. 通过在命令行上执行SQL命令来测试“baz”
7. 类型：创建数据库my\_foobar\_db;

### 示例：一个不好的教程会变得更好

1. 从我们的服务器下载软件包，网址为 `http://example.com`:
2. `$curl -I http://example.com`
3. 将shell脚本复制到主目录:
4. `$cp foobar.sh ~`
5. 在主目录中执行shell脚本:
6. `$cd ~; foobar.sh`
7. foobar系统将首先与身份验证系统通信。经过身份验证后，foobar将引导一个名为“baz”的新数据库并打开一个输入shell。
8. 通过在命令行上执行SQL命令来测试“baz”:
9. `baz:$CREATE DATABASE my_foobar_db;`

# 概念文档

- 提供清晰的系统概述，鸟瞰视角
- 是详细文档的补充而不是替代



# 登陆页

提供部门/团队的总体介绍

- 简洁
- 明确用途： 尽量避免混淆内外部信息





## 如何维护好文档

- 随着时间的推移，文档会变得陈旧、过时或（通常）被废弃
  - 实践之后发现有更好的方案
  - 当初的一些设想没能落实
  - 整个系统被重构了，但原始设计还被当做权威文档
  - ...
- 具体怎么做？
  - 不像代码可以跑测试发现问题
  - 到底应该多久更新一次？
  - 谁去改？



## 如何维护好文档

- **建立一套流程**
  - 谷歌里大部分文档用g3doc格式跟代码一起提交，并且附带以下元数据：  
freshness: { owner: `kluo` reviewed: '2022-05-27' }
  - 系统会定期（比如每三个月）向owner file bug，提醒owner查看以下文档是否update to date
  - 元数据会以 “Last reviewed by kluo on 2022-05-27” 的形式展现在文档开头，增加主人翁意识
- **提供反馈机制**
  - 发现问题及时file bug
  - 文档里的评论尽量及时处理
- **养成随手就改的习惯**
  - 新人如果跑哪个流程发现中间环节变了应该随手更新文档
  - 实现项目的时候如果发现有哪块跟设计有出入，记得随手更新文档（最起码应该在文档里标注“此设计已废弃”）

# 4 总结

## 回顾总结

### 什么算作文档（What）？

- 工程师为完成工作需要编写的每一个补充文本

### 为什么（我）需要写文档（Why）？

- 长期投资
- 有助于做出更好的设计
- 维护路线图

### 怎样写好文档（How）？

- 把文档当做代码
- 了解受众
- 了解文档类型
- 怎样维护好文档？



## 相关资料

- [Software Engineering at Google](#)
- [媒资文档编写规范](#)
- [媒资技术文档评审流程](#)
- [如何编写软件工程设计文档](#)
- [Golang代码规范](#)
- [如何做好Code Review](#)

# Thanks

