软件工程设计文档 模板

hankzheng@, yuankuisun@, more@（作者，可以多位联署，便于读者找到联系人）

2021/11/3（最后更新日期）

*（请生成副本后，删除斜体部分）*

*（除非有机密内容，请将文档设置为所有“企业成员可查看”，方便公司内交流）*

# 目标

* *用bullet points的形式列出设计的几个目标，确定设计的范围，例如：*
* *设计一个新的统一视频存储系统，以替换现有的多个老系统；*
* *改进登录系统的可靠性和速度*

## 非目标（可选）

*有时候，显式地说明设计的“非目标”（non-goals）更为重要。“非目标”不是简单的“目标”的否定句（如“系统不能crash”），而是一些可以作为目标而被刻意选择不作为目标的项目，例如当设计一个数据存储系统时，刻意将“满足所有ACID特性”作为“非目标”。*

# 背景（可选）

*为设计文档的目标读者提供理解详细设计所需的背景信息。*

*首先需要确定目标读者。例如：*

* *如果是一个新项目的总体设计，目标读者是部门Leads，那么背景可以包含现有类似项目的简短情况介绍，必要时可以链接到一个详细的调研文档；*
* *如果是一个项目中的子系统设计，目标读者是本团队成员，那么背景就可以包含该子系统在整个项目中的作用，及与相关子系统的关系。*

*一个好的设计文档应该做到，对目标读者是“自足的”（self-contained），即不需要阅读其他文档就可以理解文中的设计。但是，对于非目标读者，不需要也不可能做到“自足的”，例如当不期望其他团队的成员是目标读者时，就不用把整个系统的来龙去脉都作为背景写出来。*

*注意保持简洁，通常控制在一页以内。不要像写论文一样把上下五百年的背景都写进去，除非这个文档的目的就是调研。不要写成需求文档，可以链接到需求文档（如tapd）。只要提供足够目标读者理解详细设计的信息即可，如果认为目标读者了解足够背景，也可以省去该节。*

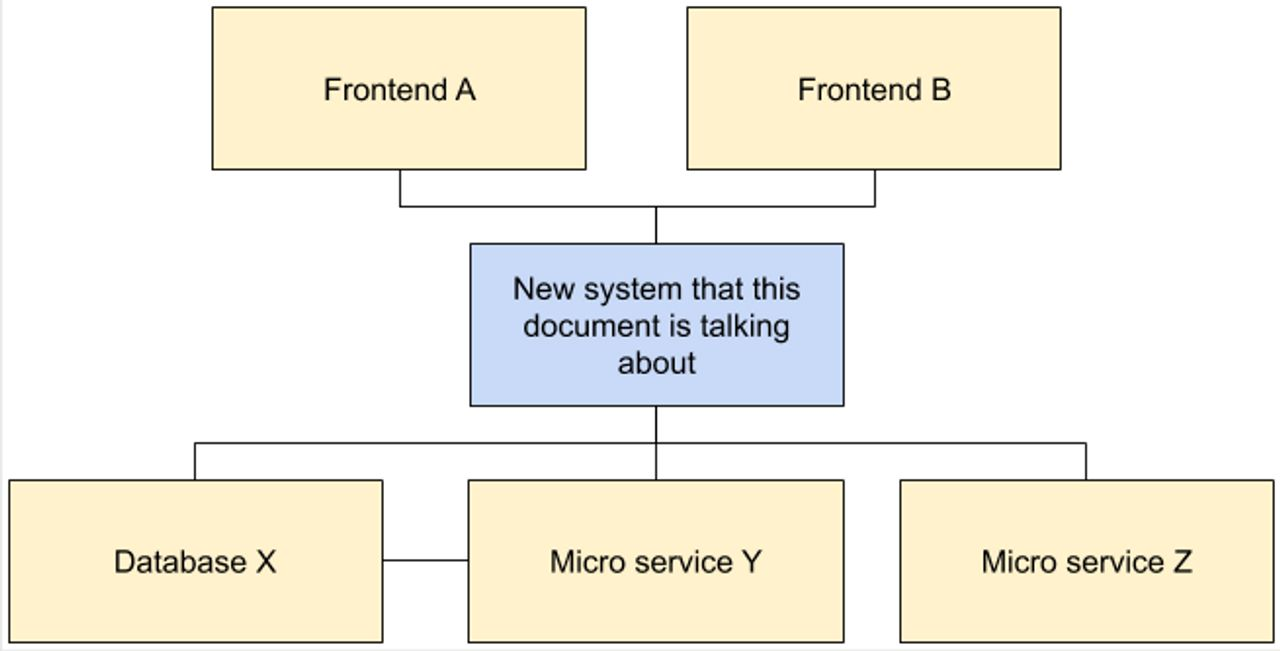
*通常来说背景里面应该包含项目立项的必要性，即 “为什么要做这个项目”。对必要性的理解会因人而异，显式地陈述出来更容易让读者和作者就项目的定位和理解达成一致，减少一些后续不必要的沟通成本。一些典型的原因包括：*

* *现有的系统构架太老，不足以高效的支撑后续的业务迭代，所以需要重构*
* *在xxx的大背景下我们需要一项yyy的能力，此项目正是提供这样一种能力*
* *某系统现有的SLA不符合未来的需求，需要改造升级*

# 总体设计

*概要地介绍设计的总体思路，帮助读者快速地了解整体设计。清楚地说明，给定目标和背景，做出了一个怎样的设计，以及为什么这个设计是最能满足目标的。好的总体设计应该做到，对于不关心设计细节的读者，读完这一节就足够了解整个设计。*

*通常，一个清晰的*[*系统上下文关系图*](https://zh.wikipedia.org/wiki/%E7%B3%BB%E7%BB%9F%E5%85%B3%E7%B3%BB%E5%9B%BE)*会非常有帮助，它可以使读者清晰地了解文中的新系统和已经熟悉的系统间的关系。它也可以包含新系统内部概要的组成模块。注意：不要只放一个图而不做任何说明，请根据上面小节的要求用文字描述设计思想。*

**

# 详细设计

*在这一节中，除了介绍设计方案的细节，还应该包括在产生最终方案过程中，主要的设计思想及考虑过的权衡（trade-off）。这一节的结构和内容因设计对象（系统，API，流程等）的不同可以自由决定，可以划分一些小节来更好地组织内容，尽可能以简洁明了的结构阐明整个设计。*

*有一些设计思考点应该贯穿于设计的始终，在以下每个小节中体现。包括但不限于：*

* *未来如何扩展（特别是水平扩展）？如果不支持，瓶颈是什么，解决方案是什么？*
* *是否需要自动弹性扩缩容？需要的话如何实现？*
* *是否需要支持多租户？需要的话，是否需要做隔离，方案是什么？*

*通常可以包含以下内容（注意，小节的命名可以更改为更清晰体现内容的标题）：*

## 各子模块的设计（可选）

*阐明一些复杂模块内部的细节，可以包含一些模块图、流程图来帮助读者理解。可以借助时序图进行展现，如一次调用在各子模块中的运行过程。*

## API接口（可选）

*如果设计的系统会暴露API接口，那么简要地描述一下API会帮助理解系统的边界。但是，也要避免将整个接口复制粘贴到文档中，因为在特定编程语言中的接口通常包含一些语言细节而显得很冗长，并且有一些细节也很很快变化。着重表现API接口跟设计最相关的主要部分即可。*

## 存储（可选）

*介绍系统依赖的存储设计。跟API接口的介绍一样，避免将所有数据定义（data schema）复制粘贴到文档中，因为这些会过于冗长而让读者忽略应该注意的地方。通常，把所有细节不加区别地写出来等于什么细节都没写，因为这使读者不能抓到其中的重点。因此，这里应该着重展现数据定义与整个设计最相关的部分，例如分析数据使用场景、对数据规模的假设、分库分表的假设、对扩展性（读量，存储增量等）地讨论等。如果是关系型数据，可以借助*[*ER图*](https://zh.wikipedia.org/wiki/ER%E6%A8%A1%E5%9E%8B)*的方式展现数据间的关系。*

# 考虑过的其他设计方案（可选）

*在介绍了最终方案后，可以有一节介绍一下设计过程中考虑过的其他设计方案（Alternatives Considered）、它们各自的优缺点和权衡点、以及导致选择最终方案的原因等。通常，有经验的读者（尤其是方案的审阅者）会很自然地想到一些其他设计方案，如果这里的介绍描述了没有选择这些方案的原因，就避免读者带着疑问看完整个设计再来询问作者。这一节可以体现设计的严谨性和全面性。*

# 系统 SLA（可选）

*注：如果该项目最终是提供一个服务，或者是一套端到端的系统则建议添加此部分。*

*Service Level Agreement（SLA）是系统（或 SOA 服务）的消费方和提供方之间契约的一部分。作为服务的所有方，必须公布系统的 SLA。这样该服务的消费方才能依赖其提供的 SLA 来设计和定义他们的系统。不同的 API 可以有不同的 SLA，但是要公开透明。同时在设计系统的时候也要考虑到如何测量 SLA。*

*我们应该从以下几个方面来衡量 SLA：*

* *准确性（Accuracy）：最大错误率是多少（高于0.01%的话请提供合理解释）？*
* *系统容量（Capacity）：能稳定支持的最大吞吐量是多少（请求/秒）？*
* *延迟（Latency）：响应延迟的中位数是多少（高于100ms请提供合理解释）？响应延迟一分钟里的 P99是多少（高于1s请提供合理解释）？*
* *可用性（Availability）：可以保证的系统可用在线比例是多少（低于99.95%请提供合理解释）？*

*在提供了SLA的同时，还要思考如何验证SLA。比如可以通过压力测试来保证系统在既定的吞吐量下能够稳定运行并满足给出的SLA。这里可以给出详细的压力测试方案和目标。*

# 系统依赖（可选）

*显式的列出系统的依赖有利于让读者更清晰的理清这些关系，而且还会指导作者考虑系统在不同的依赖出现问题时的响应机制。*

*可以以表的形式列出该系统所依赖的其他系统、服务、或者组件。*

|  |  |
| --- | --- |
| *系统依赖* | *如果依赖变的不可用或者性能下降，对系统的影响是什么？如何处理这些影响？* |
| *依赖 #1* |  |
| *依赖 #2* |  |
| *依赖 #3* |  |

# **监控和告警**（可选）

*该系统的整个链路中有哪些重要指标需要上报？是系统指标还是业务指标？是上报到哪？其中又有哪些需要设置监控和告警？*

*建议在系统设计阶段就清楚定义这些信息，防止在系统上线后再来不系统地添加各种上报指标。换个角度来说，如果在系统设计阶段不知道要追踪和监控哪些指标，则说明我们对系统的认知还不全面，会有设计不周的风险。*

* *指标*
  + *哪些关键指标、需要在哪被上报？*
  + *指标使用量级的增长是否和业务的增长相匹配？如果不是，是更快还是更慢？*
  + *大约预估每秒会上报多少指标？*
* *监控*
  + *哪些指标要通过看板来监控？*
  + *这些看板是你们团队自己用，还是会有外部团队也来使用？*
  + *从监控系统可用性的角度来看，除了用这些看板还有别的方式来监控吗？*
* *告警*
  + *哪些指标要设置告警？*
  + *如果某个指标异常或者服务挂了，需要通知哪个具体团队？*
  + *在告警被触发的时候，有没有对应的日志来帮助定位问题？*

# **系统迁移影响（可选）**

*当这个新系统上线的时候，是否涉及到老系统的迁移或者下线？如果是，迁移方案是什么。*

*新系统的上线往往伴随着老系统的下线或者流量迁移。而这个迁移方案也应该是设计新系统时候应该考虑的一部分。不然可能会陷入新系统开发完成，但是没有合理的迁移方案，或者需要因为迁移而再回去修改调整新系统的尴尬境地。*

# 其他关注点（可选）

*如果设计涉及一些公司要求的横切关注点（cross-cutting concerns）,例如安全（security）、隐私（privacy）、可用性（accessibility）、等，可以在这里简要地讨论。当部署（deployment）、容灾（*过载保护、有损降级*、*接口熔断*、*轻重分离*等）、运维等有一些值得讨论的问题时，也可以包含在这里。*

# 附录（可选）

*指向工蜂工程的链接（以形成交叉引用，方便找到相关工程；可以在工程开发后加上）*