|  |  |
| --- | --- |
| 产品名称Product name | 密级Confidentiality level |
|  |  |
| 产品版本Product version | Total 3pages 共3页 |
|  |

架构设计说明书

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 拟制:  Prepared by |  |  | 日期：  Date | yyyy-mm-dd |
| 审核:  Reviewed by |  |  | 日期：  Date | yyyy-mm-dd |
| 批准:  Granted by |  |  | 日期：  Date | yyyy-mm-dd |

元鼎智能创新有限公司

版权所有 侵权必究

All rights reserved

修订记录Revision record

|  |  |  |  |
| --- | --- | --- | --- |
| 日期  Date | 修订版本Revision version | 修改描述  change Description | 作者  Author |
| yyyy-mm-dd | 1.00 | 初稿完成 initial transmittal |  |
| yyyy-mm-dd | 1.01 | 修改XXX revised xxx |  |
| yyyy-mm-dd | 1.02 | 修改XXX revised xxx |  |
| …… | …… | …… |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

目 录Table of Contents

[1 文档简介 4](#_Toc2031)

[1.1 文档目的 4](#_Toc32607)

[1.2 文档范围 4](#_Toc21531)

[1.3 定义、缩写词和缩略语 4](#_Toc27701)

[1.4 参考资料 4](#_Toc19113)

[2 架构设计目标 4](#_Toc15856)

[2.1 关键功能 5](#_Toc28512)

[2.2 关键质量属性 5](#_Toc21431)

[3 架构设计原则 5](#_Toc11636)

[4 架构描述方式 5](#_Toc31203)

[4.1 架构视图说明 6](#_Toc32208)

[4.2 图表与模型阅读指南 6](#_Toc5070)

[5 场景架构视图 6](#_Toc5170)

*[5.1 场景视图原则](#_Toc5340)* [6](#_Toc5340)

[6 逻辑架构视图 7](#_Toc29320)

[6.1 逻辑视图原则 7](#_Toc19192)

[6.2 逻辑视图分层 8](#_Toc31741)

[6.2.1 应用层 8](#_Toc19937)

[6.2.2 业务层 8](#_Toc11660)

[6.2.3 中间层 8](#_Toc16889)

[6.2.4 系统层 9](#_Toc4120)

[7 开发架构视图 9](#_Toc21110)

[7.1 开发视图原则 9](#_Toc22619)

[8 运行架构视图 10](#_Toc9760)

[8.1 运行视图原则 10](#_Toc2853)

[8.2 控制流组织 10](#_Toc27398)

[8.3 控制流的创建、销毁、通信 10](#_Toc28870)

[8.4 加锁设计 10](#_Toc14541)

[9 物理架构视图 11](#_Toc14488)

[9.1 物理视图原则 11](#_Toc18053)

[9.2 物理拓扑 11](#_Toc30209)

[9.3 软件到硬件的映射 12](#_Toc6192)

[9.4 优化部署 12](#_Toc10859)

[10 关键质量属性的设计原理 13](#_Toc10580)

[11 其他说明 13](#_Toc27643)

[12 附录 13](#_Toc1363)

架构设计说明书

关键词Key words： 能够体现文档描述内容主要方面的词汇。

摘 要Abstract：

缩略语清单List of abbreviations：<对本文所用缩略语进行说明，要求提供每个缩略语的英文全名和中文解释。Describe abbreviations in this document, full spelling of the abbreviation and Chinese explanation should be provided.>

| Abbreviations缩略语 | Full spelling 英文全名 | Chinese explanation 中文解释 |
| --- | --- | --- |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |

# 文档简介

文档的简介应提供整个软件构架文档的概述。它应包括此软件构架文档的目的、范围、定义、首字母缩写词、缩略语、参考资料和概述。

## 文档目的

从架构方面对系统进行综合概述，其中会使用多种不同的架构视图来描述系统的各个方面。

## 文档范围

说明该文档的使用范围和影响范围

## 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

完整地列出此软件架构文档中所引用的文档及材料，要有标题、报告号，日期及出版信息

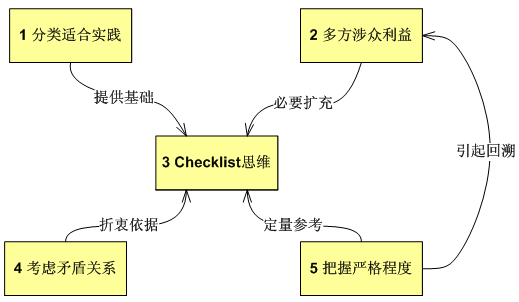
# 架构设计目标

功能、质量、约束（来源历史问题、专利、技术分析等）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



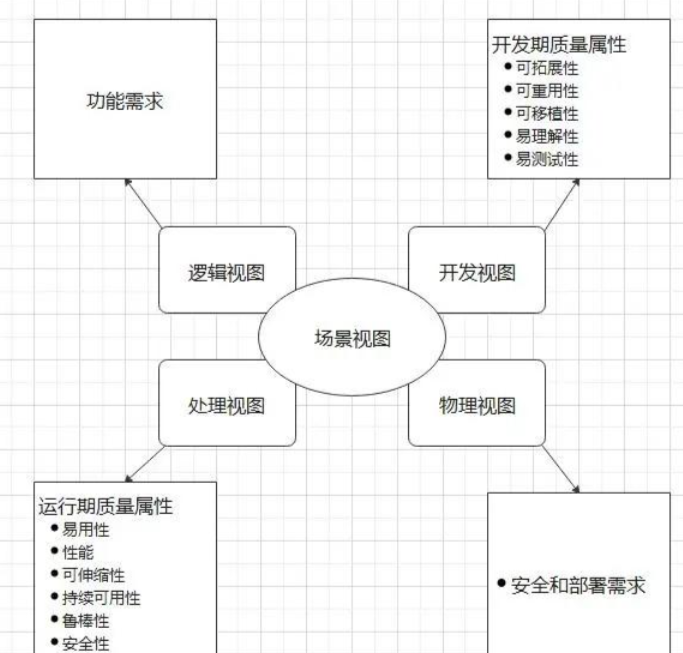
# 架构设计原则

架构设计原则是一系列指导软件系统架构设计的最佳实践，这些原则旨在增强系统的可维护性、可扩展性和可靠性。

# 架构描述方式

为了让阅读者更好地理解《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

## 架构视图说明



## 图表与模型阅读指南

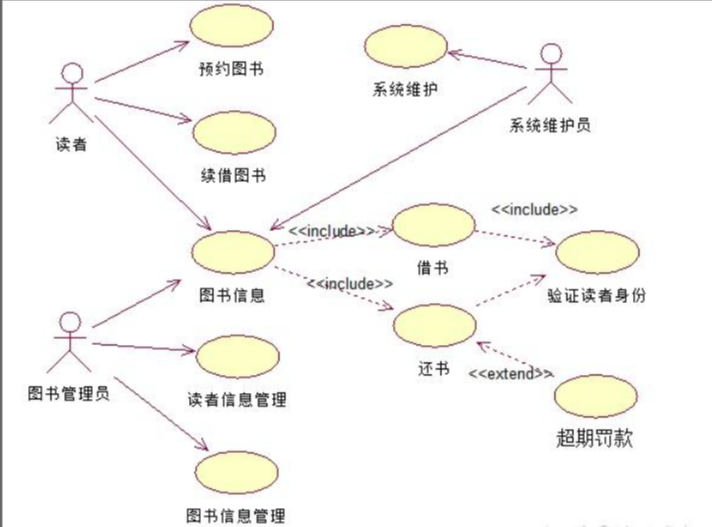
*在此简单介绍软件架构的整体情况，包括场景视图、逻辑视图、处理视图、开发视图和部署视图的简单介绍。另外，简要介绍各种视图的作用和针对的用户*

# 场景架构视图

## *场景视图原则*

*场景视图用于描述系统的角色与行为用例间的关系，反映系统的最终需求和交互设计，通常由用例图表示。*

*通过用例视图的设计过程，可以正确的识别系统的用户和其它系统（Actor）、系统边界（Boundary）和用例（Use Case），并对系统的功能场景进行充分的分析，以确定系统提供的功能可以满足用户需求，而其它4个视图都是需要围绕着这些信息进行设计。*



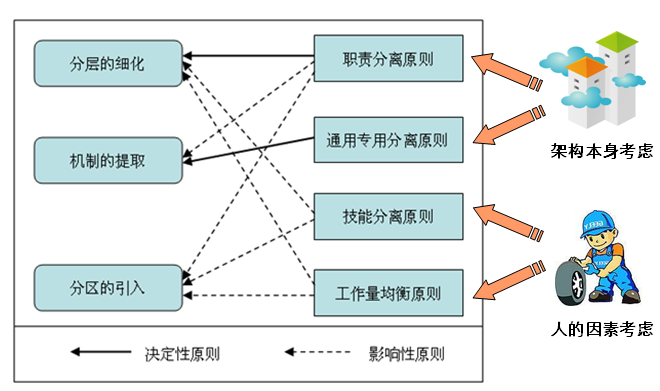
# 逻辑架构视图

*也称为设计师视图，主要关注系统的功能性需求，展示的是系统的设计类、接口以及它们之间的关系。即系统给用户提供哪些服务；以及描述系统软件功能拆解后的组件关系、组件约束和边界，反映系统整体组成与系统如何构建的过程。这一视图通常反映了面向对象设计的方面，强调内部对象和组件如何协作*

## 逻辑视图原则

*在此用图形的方式显示此软件架构的分层、子系统、包之间的关系，并且加入必要的文字说明，特别是架构的分层、机制和模式的说明*

*在此，还需要将和本系统有接口关系的其它系统都表示出来，这样可以清楚的看到本系统和其它系统的接口逻辑视图，并且还需要将在本系统中负责接口的子系统表示出来*



## 逻辑视图分层

*在此说明逻辑分层的每一层和每一层中的包、子系统。对于每个重要的包，都用一个小节来加以说明，其中应包括该包的名称、简要说明以及显示该包中所有重要的类和包的图。*

*对于该包中的每个重要类，应包括其名称、简要说明，还可选择包括对其部分主要职责、操作和属性的说明*

*下面预先设置应用层、业务层、中间层和系统层，实际使用中可以根据需要进行修改*

### 应用层

*在此说明软件架构子系统的高层组织，是系统面向外部（主角）、面向功能（用例）的子系统。在这一层中的子系统，要根据中间层的约定继续分区（tier）*

### 业务层

*在此说明软件架构子系统的高层组织，是应用层子系统公用的、为应用层子系统提供服务的、和业务相关的子系统。在这一层中的子系统，要根据中间层的约定继续分区*

### 中间层

*在此说明软件架构子系统的底层组织，是面向系统内部（技术）、面向非功能（补充规约）的子系统（框架、包、机制、模式）。在这一层的子系统，通常包括分区、支持分区的框架、架构模式、设计机制（事务、存储等）、公用元素及服务（邮件、翻页、列表控件、树控件、按钮控件等）等*

### 系统层

*在此说明软件架构的操作系统、运行平台（应用服务器、数据库等）、开发语言、第三方技术（比如GIS、CORBA、J2EE、COM、.net）等*

# 开发架构视图

## 开发视图原则

*开发视图用于描述系统的模块划分和组成，以及细化到内部包的组成设计，服务于开发人员，反映系统开发实施过程。*

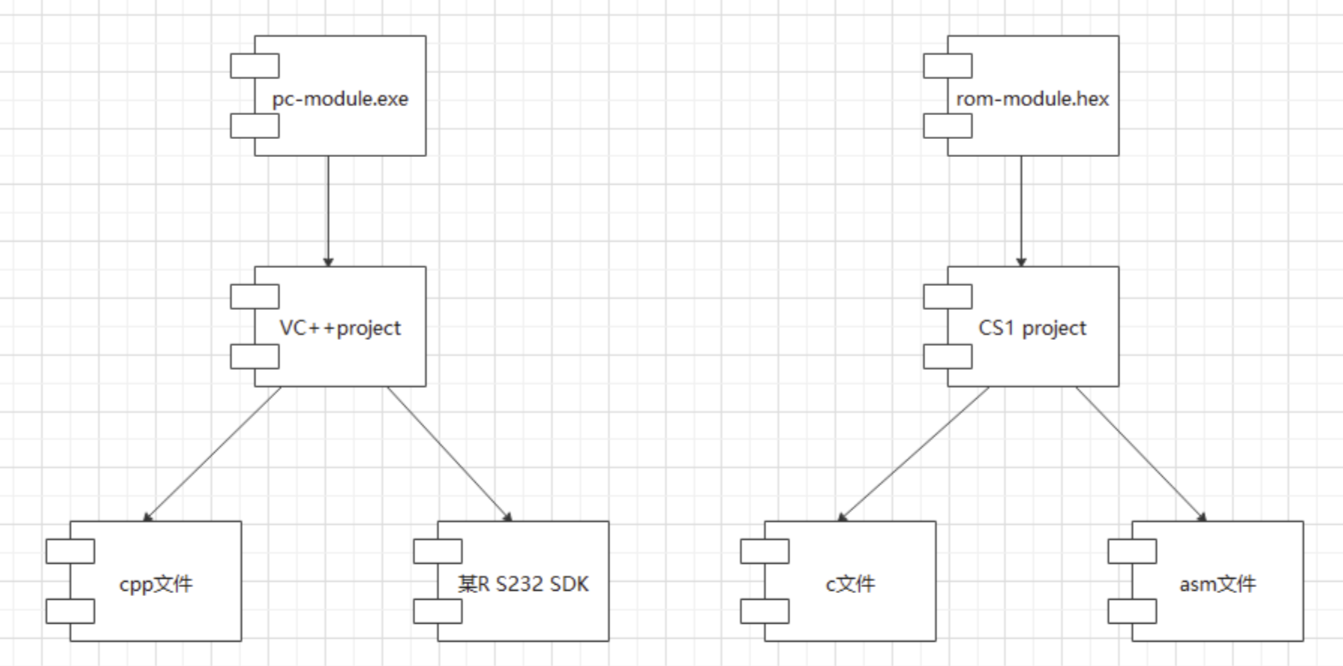
*对逻辑架构元素，描述其代码位置，可以是代码仓位置，或代码目录，或是开源软件的版本信息等；系统的构建，即如何将代码编译成二进制交付件（比如.so/.bin）。这个构建信息需要包括构建依赖、构建工具链、构建环境信息等。*

*良好的开发视图，应满足如下要求：*

*通过逻辑架构元素，能够找到所有的代码和所有的二进制交付件*

*每个代码源文件，都能找到它所属的逻辑逻辑架构元素*

*每个二进制交付件，都能够找到它集成了哪些逻辑架构元素*



# 运行架构视图

## 运行视图原则

关注点：此架构设计视图的关注点是控制流组织，描述系统软件组件之间的通信时序，数据的输入输出，在UML中通常由时序图和流程图表示。

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

## 物理视图原则

*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。*

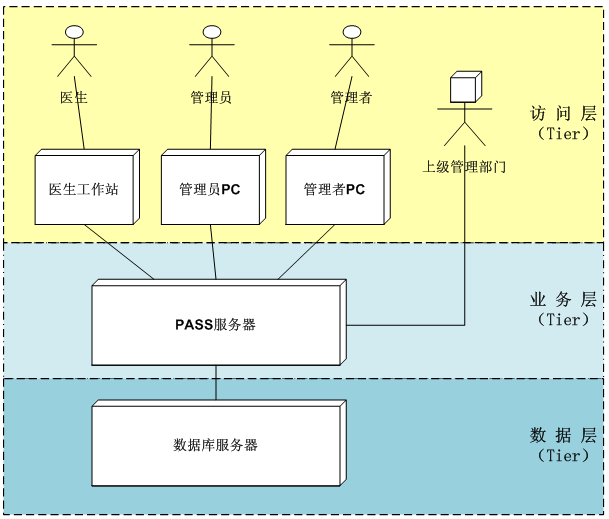
*软件系统，最终是要运行在物理或软件环境上。物理环境可能是服务器、[PC机](https://www.zhihu.com/search?q=PC%E6%9C%BA&search_source=Entity&hybrid_search_source=Entity&hybrid_search_extra=%7B%22sourceType%22%3A%22answer%22%2C%22sourceId%22%3A1544226906%7D" \t "_blank)、移动终端等物理设备；软件环境可以是[虚拟机](https://www.zhihu.com/search?q=%E8%99%9A%E6%8B%9F%E6%9C%BA&search_source=Entity&hybrid_search_source=Entity&hybrid_search_extra=%7B%22sourceType%22%3A%22answer%22%2C%22sourceId%22%3A1544226906%7D" \t "_blank)、容器、进程或线程。[部署视图](https://www.zhihu.com/search?q=%E9%83%A8%E7%BD%B2%E8%A7%86%E5%9B%BE&search_source=Entity&hybrid_search_source=Entity&hybrid_search_extra=%7B%22sourceType%22%3A%22answer%22%2C%22sourceId%22%3A1544226906%7D" \t "_blank)就是对这个部署信息进行描述。*

## 物理拓扑

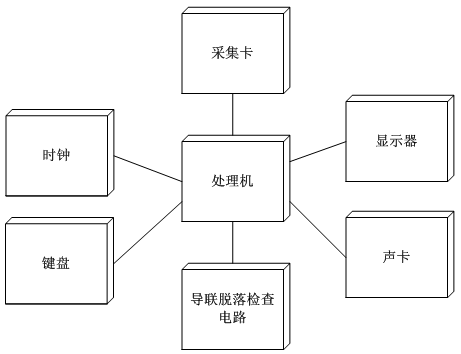
内容：一为硬件选型，二为硬件之间的拓扑连接关系。

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。

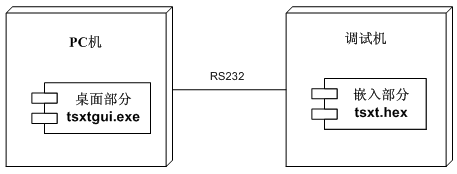


## 软件到硬件的映射

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。

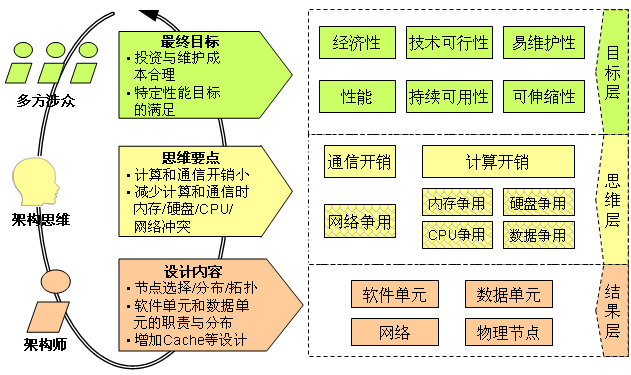
参考：下图所示为设备调试系统中，软件到硬件的映射关系。



## 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。



# 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。

# 其他说明

# 附录