

Software Architecture Document

Version <1.0>

Revision History

|  |  |  |  |
| --- | --- | --- | --- |
| **Date** | **Version** | **Description** | **Author** |
| < yyyy-mm-dd > | <x.x> | <details> | <name> |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

目 录

1. 文档简介 4

1.1 文档目的 4

1.2 文档范围 4

1.3 定义、缩写词和缩略语 4

1.4 参考资料 4

2. 架构描述方式 4

2.1 架构视图阅读指南 4

2.2 图表与模型阅读指南 5

3. 架构设计目标 5

3.1 关键功能 5

3.2 关键质量属性 5

3.3 业务需求和约束因素 6

4. 架构设计原则 6

4.1 架构设计原则 6

4.2 备选架构设计方案及被否原因 6

4.3 架构设计对后续工作的限制（详设，部署等） 6

5. 逻辑架构视图 7

5.1 职责划分与职责确定 7

5.2 接口设计与协作机制 8

5.3 重要设计包 10

6. 开发架构视图 11

6.1 Project划分 11

6.2 Project 1 11

6.2.1 Project目录结构指导 12

6.2.2 程序单元组织 12

6.2.3 框架与应用之间的关系（可选） 12

6.3 Project 2…… 13

6.4 Project n…… 13

7. 运行架构视图 13

7.1 控制流组织 13

7.2 控制流的创建、销毁、通信 14

7.3 加锁设计 14

8. 物理架构视图 14

8.1 物理拓扑 14

8.2 软件到硬件的映射 15

8.3 优化部署 16

9. 数据架构视图 16

9.1 持久化机制的选择 17

9.2 持久化存储方案 17

9.3 数据同步与复制策略 17

10. 关键质量属性的设计原理 17

# 文档简介

[帮助读者对本文档建立基本印象，并为阅读后续内容扫清障碍。]

## 文档目的

[文档目的，非项目目的。否则造成同一项目多个文档之间的内容重复，不利于文档维护。本小节应指明文档针对的读者对象，最好列出各种读者角色，并说明每种读者角色应该重点阅读的章节。]

## 文档范围

[文档的Scope，非项目的Scope。否则造成同一项目多个文档之间的内容重复，不利于文档维护。]

## 定义、缩写词和缩略语

[集中列举文档中的定义、缩写词和缩略语。]

## 参考资料

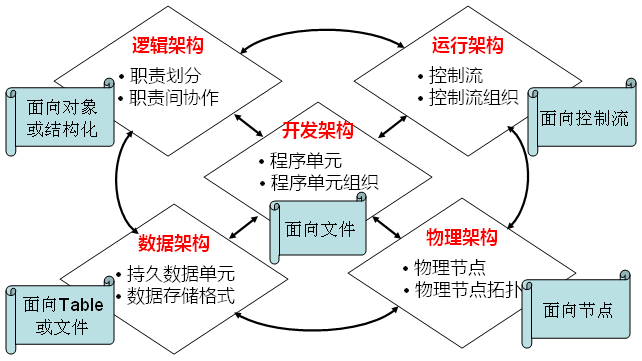
[本项目经审核的计划书、合同、上级批文；本项目的其他已发表文件；本文档引用的文件资料，如软件开发标准。具体而言，应包括参考资料的题目（必须）、编号、版本号（必须）、发表日期、发布方，必要时还可以说明如何使用这些资料。]

# 架构描述方式

[为了让读者更好地理解《架构文档》，在本节应当说明文档涉及的架构视图，并指明为了描述设计决策用到了哪些图表和模型。]

## 架构视图阅读指南

[以多视图的方式来组织《架构文档》是大势所趋。ADMEMS推荐的是经过优化的5视图方法，如下图所示。]



## 图表与模型阅读指南

[对后续文档内容中所用到的建模语言（例如UML）、表格（例如目标-场景-决策表）等进行说明。]

# 架构设计目标

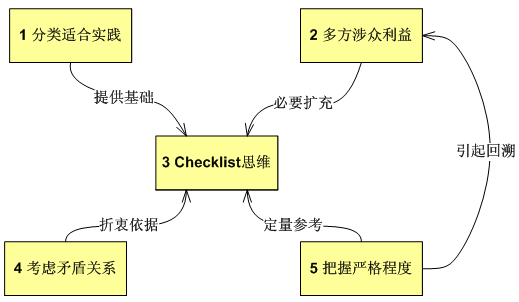
[功能、质量、约束，一个都不能少。]

## 关键功能

[对架构设计至关重要的功能，包括如下4类：核心功能、必做功能、高风险功能、独特功能。所谓独特功能，指这个功能覆盖了上述3类功能没有涉及到的职责。]

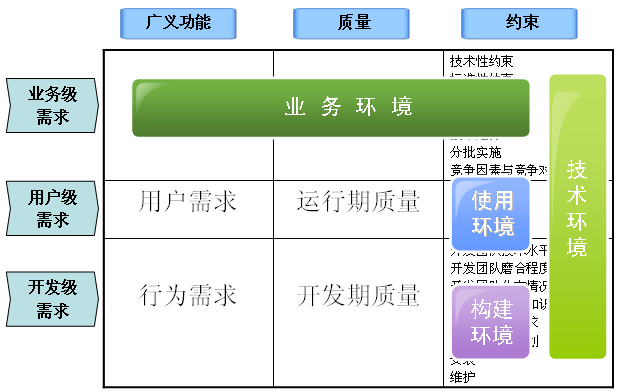
## 关键质量属性

[人之所以痛苦，很多时候是因为追求错误的东西。下图是ADMEMS方法确定关键质量的5大原则的整体思路图。]



## 业务需求和约束因素

[ADMEMS方法创造性地提出约束需求的4大类型，这是一种极为实用的分类方式。特别是业务需求对架构设计而言是一种约束的观点，解决了很多架构师的现实困惑。下图标明了4类约束在“需求层次-需求方面矩阵（又称ADMEMS矩阵）”中的位置，可以帮助我们理解产生约束需求的根源。]



# 架构设计原则

[投标时经常讲“架构设计原则”，但到了《架构文档》，这些着眼大局的考虑却“丢了”。ADMEMS方法推荐的本文档模板，认为应当把它们“找回来”。]

## 架构设计原则

[着重描述重大的权衡取舍考虑。]

## 备选架构设计方案及被否原因

[在概念架构一级，对备选架构设计方案进行描述，并阐述它们未被采用的原因。这有利于团队了解当前架构设计方案的来龙去脉，提高团队对当前架构设计方案的认可度。]

## 架构设计对后续工作的限制（详设，部署等）

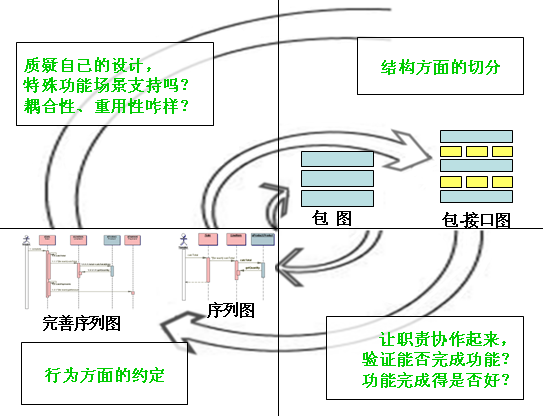
[架构设计不仅应该包含“指导”，也应该包含重要的“限制”。例如，一份只是说明“性能和可扩展性都重要”的《架构文档》，实际上忽视了“可扩展性和性能之间存在的矛盾关系”。此时，最有效的办法就是在《架构文档》中明确说明“任何提升可扩展性的架构设计和详细设计，都应通过架构团队的评审才能引入，以确保性能目标不受重大影响”。]

# 逻辑架构视图

[关注点：此架构设计视图的关注点是职责划分。]

[注意：逻辑架构视图无疑是最重要的，但同时也应避免“架构 = 模块 + 接口”等以偏概全的认识。]

[参考：任何复杂系统的架构设计都不是一蹴而就的，所以架构师需要理性思维过程的指导。针对逻辑架构设计这个关键环节，《一线架构师实践指南》一书给出了2条建议：一是“以质疑驱动的螺旋思维”，二是相对分离地考虑“结构方面的切分”和“行为方面的定义”。下图所示即为ADMEMS方法推荐的逻辑架构设计理性思维过程。]

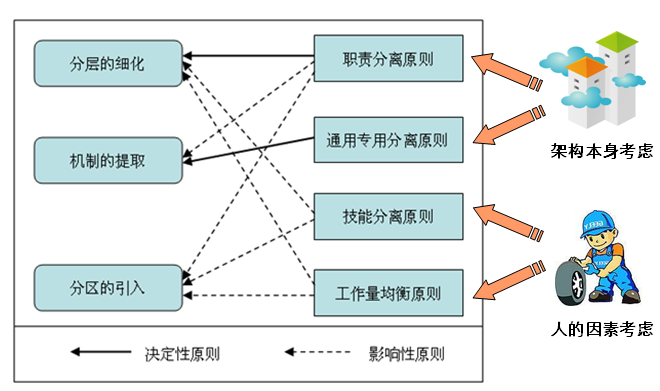


## 职责划分与职责确定

[内容：将系统切分成更小的单元，并明确这些单元的职责。具体而言，职责单元可以是层、子系统、模块、关键类等。]

[意义：一句话，职责划分不合理，功能和质量都会受到影响。也就是说，功能需求和质量需求无一不和职责划分相关：一方面，每个功能都是由一条职责协作链完成的；另一方面，职责划分方式也影响着质量，于是需要职责模型针对特定质量属性要求做出相应调整和优化。很多人认为架构设计就是职责划分的艺术，虽略显片面，但足以表明职责划分的重要性。]

[参考：基于对业界大量案例的研究，ADMEMS方法梳理出了“模块划分的3种必用手段”，如下图所示，更多内容可参考《一线架构师实践指南》一书。]

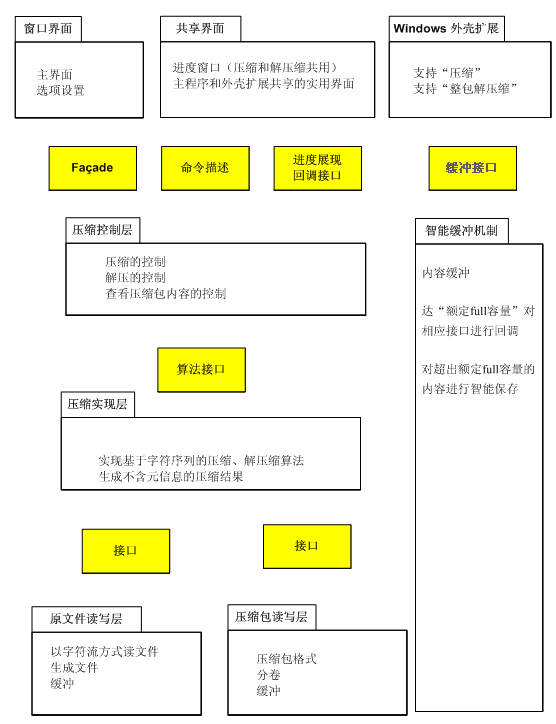


## 接口设计与协作机制

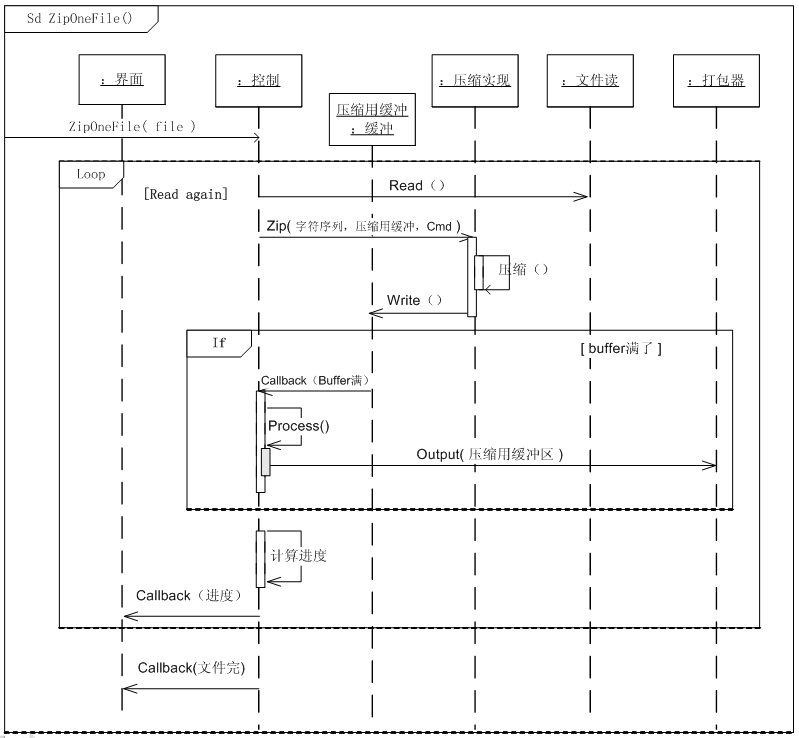
[内容：本节描述接口的定义，以及协作的方式和规范。]

[意义：恰恰是因为有了各模块之间“未来合作的契约”，分头开发各模块才有了基本保证。]

[参考：ADMEMS方法推荐利用“包-接口”图，来识别接口。下图为一个“包-接口”图的示例。]



[参考：ADMEMS方法推荐使用序列图，建议少用、甚至杜绝使用协作图。下图为一个序列图的示例。]

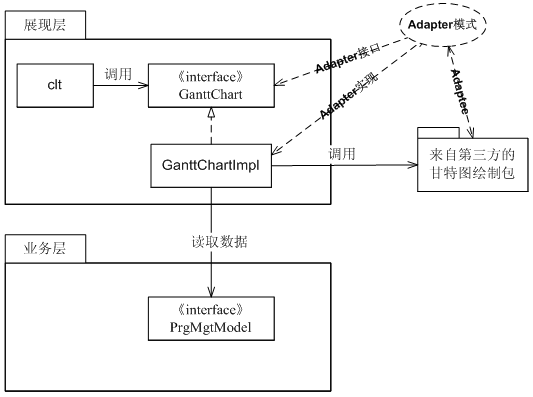


## 重要设计包

[内容：对重要子系统的设计进行“灰盒”级描述。]

[意义：“每个子系统在架构设计中都应保持黑盒子”的观点，过于理想化了。对于业务层、通用协作机制而言，经常需要在架构设计期间就引入“灰盒”级描述。]

[参考：类图和灰盒包图，在本节中较多出现。下图为一灰盒包图示例。]



# 开发架构视图

[关注点：此架构设计视图的关注点是程序单元组织。]

[注意：此架构设计视图是必须的、不应“剪裁”掉的。但实际情况却是，很多架构师不关注开发架构视图，导致很多程序开发人员抱怨“架构师就知道高来高去，架构对编程工作没什么指导性”。]

## Project划分

[内容：本节说明整个系统将划分成哪几个Project来开发，其中，Project指开发环境所感知到的“工程”。]

[意义：基本好处是，有利于开发的组织；而对一些大型的集成系统而言，由于同时涉及了Web应用、桌面应用、嵌入式应用等软件形态，所以此时Project划分其实是不得不做的；最后，我们推荐核心代码应主动地切分到单独的Project以进行独立的软件配置管理（SCM），以降低核心代码外泄的风险。]

[参考：Project划分必然是属于“架构设计”的工作，严格来讲仅靠“需求分析”划分的业务域（Business Area）直接映射到Project经常意味着工作内容的遗漏。其实，业界不少有见地的专家已经认识到WBS（工作分解结构）做得太早太草率危害很大，就与“Project划分不到位”不无关系。]

## Project 1

[内容：对Project划分后的每个Project进行目录结构、程序单元组织、框架与应用关系的说明。]

### Project目录结构指导

[内容：关于该Project一级目录、二级目录等基本目录结构的约定。]

[意义：为团队并行开发提供必要基础，让不同程序小组看到自己应该负责的程序目录。]

[参考：不要把所有程序目录的约定都定义得太细，否则这份《架构文档》就要天天更新了。]

### 程序单元组织

[内容：源码、程序库、框架、目标码等类型程序单元之间的编译依赖关系。]

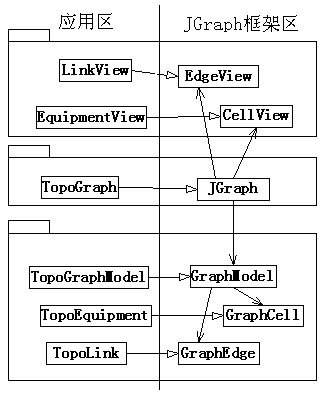
[意义：或许有人认为这没什么技术含量，但架构设计本来就不是只关心技术含量最高问题的。君不见，很多软件工程师跳槽到新的企业之后，竟然连一个能正常编译源码的开发环境都建不起来——其实，他们“不知道Project所依赖的Library有哪些”是其中重要原因——这本应在《架构文档》中给出明确描述的。]

### 框架与应用之间的关系（可选）

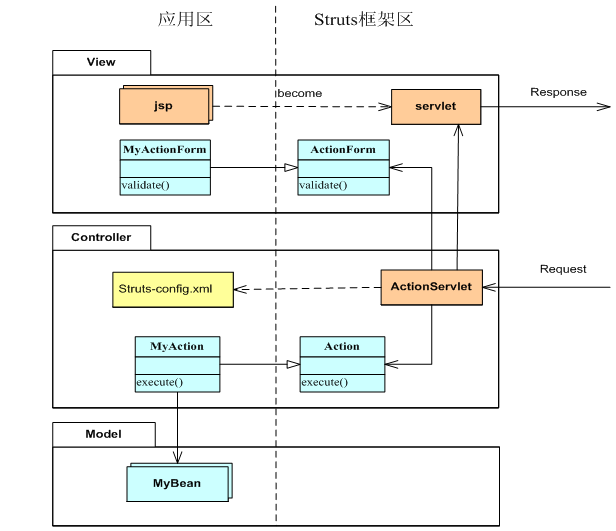
[内容：框架（Framework）。]

[意义：既然不适用Framework的开发越来越少了，既然程序员犯的很多错误都和对Framework理解不到位有关，架构师就有责任明确说明Framework和待开发系统之间的关系。]

[参考：下图描述了JGraph框架和待开发应用的关系。]



[参考：下图描述了Struts框架和待开发应用的关系。]



## Project 2……

[内容：对Project划分后的每个Project进行目录结构、程序单元组织、框架与应用关系的说明。]

## Project n……

[内容：对Project划分后的每个Project进行目录结构、程序单元组织、框架与应用关系的说明。]

# 运行架构视图

[关注点：此架构设计视图的关注点是控制流组织。]

[注意：进程和线程是广为人知的控制流实现技术，但在架构设计思维当中，对于系统软件和嵌入式软件极为重要的中断服务程序也是控制流，这样利于架构师统一利用不同控制流手段设计并行和并发。]

## 控制流组织

[内容：控制流有哪些，每条控制流各是何种形式（例如进程、线程、中断服务程序），哪些软件单元是控制流的起点，整条控制流中分别调用了哪些软件单元。]

[意义：这是对系统运行时结构的刻画，主要反映系统的动态结构。]

## 控制流的创建、销毁、通信

[内容：描述进程、线程和中断服务程序的创建和销毁，以及多条控制流之间的通信关系的定义。]

[意义：一旦引入了多条控制流，附加工作就产生了——此时控制流的创建和销毁、以及控制流之间的通信关系往往是必须考虑的。]

## 加锁设计

[内容：系统中有多条控制流在同时运行的情况下，一个经典问题是多于一条控制流可能会同时修改某些数据结构，而造成数据的不一致。为此，架构师需要关注加锁设计，合理引入临界区或同步机制。]

[意义：加锁设计事关系统的正确性。值得注意的是，忽略加锁设计造成的问题往往以“不易重现的Bug”的形式出现，困惑的程序员会对测试人员说，“你看你报的Bug在我机器上根本就不存在呀”。]

[参考：对通用组件、通用模块的设计而言，加锁设计应予以专门关注，思维要点是研究未来通用模块的各种可能使用场景。]

# 物理架构视图

[关注点：此架构设计视图的关注点是物理节点（Node）分布，以及软件到硬件的具体映射关系。]

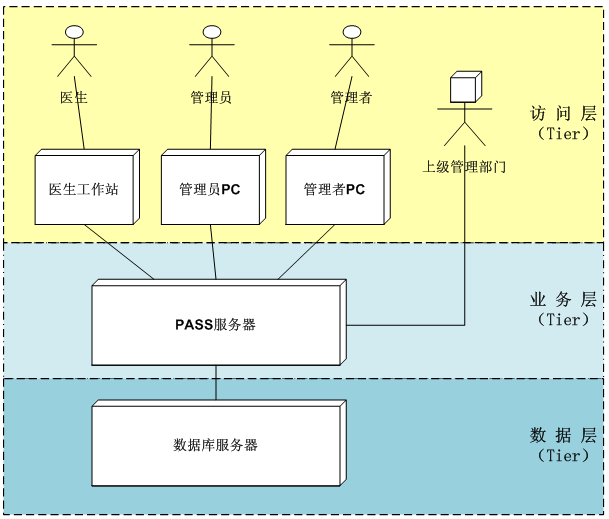
[注意：物理节点即可以是PC机或服务器，也可以是单片机、单板机或专用机，从而物理架构视图既适用于描述企业信息系统，也适合于描述嵌入式软件系统。]

## 物理拓扑

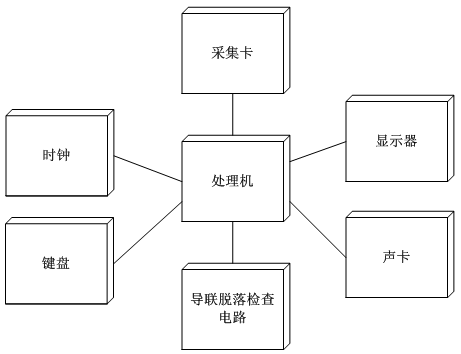
[内容：一为硬件选型，二为硬件之间的拓扑连接关系。]

[意义：对于分布式系统的设计，此节极为重要、而且是必须的。]

[参考：下图是某企业级系统的物理拓扑图。]



[参考：下图是某嵌入式系统的物理拓扑图。]

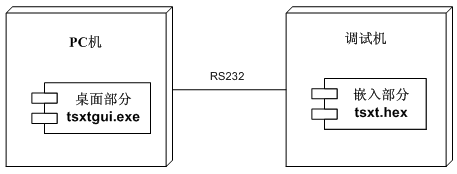


## 软件到硬件的映射

[内容：明确每个物理节点上有哪些（一到多个）软件的目标单元，并说明具体的“映射方式”是安装、是部署、还是烧写、抑或是下载。]

[意义：如果把此节漏了，就无法表明本文档的主题——软件系统——和上述硬件、硬件拓扑的关系。]

[参考：下图所示为设备调试系统中，软件到硬件的映射关系。]

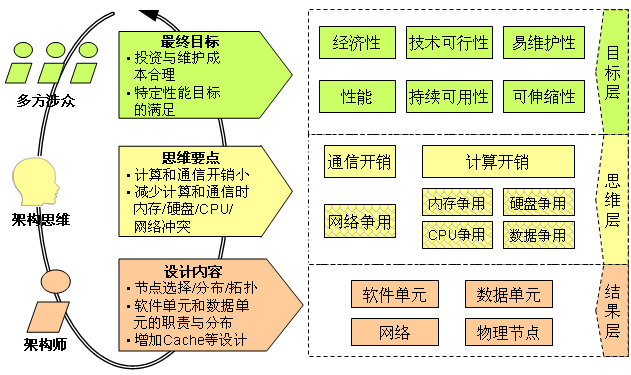


## 优化部署

[内容：为达降低成本、提高性能和可靠性等等目标，应特别关注的部署考虑。]

[意义：物理架构设计的优劣，造成的成本差异和质量差异，可能是天壤之别。所以必须重视。]

[参考：下图展示的，是ADMEMS方法重点推荐的“物理架构设计思维要点”，更多内容可参考《一线架构师实践指南》一书。]



# 数据架构视图

[关注点：此架构设计视图的关注点是持久化。具体而言，场景化可以借助扁平文件、关系数据库、实时数据库、Flash等方式中的一种或多种完成。]

[注意：本视图单独归档时，请在此节注明其文档名称等信息。]

## 持久化机制的选择

[内容：如下持久化机制的一种或多种：扁平文件、关系数据库、实时数据库、Flash。]

[意义：不要假设在你的系统中，持久化只需一种机制；随着如今的系统变得越来越复杂，我们经常需要综合利用不同持久化机制。]

## 持久化存储方案

[内容：持久化数据的格式定义。]

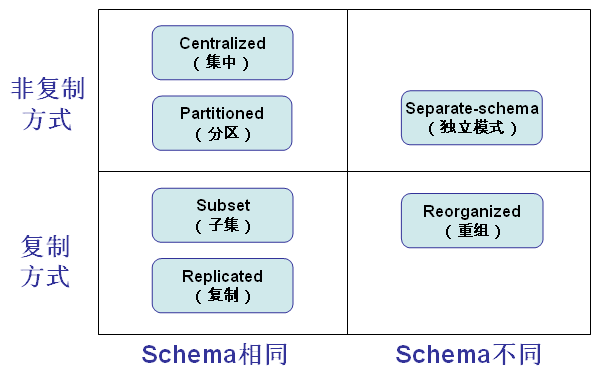
[意义：统一定义表、文件格式、Flash数据结构等内容。]

## 数据同步与复制策略

[内容：由于数据分布所引起的，包含数据分布、同步、复制等内容的重要设计决策。]

[意义：在数据分布的情况下，此节为必须。]

[参考：在实际中，数据分布的策略绝大多数情况下不会超越下图所示的6种手段，更多内容可参考《一线架构师实践指南》一书。]



# 关键质量属性的设计原理

[内容：因软件系统的不同，性能、安全性、可伸缩性、互操作性、可扩展性、可测试性、可重用性、可维护性等质量属性，都可以是本系统的关键质量属性。本文档的前面部分已经涉及了关键质量属性的设计决策，而本节更集中、更全面地描述这些架构设计决策，并且阐述“为什么”这么设计。]

[意义：只描述架构设计决策本身，不利于读者理解“为什么”这么设计。而且，描述设计原理有利于在整个软件企业层面促进团队的架构设计能力。]

[参考：关于描述“为什么”这么设计，目标-场景-决策表是此方面的卓越工具。下图为示例，更多内容可参考《一线架构师实践指南》一书。]

