**恒生电子股份有限公司**

**HUI Vue3 组件库**

**架构设计说明书**

<技术平台总部>

2024年 3月

**目录**

[1. 引言 4](#_Toc162370718)

[1.1 编写目的 4](#_Toc162370719)

[1.2 背景 4](#_Toc162370720)

[1.3 术语与缩略语 4](#_Toc162370721)

[1.4 引用文件 5](#_Toc162370722)

[2. 架构目标与设计约束 6](#_Toc162370723)

[2.1 功能性需求 6](#_Toc162370724)

[2.1.1 组织结构 6](#_Toc162370725)

[2.1.2 业务功能 6](#_Toc162370726)

[2.1.3 业务流程 7](#_Toc162370727)

[2.1.4 业务数据 9](#_Toc162370728)

[2.2 非功能性需求 9](#_Toc162370729)

[2.2.1 性能需求 9](#_Toc162370730)

[2.2.2 安全性需求 10](#_Toc162370731)

[2.2.3 可维护性需求 10](#_Toc162370732)

[2.2.4 可测性需求 10](#_Toc162370733)

[2.2.5 其他非功能性需求 11](#_Toc162370734)

[2.3 系统周边与交互 11](#_Toc162370735)

[2.4 开发与运行环境限制 12](#_Toc162370736)

[2.4.1 公司技术选型约定 12](#_Toc162370737)

[2.4.2 运行环境约定 12](#_Toc162370738)

[2.4.3 开发环境约定 13](#_Toc162370739)

[3. 架构设计 14](#_Toc162370740)

[3.1 逻辑视图 14](#_Toc162370741)

[3.1.1 子系统划分 14](#_Toc162370742)

[3.2 开发视图 16](#_Toc162370743)

[3.2.1 功能需求映射与组件识别 16](#_Toc162370744)

[3.2.2 功能需求矩阵 16](#_Toc162370745)

[3.2.3 各组件说明 17](#_Toc162370746)

[3.2.4 技术选型 17](#_Toc162370747)

[3.2.5 各组件对应的代码工程与开发技术 18](#_Toc162370748)

[3.3 运行视图 18](#_Toc162370749)

[3.3.1 Table组件运行态架构设计 19](#_Toc162370750)

[3.3.2 全局定制与主题动态运行的架构设计 19](#_Toc162370751)

[3.3.3 组件支持键盘操作的架构设计 19](#_Toc162370752)

[3.4 数据视图 20](#_Toc162370753)

[3.4.1 Table 组件数据视图 20](#_Toc162370754)

[3.4.2 数据存储与分布 21](#_Toc162370755)

[3.4.3 Table 组件数据管理策略 23](#_Toc162370756)

[4.关键功能模块设计 27](#_Toc162370757)

[4.1 全局定制与主题功能模块设计 27](#_Toc162370758)

[4.2 全键盘方案功能模块设计 28](#_Toc162370759)

[4.2.1 功能性需求 28](#_Toc162370760)

[4.2.2 非功能性需求 28](#_Toc162370761)

[4.2.3 子系统的组件划分、组件协作图 28](#_Toc162370762)

[4.2.4 本CBB与各层组件的协作图 28](#_Toc162370763)

[4.3 Table 组件方案设计 28](#_Toc162370764)

[4.3.1 功能性需求 28](#_Toc162370765)

[4.3.2 非功能性需求 28](#_Toc162370766)

[4.3.3 子系统的组件划分、组件协作图 28](#_Toc162370767)

[4.3.4 重点功能的实现协作图 28](#_Toc162370768)

[5. 非功能性架构设计 30](#_Toc162370769)

[5.1 性能设计 30](#_Toc162370770)

[5.2 可靠性设计 32](#_Toc162370771)

[5.2.1 异常处置设计 32](#_Toc162370772)

[5.3 安全设计 33](#_Toc162370773)

[5.4 可维护性设计 33](#_Toc162370774)

[5.4.1 安装升级 33](#_Toc162370775)

[5.4.2 排查辅助 33](#_Toc162370776)

[5.4.3 生产环境测试设计 34](#_Toc162370777)

[5.5 可测试性设计 34](#_Toc162370778)

[5.5.1 测试数据设计 34](#_Toc162370779)

[5.5.2 单元测试和模块测试的可测性设计 34](#_Toc162370780)

[5.5.3 系统验证测试的可测性设计 35](#_Toc162370781)

[5.6 跨平台性设计 35](#_Toc162370782)

[5.7 其他非功能性设计 35](#_Toc162370783)

[6. 模板修订记录 36](#_Toc162370784)

# 引言

## 编写目的

本文档编写的目的是指导软件开发人员在开发组件的过程中按照设计方案进行开发，本说明书的预期 读者为软件开发人员，软件测试人员和项目评审人员。

## 背景

公司的 HUI vue2 版本的组件库已经支撑公司业务 6 年左右的时间了，产品的生命周期已经逐步走到尽头

vue3 版本在 2021 年进行了发版，新版本的周边生态已经成熟。此外 vue3 带来的模块化和 composition api 组合式的组件编程方式。此外 vue3就虚拟 DOM 树结构和差异对比算法进行了优化，减少了静态内容的重复创建，引入了块级的更新策略。渲染性能平均提升了 1.3 倍左右（官网数据）。根据公司现有业务的需求，组件库适配 vue3 版本已水到渠成。

适配 vue3 版本的组件目的是满足业务方对 vue3 组件需求以及提升 组件库的性能。

## 术语与缩略语

|  |  |
| --- | --- |
| **词汇** | **解释** |
| *Vue3* | 一个 mvvm 的 javascript ui 框架，为第三个版本 |
| 全局定制 | 根据配置设置组件的字体大小，以及 padding 和line-height |
| 键盘快捷键支持 | 组件支持业务方配置快捷键 |
| css 变量 | 一种在 CSS 中定义和复用样式值的机制，使得样式的维护和修改更加灵活、高效。 |
| tree sharking | 一种通过静态分析代码来消除无用代码的优化技术，以减小最终打包的 JavaScript 文件大小。 |
| 虚拟滚动 | 一种优化技术，通过只渲染可视区域内的元素并动态更新，来提高大数据列表的渲染性能。 |
| xss攻击 | 一种通过在网页中注入恶意脚本来窃取用户信息、劫持用户操作或破坏网站的安全漏洞。 |
| JSX | 一种在 JavaScript 中编写类似 HTML 的语法扩展，用于描述组件的结构和外观，提高了代码的可读性和开发效率 |
| 自定义 render | 一种允许开发者自定义组件渲染逻辑的技术 |
| css in js | 种将 CSS 样式直接写入 JavaScript 代码中的技术，通过 JavaScript 的能力来管理和生成样式，提供了更好的样式封装、动态性和可维护性。 |
| composition api | vue3 允许开发者使用函数式的方式组织和重用逻辑，从而更灵活地构建组件。 |

## 引用文件

*应列出本文档引用的所有文档的编号、标题、修订版本和日期，方便读者查阅参考。*

*外部参考：相关行业标准/规范*

*内部参考：公司内部标准/规范，如《产品性能指标说明》、《安全需求与实现指南》，等*

|  |  |  |
| --- | --- | --- |
| **文档名称** | **版本号** | **文档日期** |
|  |  |  |
|  |  |  |

# 架构目标与设计约束

*影响架构设计大方向（涵盖功能性需求、非功能性需求、系统周边交互、开发与运行环境限制）的架构目标与设计约束。*

*架构目标与设计约束对后续架构设计的具体展开具有重大指导和方向性影响。*

## 功能性需求

### 组织结构

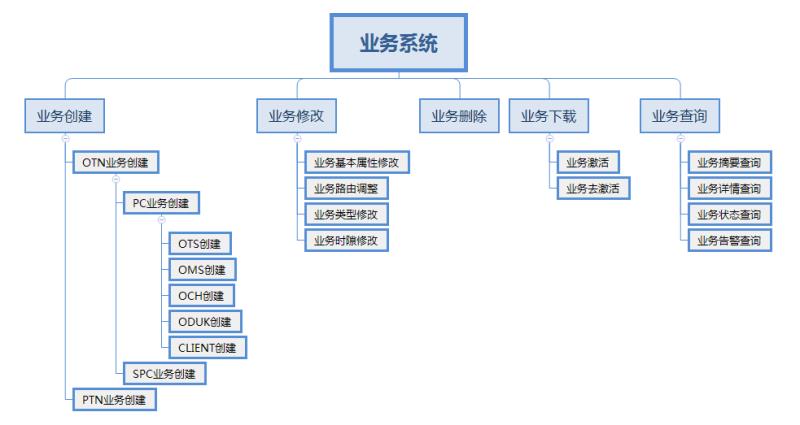
*阐述系统、产品或项目相关的组织结构、合作伙伴、业务渠道等，可以组织结构树、用户角色列表等方式描述系统参与人（部门、角色）信息。如果是面向C端产品，可以增加干系人的说明。*

### 业务功能

*阐述系统、产品或项目相关的顶层价值链分析、业务功能层次分解及主要功能的定义，可以功能树、功能框图、功能表格、功能分类列表等方式描述功能范围*

*【注意项1】:全面涵盖待研发软件的功能范围，无需展开对功能介绍的详细描述。*

#### 业务功能树

**

#### 关键功能识别与问题单分析

*【背景1】关键需求决定架构。当设计需要权衡时，照顾的是关键需求*

*【背景2】企业的问题单数据库，记录了同类系统暴露过的重大问题*

*本节的重点是需要分析历史问题、提高软件设计的针对性、避免问题重演*

*A）列出关键功能*

*B）列出对问题单的单号、描述、风险点分析，设计时提前考虑风险和应对方案*

*【示例1：关键功能问题分析】*

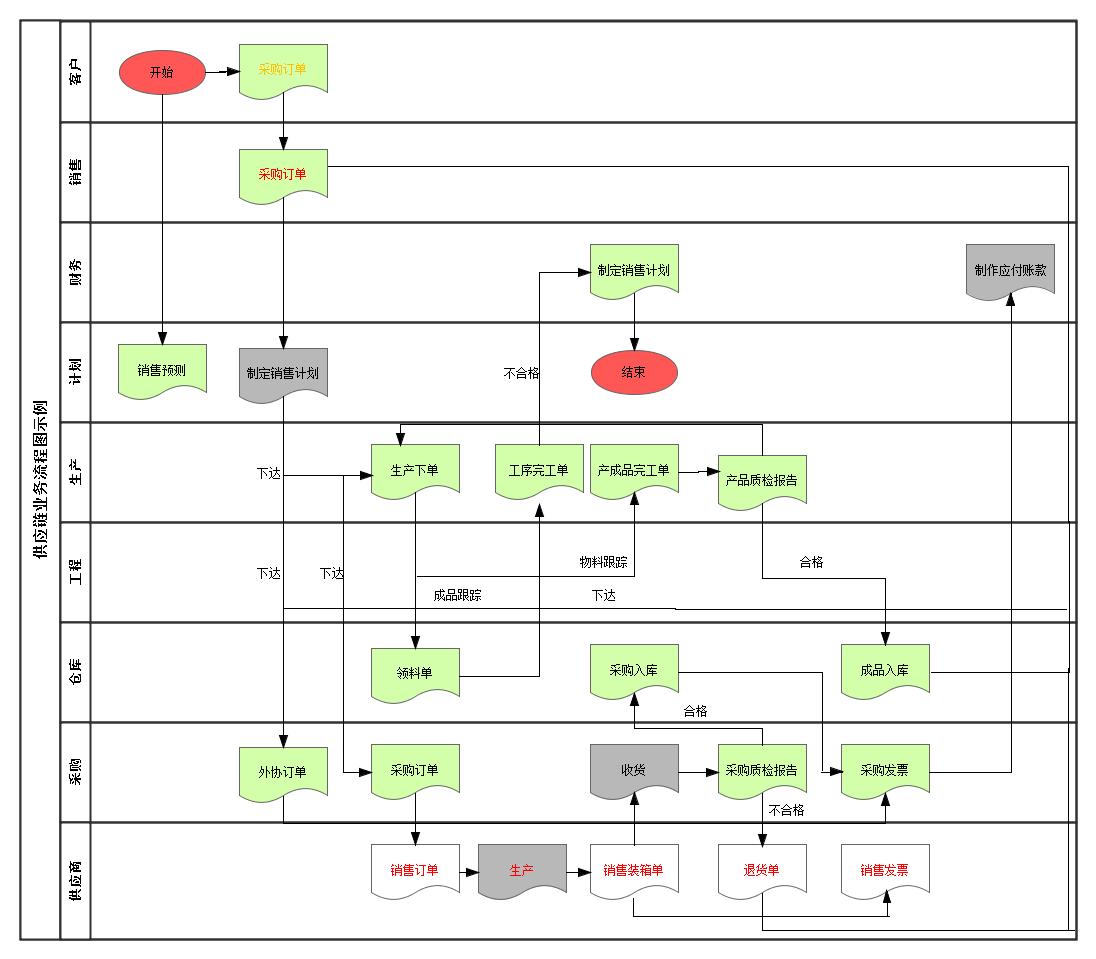
|  |  |  |  |
| --- | --- | --- | --- |
| **关键功能** | **问题单分析** | | |
| **单号** | **历史问题** | **设计风险** |
|  |  |  |  |
|  |  |  |
|  |  |  |  |
|  |  |  |

### 业务流程

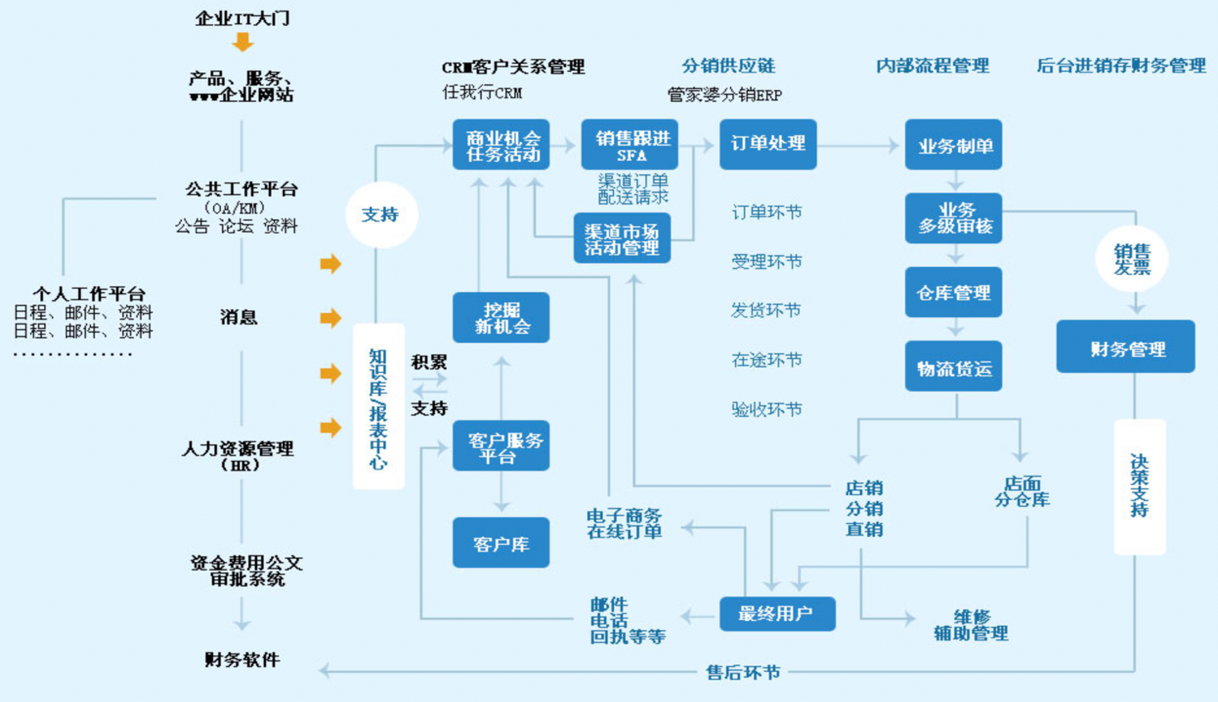
*阐述系统、产品或项目相关的主要关键流程及主要分支流程，流程中识别出业务组件、业务对象、业务规则等*

*关键流程的定义：与架构设计强相关的内容*

*【示例 1：业务流程图】*

**

*【示例2：业务流程图】*



### 业务数据

*概要阐述系统、产品或项目相关业务对象的主要数据对象，可以参考下述内容：*

*业务流程中产生哪些数据以及数据的变化。 如：DDD中的事件风暴和实体；*

*说明数据的时效性 以及 数据的生命周期；*

*如果参考国家/行业标准也可以备注说明；*

## 非功能性需求

### 性能需求

*阐述性能要求。如：响应速度、业务处理量、数据容量等。*

*根据产品性能分级，明确产品性能和容量的分级属性，填写《产品性能指标说明》*

*参考《产品性能指标说明》 “立项阶段填写页”中的指标项内容进行填写；*

*产品的性能定位里如果没有分类项，可以不选，说明没有的理由即可；*

*产品的性能定位如果不包含在分级档位内，可以不选，并自行添加；*

*指标分类如不满足要求，可自行添加*

*【示例：性能指标及要求】*

**

### 安全性需求

*参考公司安全裁剪表（可以裁剪）*

*如果客户内部有交付安全要求，增加客户安全性说明*

### 可维护性需求

*系统可维护性需求主要涵盖：*

* *部署升级：系统应提供完备的操作手册，安装手册，操作过程中，各种操作提示应该准确、清晰。上线初期应配合做好系统使用的培训工作；*
* *辅助排查：收集各种日志信息，并有工具可视化排错；*
* *运行监控：资源层的监控、应用业务层监控以及设置各监控指标的告警阈值；*
* *其他日常运维：包含生产性测试需求，一键备份、一键恢复等。*

### 可测性需求

*可测试性是指软件发现故障并隔离、定位其故障的能力特性，以及在一定的时间和成本前提下，进行测试设计、测试执行的能力。*

*【示例：可测试需求 参考内容项】*

|  |  |  |
| --- | --- | --- |
| 序号 | 需求分类 | 具体参考内容 |
| *1* | *可观察性* | *所看见的，就是所测试的；*   * *更容易观察到测试结果：业务记录是否完整、业务异常是否有明确输出* * *版本状态可查* * *业务路径可查：业务执行状态和过程可观察、调用链可查* |
| *2* | *可控制性* | *对软件的控制越好，测试越能够被自动执行与优化；*   * *适于自动化测试：测试数据准备、界面稳定性、组件标准化、验证码开关控制、前后台调用记录可查* * *适于功能测试：支持测试环境跳日期场景、配置参数实时生效、KEY开关控制、报文模拟器* |
| *3* | *可分解性* | *通过控制测试范围，能够更好地分解问题，执行更灵巧的再测试：投产风险可控；配置分级原则* |
| *4* | *简单性和稳定性* | *每次增加或修改的功能特性，应采用最小的改动；改变越少，对测试的破坏越小；需要测试的内容越少，测试的速度越快；*  *软件的变化不影响已有的测试，换个表达方式：软件设计上向前兼容，不影响原有功能，原有的用例应都能运行通过；* |
| *5* | *易理解性* | *得到的信息越多，进行的测试越灵巧；*  *信息的内容来源于:系统功能清单、系统操作说明、项目文档、接口文档等，需要能够提供充分的信息内容* |

### 其他非功能性需求

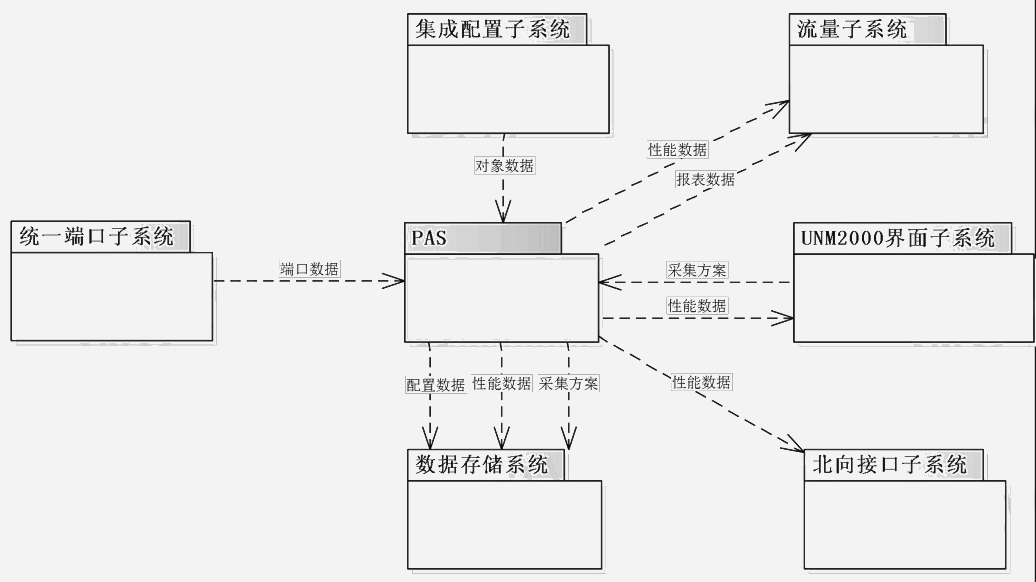
*描述其他非功能性需求，比如新老系统切换时的数据迁移需求、二次开发需求等*

*如兼容性需求：标准协议的兼容性、API兼容性、程序运行环境的兼容性、数据库可移植的兼容性*

## 系统周边与交互

*系统周边与交互描述的是：以本系统的视角，与外部周边系统的交互。强调的是以顶层的视角看到系统与外部的关联关系*

*【示例 ：系统与周边交互图】*

**

## 开发与运行环境限制

### 运行环境约定

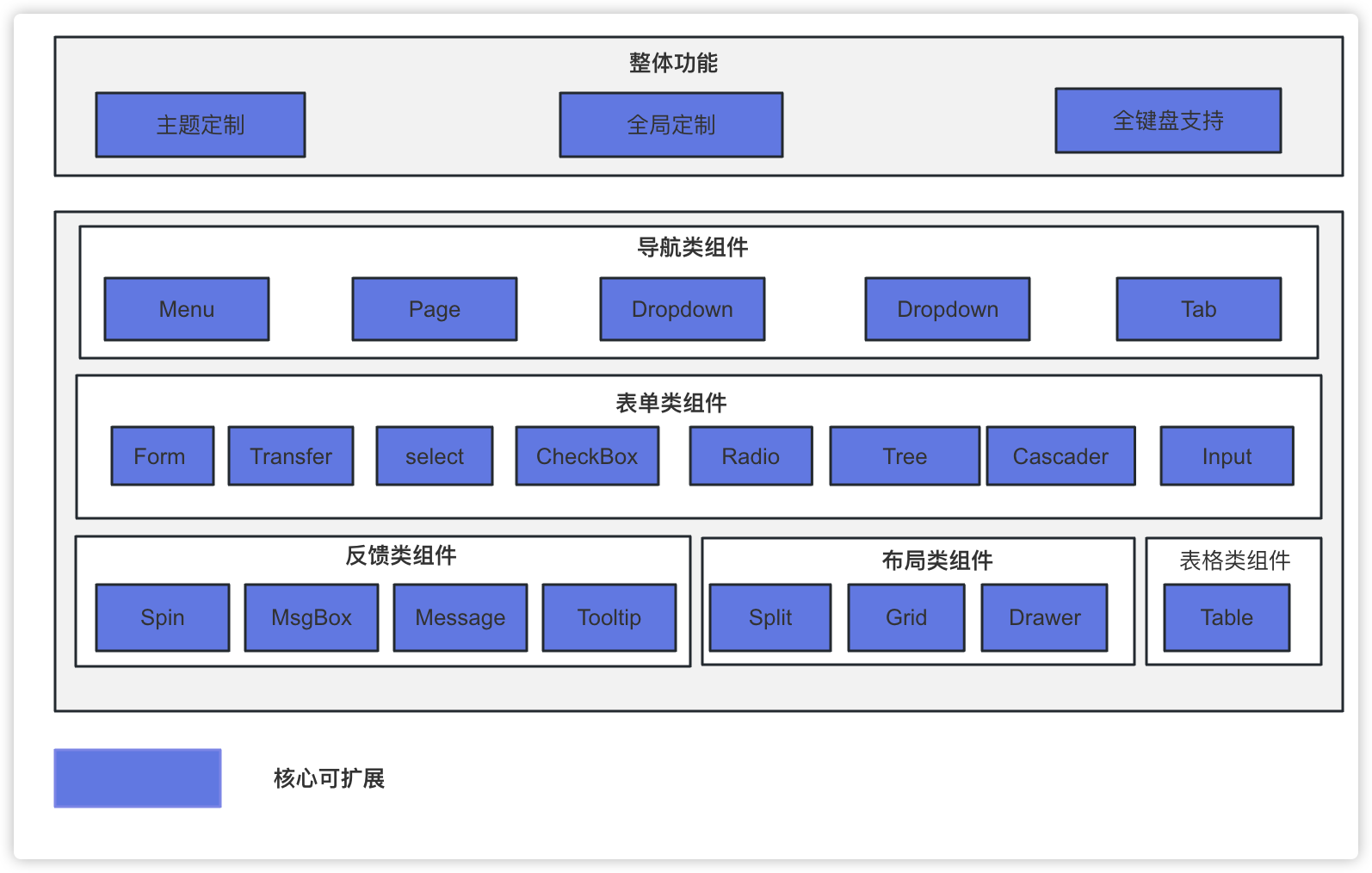
|  |  |  |
| --- | --- | --- |
| **运行环境** | **约束类型** | **说明** |
| WebView/浏览器 | CEF /Chrome 95+ | 95 代表内核版本为95 |
| firefox 103+ |  |
| QtWebEngine 6.3.0 | *内核为chromium95* |
| 360信创浏览器 | *内核为chromium95* |
| webview2 1.0.945.30 + | *内核为Chromium 95.0.4638.69* |

### 开发环境约定

|  |  |  |
| --- | --- | --- |
| **工具分类** | **工具名称** | **版本** |
| 构建工具 | roullup |  |
| 开发语言 | javascript | ES2015+ |
| 版本管理工具 | git |  |
| 单测工具 | vitest | v1.4.0+ |
| 组件 UI 测试工具 | cypress | v13+ |
| 其他 |  |  |

# 架构设计

## 逻辑视图



*逻辑架构的设计着重考虑功能模块，系统应当向用户提供什么样的服务，关注点主要元素：系统、功能模块、子系统，子系统功能模块，以及子系统间交互；*

*逻辑架构的静态方面是抽象职责的划分（包含横向的功能划分和纵向的技术分层），动态方面是承担不同职责的逻辑单元之间的交互与协作。*

### 子系统划分

*本节以功能树、功能框图、功能表格、功能分类列表等方式描述功能子系统划分*

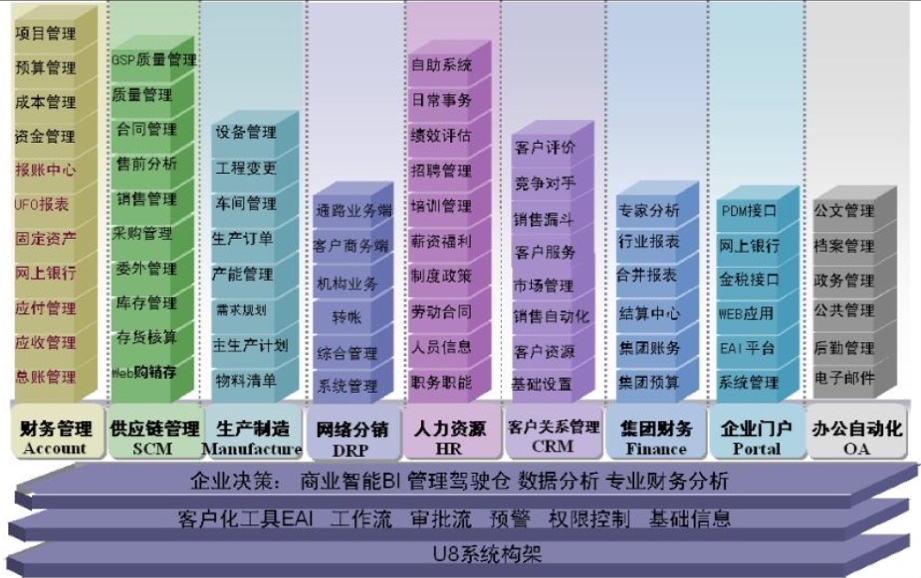
*列出或画出包含的功能子系统，并说明其功能边界。*

*每个子系统应该是可以交付的实体，子系统包含API、后端驱动、前端插件尽早确定。*

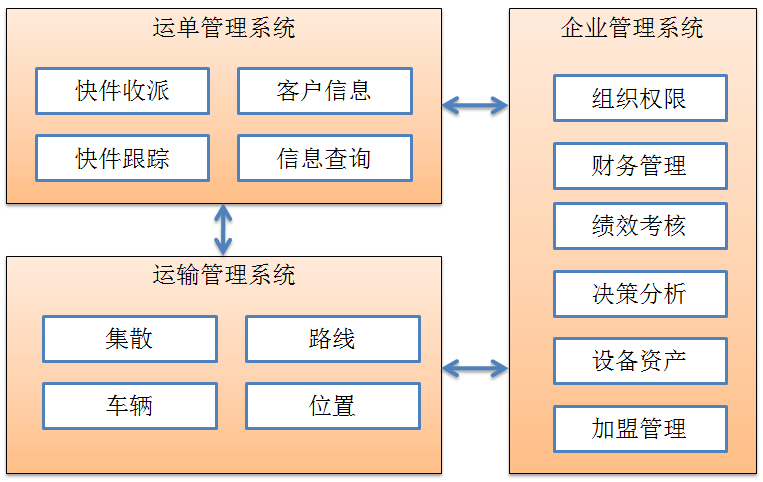
*子系统划分的原则：*

* *业务领域强相关原则（可参考DDD领域划分）*
* *高内聚/低耦合原则（业务功能尽量保持独立）*
* *按质量属性拆分原则（性能、可用性、快速迭代、稳定性、安全性）*
* *康威定律原则（组织结构、团队人数、组织形态、内部资源独立）*
* *数据冗余最小原则（数据尽量保持独立、一致性）*
* *部署硬件适当原则（适应业务的硬件投入需求）*
* *交付隔离适当原则（系统独立交付和聚合交付）*
* *外部系统独立原则（非本部门/非本公司系统，尽量独立部署）*

*【示例】子系统划分形式一：功能树*

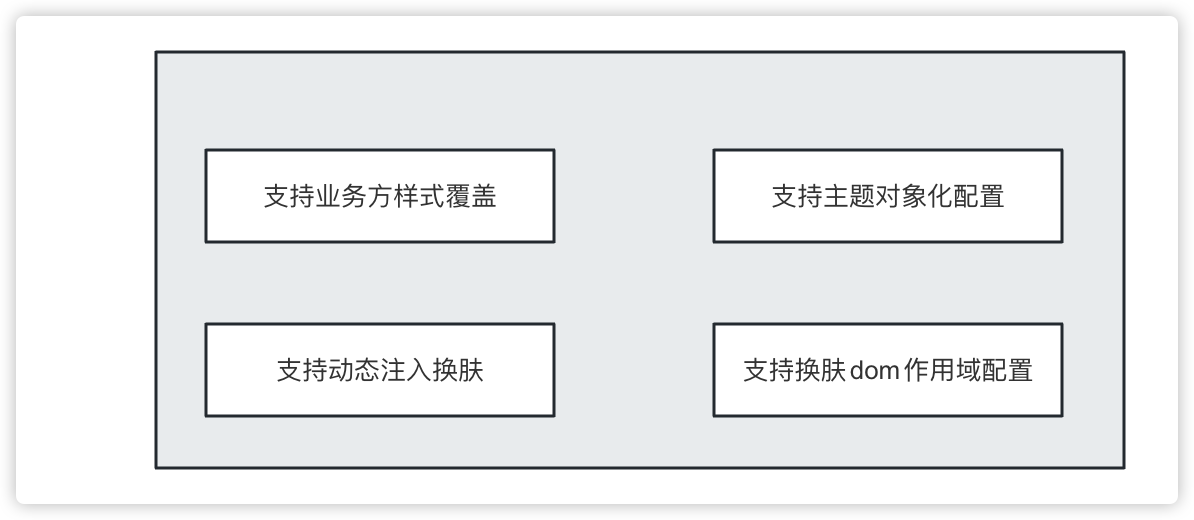


*【示例】子系统划分形式二：功能框图*



#### 主题换肤功能边界说明

主题换肤功能边界说明：



1. 主题换肤功能个边界支持业务采用样式覆盖进行
2. 主题换肤支持对象化配置。
3. 主题换肤支持动态注入：可以通过修改主题对象来添加，修改或删除主题。
4. 主题换肤支持主题的作用域：默认主题相关的class 目标节点为 html节点，支持自定义dom 节点。
5. 主题换肤支持修改主题名就可以进行换肤。
6. 主题换肤支持如下类别

|  |  |  |
| --- | --- | --- |
| 类别 | 一级子类 | 二级子类 |
| 主题色 | 全键盘 |  |
| 点击 |  |
| 常规 |  |
| 悬浮 |  |
| 聚集外边框 |  |
| 特殊场景 |  |
| 浅色背景 |  |
| 功能色 | 成功色 | 全键盘 |
| 点击 |
| 常规 |
| 悬浮 |
| 聚集外边框 |
| 特殊场景 |
| 浅色背景 |
| 警告/提醒 | 全键盘 |
| 点击 |
| 常规 |
| 悬浮 |
| 聚集外边框 |
| 特殊场景 |
| 浅色背景 |
| 错误 | 全键盘 |
| 点击 |
| 常规 |
| 悬浮 |
| 聚集外边框 |
| 特殊场景 |
| 浅色背景 |
| 链接/信息 | 全键盘 |
| 点击 |
| 常规 |
| 悬浮 |
| 聚集外边框 |
| 特殊场景 |
| 浅色背景 |
| 中性色 | 文本色 | 标题/正文 |
| 描述 |
| 提示/辅助 |
| 禁用 |
| 白色 |
| 背景色 | 特殊场景1 |
| 特殊场景2 |
| 特殊场景3 |
| 禁用 |
| 斑马纹 |
| 特殊场景4 |
| 面板背景 | 背景层 |
| 主内容区 |
| 边框色 | 常规边框 1 |
| 常规边框 2 |
| 分割线 | 页面 |
| 阴影/遮照色 | shadow-1 |
| shadow-2 |
| shadow-5 |
| shadow-3 (弹窗/抽屉) |
| shadow-4(气泡卡片/下拉面板) |
| 图标色 | icon-1 |
| icon-2 |
| icon-3 |
| icon-4 |
| icon-5 |

#### 全局定制功能边界说明

#### Table 表格组件的功能边界

*阐述具体某个子系统的功能边界说明*

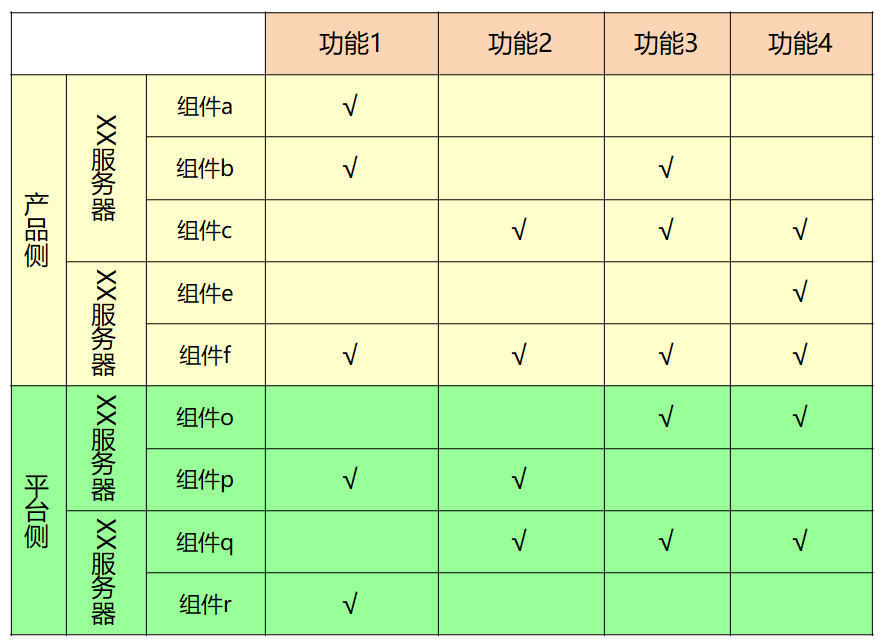
#### 组件支持键盘操作功能边界

## 开发视图

*开发视图描述开发环境的静态组织结构，设计着重考虑开发期间的质量属性，关注点在软件开发环境中模块的实际组织方式，具体涉及源程序文件、配置文件、源程序包、编译打包后的目标文件、直接使用的第三方SDK/框架/类库、以及开发的系统将运行于其上的系统软件或中间件。*

### 功能需求映射与组件识别

*【示例】功能与参与功能实现的组件的映射矩阵*

**

*【备注说明：】组件的定义：技术类组件，进程级别； 一个子系统可以由多个组件组成。*

### 功能需求矩阵

*描述功能需求如何与效能平台对应，便于需求的实现在效能平台上进行跟踪*

*说明：如果需求已经在效能平台上管理，添加效对应链接即可，该部分无需再填写；*

*如果需求在效能平台没有管控，则按照以下示例表格进行填写说明；*

*【示例 -- 功能需求跟踪矩阵】*

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **功能名称** | **效能平台**  **需求编号** | **需求标题** | **所属子系统名称** | **开发工作量（总）** | **需求分类** |
| 功能1 | S2107210545 | 开户功能实现 | 账户中心 | 8 | 新增功能 |
| 功能2 | S2111250239 | 风险测评 | 账户中心 | 4 | 功能优化 |
| 功能3 | S2109130089 | 对账功能实现 | 清算中心 | 12 | 功能优化 |
| 功能4 | S2112030054 | 换卡业务 | 账户中心 | 3 | 缺陷 |
| 功能5 | S2110210694 | 待办事项 | 销售柜台 | 6 | 功能优化 |

### 各组件说明

*要求1：画UML组件图*

*要求2：采用多种方式说明：组件能力、组件边界（列表格、文字说明）*

### 技术选型

*结合组件的功能 和 系统分层架构（3.1.3）， 确定系统的技术选型；*

*【示例 ：组件技术选型】*



*说明选型依据和原因：  
前端展示层：采用BSCS混合模式，C#客户端，嵌入BS网页。  
 对于高频低延时全键盘操作，采用CS模式UCF框架，提升用户体验满足盯盘抢单的需求  
 对日常运维、系统设置、报表查询的等非高频低延时操作的界面，采用BS模式HUI框架，敏捷开发快速迭代，及时响应需求。*

### 各组件对应的代码工程与开发技术

*包含的多个开发工程（Project）或Makefile。*

*要求1，说明每个组件的工程目录结构以及命名规范，及组件内的代码组织形式；*

*要求2：说明每个组件开发所需的三方库、框架等的技术选型；*

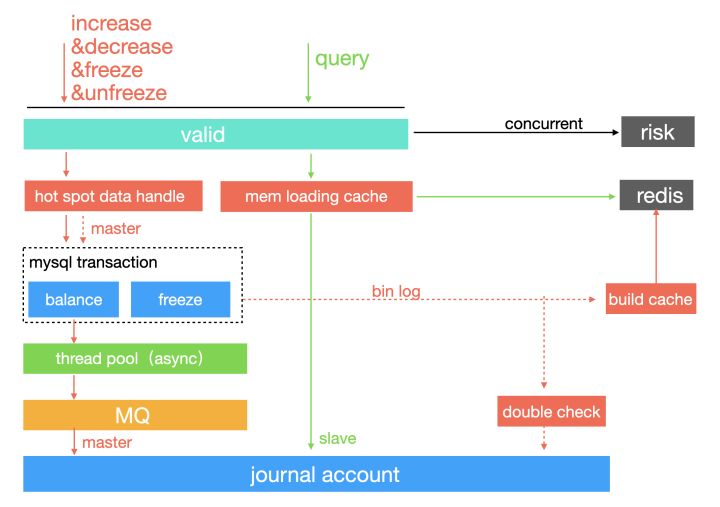
## 运行视图

*通过逻辑视图、开发视图和部署视图，描述的都是系统的静态信息，有助于帮助我们清晰的知道系统中每一个逻辑架构元素、每一份代码，最终会运行在什么位置上；*

*运行架构的设计着重考虑运行期质量属性，关注点是系统的并发、同步、通信等问题，这势必涉及到进程、线程、对象等运行时概念，以及相关的并发、同步、通信等；*

*阐述本系统最核心、最复杂的组件运行态架构设计；*

*【示例： 组件间运行视图】*

**

*运行视图的文字说明：*

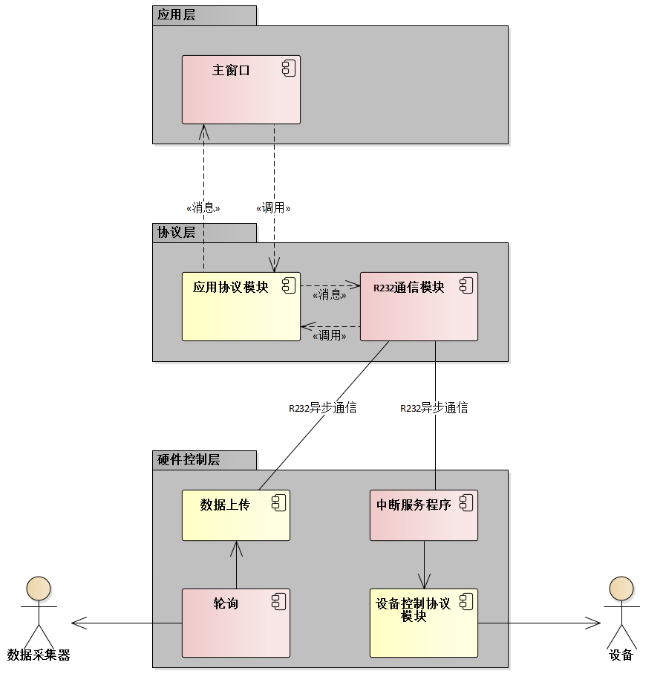
*写请求：经过合法性校验，这部分IO较多，进行并行处理节省耗时；基于数据库事务完成余额表更新，通过线程池完成余额更新后的消息异步发放*

*读请求：同样进行合法性校验，直接命中进程内loading cache，否则会依次进行缓存穿透*

### Table组件运行态架构设计

*阐述运行时的各核心组件协作状态*

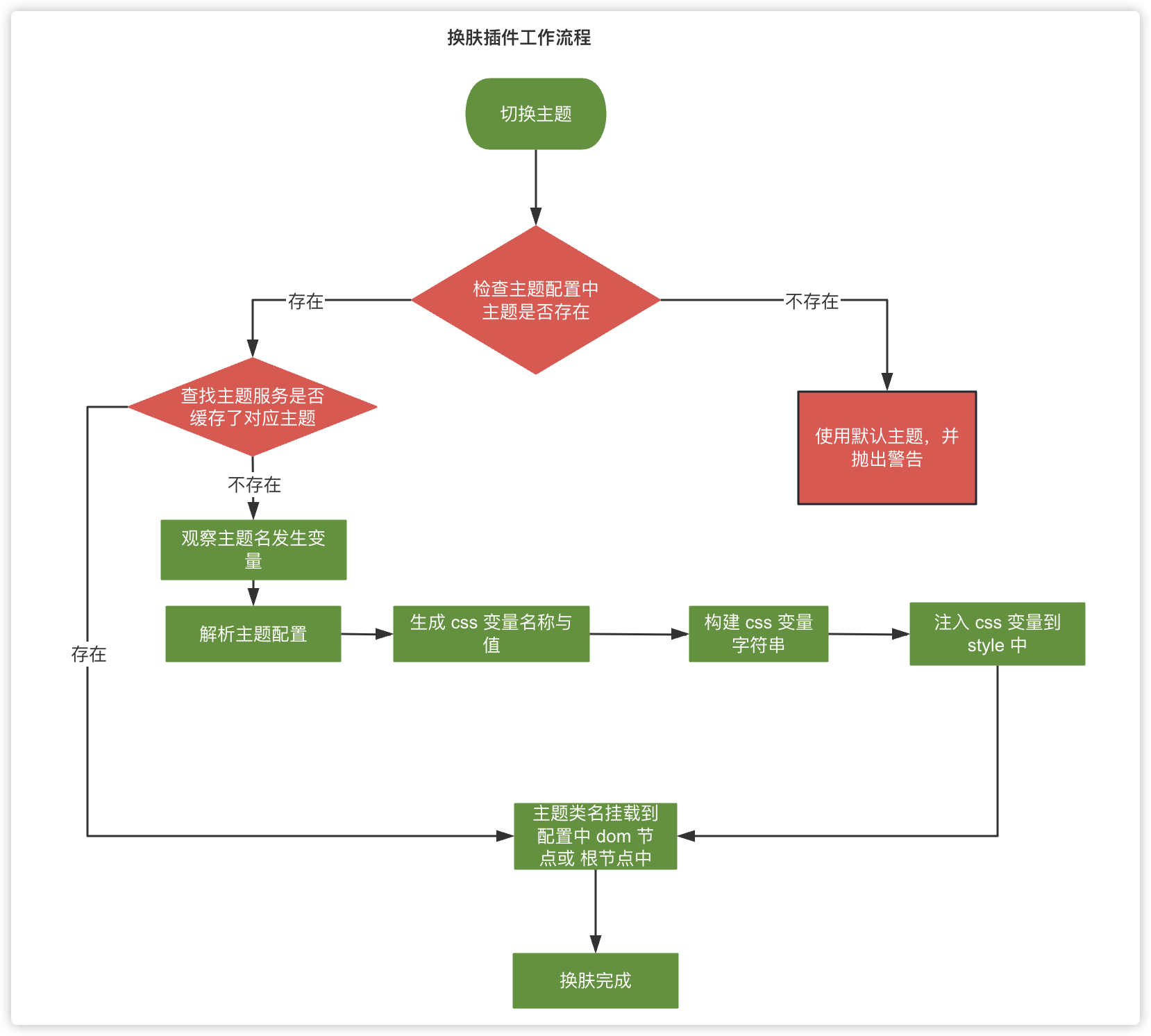
*【示例 – XX系统运行视图】*

**

### 主题切换运行的设计

主题换肤通过 vue 插件的形式进行全局依赖注入，提供 composition api 进行换肤操作。

换肤运行流程如下：



组件换肤支持用户调用 api 的形式进行换肤，大致原理是利用 vue 的 provide inject ，进行全局注入。

用户调用换肤插件提供的 compostion api 进行换肤操作，并支持动态注入皮肤主题。

对已经注入的皮肤进行缓存，减少主题解析流程。

主题配置分为 color,font模块。配置是将 css自定义变量与 js 做映射。

### 全局定制运行的架构设计

### 组件支持键盘操作运行视图

## 数据视图

*数据视图是以“数据”的视角，来描述我们系统相关的架构设虑，在设计时着重关注：数据模型、持久化存储与分布、复制同步、备份等方面；*

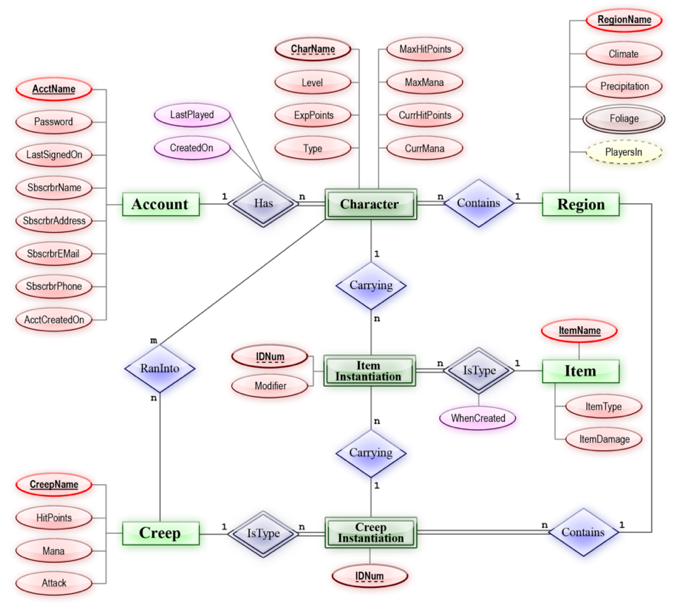
*数据作为有状态的服务，也是我们软件的重点和难点。作为架构师，我们应该意识到数据是企业的生命线，必须引起足够的重视。*

### Table 组件数据视图

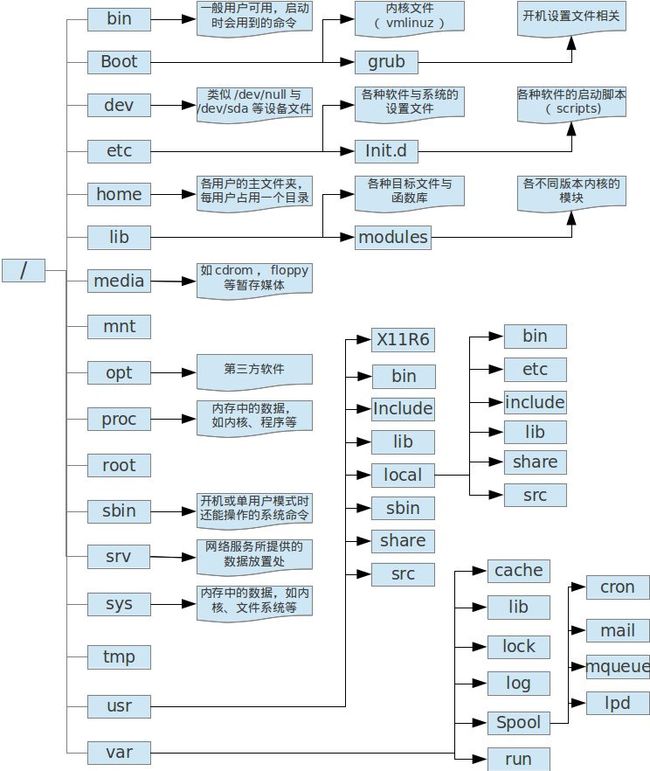
*【说明1】结构化以ER图、树形结构等方式描述核心数据的关联信息*

*【说明2】非结构化的数据模型以树形结构描述更简单清楚*

*【示例1】 ER图表代表顶层的ER图（主数据相关模型）*

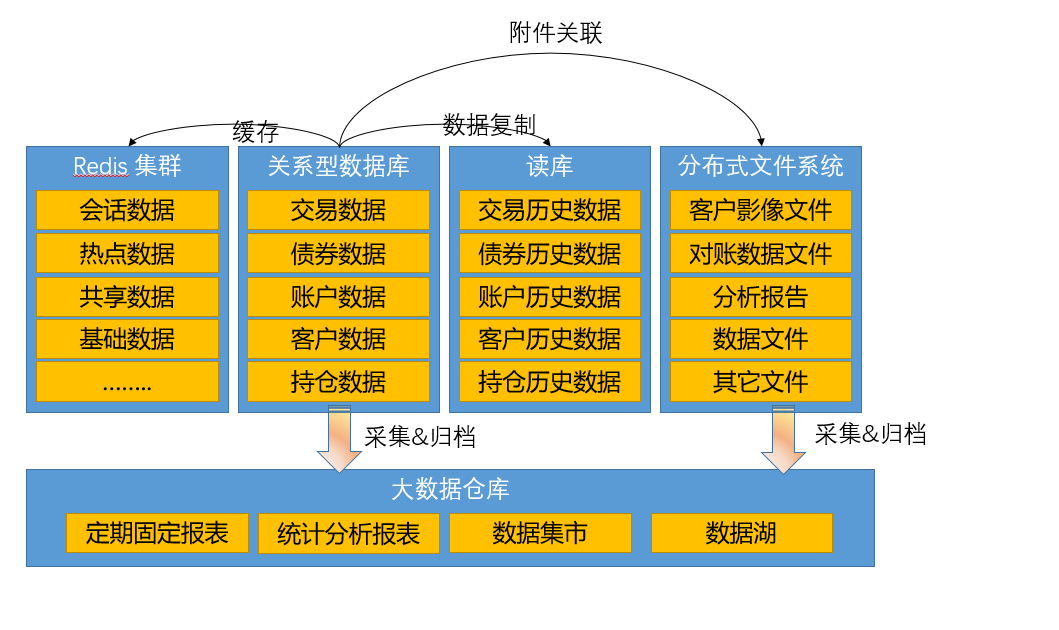


*【示例2】以树形结构描述非结构化存储目录的说明及规则*

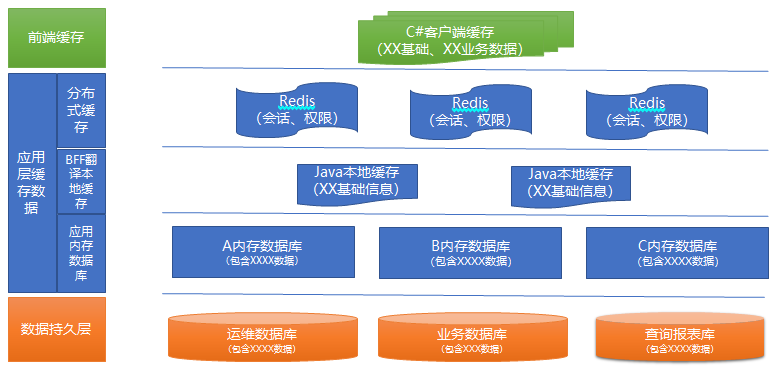


### 数据存储与分布

*【示例1】 数据存储表，说明清楚核心数据模型在存储系统分布情况*

**

*【示例2】分层数据存储分布图*

**

*数据存储分布说明，包含但不限于：*

* *Oracle RAC存储方案*
* *Mysql存储方案*
* *分布式NoSQL缓存方案*
* *分布式文件系统存储方案*
* *大数据仓库存储方案*
* *本地内存存储*

*系统中各核心数据存储说明：*

|  |  |  |
| --- | --- | --- |
| 存储系统名称 | 核心数据类型 | 说明 |
| *Oracle RAC* | *存储联机交易数据* |  |
| *存储结构化数据* |  |
| *存储非结构化数据* |  |
| *MySQL分布式数据库* | *存储结构化历史数据* |  |
| *存储非结构化历史数据* |  |
| *存储非核心交易联机数据* |  |
| *Redis Cluster* | *公共数据缓存* |  |
| *系统参数缓存* |  |
| *热点数据缓存* |  |
| *分布式文件系统* | *存储文件及电子文档* |  |
| *大数据仓库* | *存储历史数据、事实数据、分析数据。* |  |
| *本地内存存储* | *缓存类数据，热点数据* |  |
| *。。。* | *。。。* |  |

### Table 组件数据管理策略

#### 数据存储机制

*不同数据特性会有不同的存储机制，通常考虑到：存储的成本、用户访问的速度以及存储的容量要求，数据的存放策略一般是：*

* *只有当前最普遍被访问的数据保留在在线存储当中（如数据库系统）；*
* *不经常访问的数据应该被保存在近实时的介质中（如大数据平台）；*
* *而处于静止状态的不活动数据存储在离线的介质中（共享存储、磁带、光盘等）。*

*数据存储判断依据是：*

* *访问频率*
* *更新频率*
* *历史的深度*

*数据按照存储不同可以做以下分类，并且针对不同分类的数据，在企业中存储的方式也是不同的，具体有如下考虑：*

*1、业务系统数据：*

* *核心数据：属于热数据，需要在线存储*
* *历史数据：根据业务需要保留必要的周期，超期的历史数据进入历史数据平台的近线存储中，也可以选择近线磁带库这样的离线存储。*

*2、分析系统数据：*

* *数据仓库：数据仓库内所有数据均会在历史数据平台的近线存储上长期存储*
* *缓冲层：在线存储一定周期（不超过1个月）*
* *贴源模型层：历史拉链表在线存储，流水表根据需要保留一定周期*
* *主题模型层：历史拉链表在线存储，流水表根据需要保留一定周期*
* *共性加工层：保留加工处理所需要的一定周期的数据*
* *仓内集市层：保留加工处理所需要的一定周期的数据*
* *应用集市：保留访问所需要的一定周期的数据*
* *大数据平台：保留加工处理所需要的一定周期的数据*
* *历史数据平台：近线存储所有业务系统和分析系统的数据，如果确定不再需要，将其下线*

*3、文件类数据*

* *日志文件：存储至近线存储（大数据平台）上，保留一定周期（例如1年内）的数据，告警和报错类的日志长期保留*
* *数据文件：存储至近线存储（大数据平台）上，同时也需要归档至离线存储*

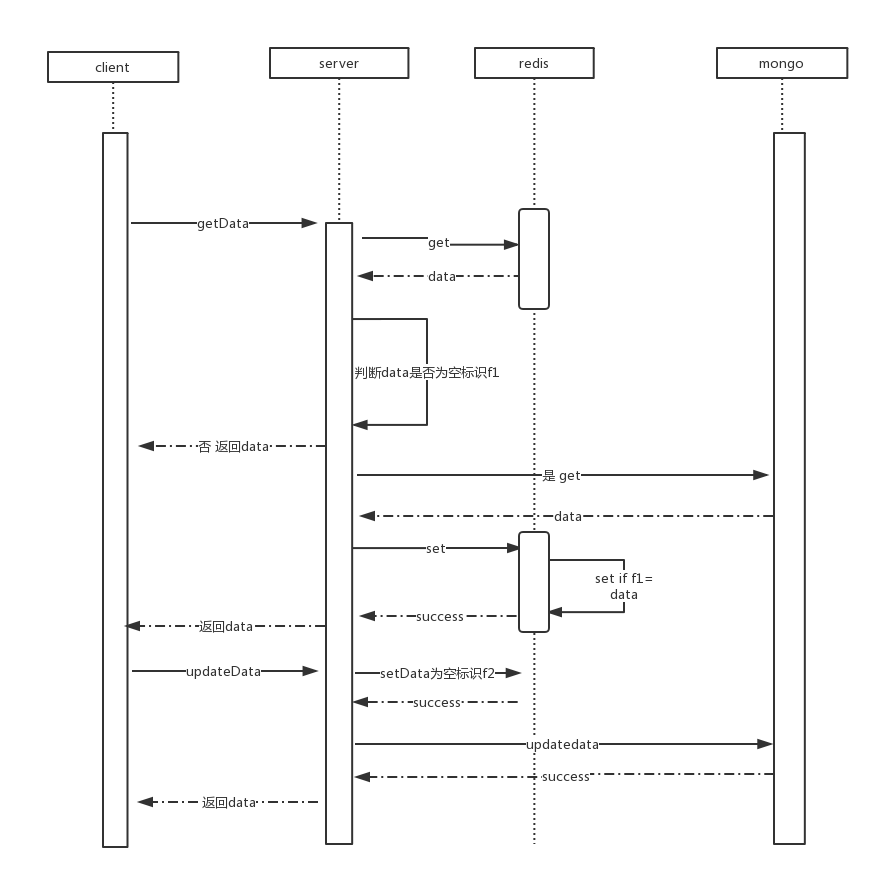
*4、系统数据*

* *数据量较小，本地存储和异地存储即可*

#### 数据同步机制

*使用时序图、流程图描述不同数据存储之间的同步机制。如分布式缓存与数据库同步机制 、内存数据库与关系型数据库的同步。*

*【示例 - 数据同步流程】*



# 

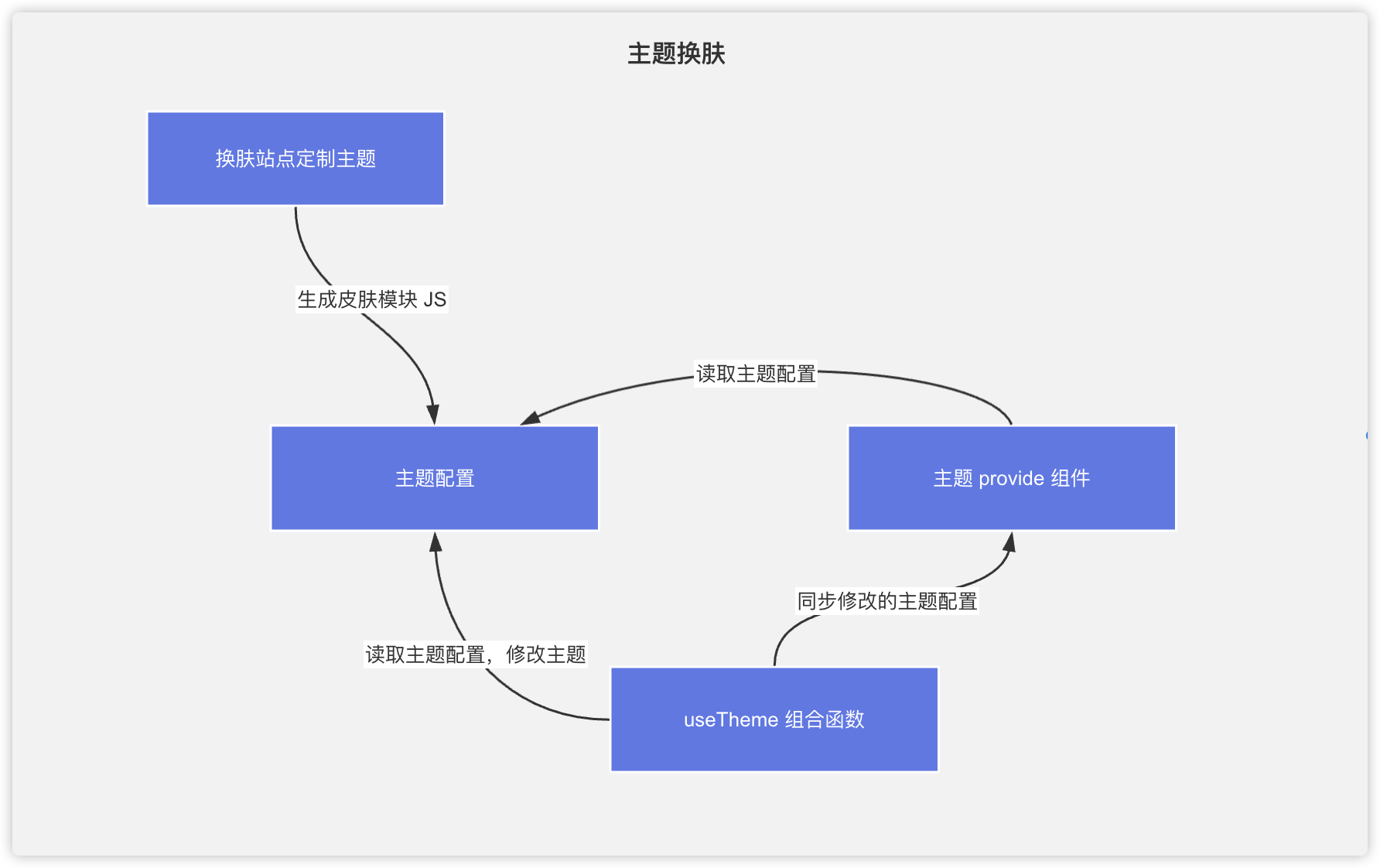
# 4.关键功能模块设计

*架构师是为目标负责，而“各个子系统都是黑盒”式的架构设计必然是无法保证支撑“架构级设计目标”，所以本版本的架构设计说明书，增加本节关键子系统设计。*

*产品架构设计说明书，对关键子系统的设计，无需面面俱到；主要关注是关键子系统的核心组件划分，以及组件之间的协作图；*

## 主题换肤模块设计

主题换肤由三部分组成如下所示：



换肤站点：换肤站点主要是自定义主题预览，生成自定义皮肤模块。

主题配置：主题配置为主题全局配置支持自定义主题。

主题配置的接口定义 大致如下

interface Theme{

defaultTheme:string,

themes: ThemeDefinition

}

interface ThemeDefinition{

color:{

[propName: string]: string;

}

fontSize:string

}

主题 provide 组件：为提供主题上下文的组件，用来设置页面全局的主题样式。并支持主题嵌套。

useTheme组合函数用于动态切换主题，修改主题配置，以及获取当前主题。

主题配置默认预制 light,dark 主题，css 变量的映射参考主题换肤功能边界说明的 css 变量表。

## 全局定制

## 全键盘方案功能模块设计

### 功能性需求

*【说明】参考2.1内容*

### 非功能性需求

*【说明】参考2.2内容*

### 子系统的组件划分、组件协作图

*【说明】子系统内，有图来描述所有组件及其协作，有文字和表格对每个组件进行说明*

### 本CBB与各层组件的协作图

*【说明】协作图产出标准*

*协作图 = 分层架构大图+嵌入传统协作图*

* *层内：运行环境+开发技术+组件*
* *层间：接口技术+接口*
* *组件：程序组件+库+数据库+库内数据*
* *关系：组件间调用+数据流*

## Table 组件方案设计

### 功能性需求

*【说明】参考2.1内容*

### 非功能性需求

支持

### 子系统的组件划分、组件协作图

*【说明】子系统内，有图来描述所有组件及这些组件在各层的分布，有文字和表格对每个组件进行说明。*

### 重点功能的实现协作图

*【说明】协作图产出标准*

*协作图 = 分层架构大图+嵌入传统协作图*

* *层内：运行环境+开发技术+组件*
* *层间：接口技术+接口*
* *组件：程序组件+库+数据库+库内数据*
* *关系：组件间调用+数据流*

# 非功能性架构设计

*架构师在执行产品架构设计时，最为关键、最为核心的就是持续不断的满足非功能性需求。*

*前面的架构目标与设计约束中，是我们架构师需要识别和理清产品的非功能性需求，*

*在本章节，以产品架构目标与设计约束 非功能性需求 为核心目标，进行软件架构设计。主要涵盖：性能性能、可靠性设计（含高可用）、安全设计、可维护性设计、可测试性测试、兼容性设计等方面。*

## 性能设计

*性能设计目标：最终要满足架构目标与设计约束 非功能性需求 的性能需求*

*根据产品立项环节制定的《产品性能分级》，需求环节确定的《产品性能指标》来阐述通过怎么样的设计来达到里面的要求*

* *包含以下方面的设计（1）服务单笔延迟、高并发；（2）前端展示性能；（3）可支撑的系统容量（客户数、账户数、业务笔数、批处理容量）；（3）资源占用/利用率（ 内存、CPU、硬盘、带宽、缓存命中）*
* *考虑数据增长情况进行设计：考虑当日交易业务临近闭市的数据增长；考虑多日运行以后的数据变化以及相应逻辑的效率，例如，多日业务的多日数据增长以及分析查询的历史数据量的增长情况*
* *说明系统性能设计的要点，必要情况下需提供影响性能的协作图、时序图等图示说明*
* *考虑可伸缩能力（支持通过硬件进行垂直扩展、水平扩展）*

*可以参考（但不限于）以下内容，进行详细的展开说明：*

*【示例1：硬件特性性能相关设计】*

*绑核*

*网卡负载模式*

*FPGA*

*【示例2 ：高性能和低性能分开治理】*

*区分定义哪些有高性能需求的业务，哪些是低性能需求的业务；*

*阐述实现的关键设计实现途径*

*阐述区分出来的业务，互不影响的的关键设计实现途径*

*【示例3：可伸缩能力设计】*

*垂直扩展设计*

*水平扩展设计*

*负载均衡集群设计*

*分库分表如何设计*

*【示例4：读写分离的设计】*

*采用分离设计后，数据同步的性能指标，及其设计实现途径*

*【示例5：多级缓存设计】*

*前端缓存：*

* *APP缓存：SQLite、本地文件缓存、Picasso*
* *浏览器缓存：Cookie、LocalStorage、SessionStorage*
* *HTTP缓存：Last-Modified、Etag，Http状态码304*
* *CDN缓存：边缘服务器，用户就近访问；常用于：图片、视频、直播等资源的加速访问*
* *Web容器：是指反向代理缓存，常用于：静态资源：CSS、HTML、JS等文件*

*后端缓存：*

*1、本地应用缓存：*

* *进程内缓存：ConcurrentHashMap、Guava Cache，ehcache等*
* *进程外缓存：堆外内存 off heap cache*
* *本地磁盘SSD缓存：高速固态硬盘，也可以用来加速*

*2、分布式缓存：Redis、Memcached*

*3、操作系统层面：*

* *CPU缓存：L1、L2、L3*
* *内存缓存：Page Cache*

## 可靠性设计

### 异常处置设计

*系统局部异常时，需要设计故障处理预案（包含业务组件和中间件）：*

*考虑自身运行异常时的补偿措施，例如消息发送失败，通过补偿机制再次自动发送；网络闪断重连机制*

*考虑系统漏洞bug时的应急处置方案，非核心功能降级、自身补偿校对、人工处理方案等*

*考虑上下游系统调用异常时的保护机制，不阻断本系统的业务开展*

*高峰限流或熔断机制（单机、集群、网关、微服务、接口）*

*缓存失效，访问数据库*

*中间件宕机、中间价异常应急方案*

*【示例1：消息可靠性设计】*

*MC、MQ、KFK等中间件的消息可靠性设计*

*生产者发送可靠性设计*

*消费者接收可靠性设计*

*消息顺序，设计上要避免要求消息有序；若消息要求有序，需要增加发送序号，消费端保护*

*支持幂等：要求消费端需要支持接收重复消息，幂等处理*

*【示例2：数据一致性设计】*

*二阶段事务的处理*

*分布式事务三态的处理*

*业务数据的补偿机制*

*数据最终一致性*

## 安全设计

*恒生公司安全裁剪表要求的项目， 实现架构目标与设计约束 非功能性需求 的安全性需求*

## 可维护性设计

*可维护性设计目标：最终要满足架构目标与设计约束 非功能性需求 的可维护性需求*

### 安装升级

*提供安装升级工具，采用天鉴白屏安装部署升级；*

* *考虑支持升级前检查、方案化升级、升级后系统状态巡检*
* *支持选择最小配置、普通配置、豪华配置以满足不同场景的部署要求*
* *应用的快速发布部署能力，支持独立子系统或者模块*
* *考虑升级部署以及回退的程序支持，考虑支持盘中升级*
* *系统配置热起效：配置修改热起效，无需重启*
* *系统启动无依赖：启动顺序不依赖其他应用服务，如需依赖，需要发起评审*
* *考虑替换旧系统的数据迁移，提供数据迁移工具*

### 排查辅助

*提供问题排查定位的能力，包含但不限于以下内容*

* *日志规范按照规范记录日志。注意：前后端以及与三方系统交互时，包括调用与被调用，需要记录操作类的请求及应答信息*
* *日志考虑哦对接日志中心支持日志统一归集和查询*
* *支持调用的全链路分析*

### 生产环境测试设计

*通过天鉴提供一键备份，一键恢复，避免升级过程发生错误无法回退。同时考虑部分客户使用生产环境做交易所联测，需保证数据以及应用状态可以完全恢复，不会导致生产异常。*

## 可测试性设计

*可测试性设计目标：最终要满足架构目标与设计约束 非功能性需求 的可测试性需求*

### 测试数据设计

* *测试数据源是内置在系统中，还是外部提供，存储的形式和访问的方式：典型的设计：内存数据的预埋和修改*
* *测试数据源的控制是内置，还是外置*
* *测试数据的采集和处理是内置，还是外置：典型的设计：数据库操作的组件实现影子数据库，网关旁路实现生产环境数据引流到测试环境*
* *测试数据的处理是内置，还是外置*

### 单元测试和模块测试的可测性设计

*模块的调试和测试主要是指单元测试和模块测试。*

*关注点主要在于按设置测试控制序列、状态观测点和输入输出机制的设计，主要考虑以下几个方面的内容:*

* *提供模块调试测试的能控性设计：能通过输入设定的测试序列使系统处于某种特定的状态或满足某种特定条件的状态。主要考虑模块的调试测试控制点的选择和测试序列导入机制的设计。需要阐述业务链路上，各个节点控制逻辑的控制点的设计实现途径*
* *提供模块调试测试的能观性设计：能够通过系统的输出数据（例如接口返回、消息、内存、数据库数据等）判定系统是否处于某种特定的状态或满足某种特定条件的状态。主要考虑软件模块的调试测试观察点的选择和观察程序的设计。需要阐述业务链路上，各个节点业务逻辑的观察验证数据获取的设计实现途径*

*另外需要明确单元测试的方案和通过标准*

### 系统验证测试的可测性设计

*系统验证性测试是验证产品各种指标、性能是否达到设计要求，安全是否达到设计标准，高可能性是否达到测试标准。*

*在不同环境下的指标容限如何，关注点主要在于系统验证测试各种测试项目能否实现、实现是否方便、出现问题是否可以定位。主要考虑以下几个方面的内容:*

*（1）系统业务功能测试的可实现性和方便性的设计*

*（2）性能指标测试与瓶颈定位的可实现性和方便性的设计*

*（3）安全性测试与兼容性的可实现性和方便性设计*

*（4）高可靠性、高可用性测试的实现性和方便性设计*

*（5）系统告警功能验证测试的可实现性和方便性的设计*

*（6）系统容限/容错/极限性能测试的可实现性和方便性的设计*

*典型的设计有：*

*性能数据的统计和采集方式：全链路日志、分散记录日志、内存打点*

*系统状态数据的统计和采集方式：全链路跟踪分析、分散*

## 跨平台性设计

*本节需要满足公司或客户侧对于跨平台性的需求*

*跨平台设计：操作系统 数据库等*

*国产化：数据化、操作系统、芯片、浏览器、中间件等*

## 其他非功能性设计

*本节需要满足架构目标与设计约束 非功能性需求 的其他非功能性需求*

*新老系统切换时的数据迁移设计方案*

*二次开发可扩展性设计*

# 模板修订记录

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 版本号 | 批准日期 | 修订说明 | 编制部门 | 编制人员 |
| V1.0 | 2022/08/03 | 新增 | 产品技术管理总部 | 朱龙、寇旭、杨珍良、黄华、陆旭、张磊、赖毓泉 |