**综合监控（消息中间件）**

**概要设计说明书**

|  |  |  |  |
| --- | --- | --- | --- |
| 文档版本号： | V1.01 | 文档编号： | NG\_TS\_MQ\_SDS |
| 文档密级： | 内部公开 | 归属部门/项目： | AVP |
| 编写人： | Boris.F | 生效日期： | 2017-12-04 |

**版权信息**

本文件涉及之信息，属南京轨道交通系统工程有限公司所有。

未经南京轨道交通系统工程有限公司允许，文件中的任何部分都不能以任何形式向第三方散发。

网址：http://www.nanjingrail.com/

**文档修订记录**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **版本号** | **修订日期** | **修订人** | **修订说明** | **修订状态** | **审核日期** | **审核人** | **批准人** |
| V1.0 | 2017-12-04 | Boris.F | 正式版 | A |  |  |  |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |

修订状态：A--增加，M--修改，D--删除

日期格式：YYYY-MM-DD

**目　录**

[1. 引言 1](#_Toc287377107)

[1.1. 编写目的 1](#_Toc287377108)

[1.2. 背景 1](#_Toc287377109)

[1.3. 术语 1](#_Toc287377110)

[1.4. 预期读者与阅读建议 1](#_Toc287377111)

[1.5. 参考资料 1](#_Toc287377112)

[2. 总体设计 2](#_Toc287377113)

[2.1. 设计概述 2](#_Toc287377114)

[2.1.1 设计约束 2](#_Toc287377115)

[2.1.2 设计策略 2](#_Toc287377116)

[2.1.3 设计实现 2](#_Toc287377117)

[2.2. 设计目标 2](#_Toc287377118)

[2.3. 运行环境 2](#_Toc287377119)

[2.4. 平台架构 2](#_Toc287377120)

[2.5. 总体设计思路和处理流程 3](#_Toc287377121)

[2.6. 模块结构设计 3](#_Toc287377122)

[2.7. 功能需求与程序模块的关系（可选） 3](#_Toc287377123)

[2.8. 尚未解决的问题 3](#_Toc287377124)

[3. 接口设计 3](#_Toc287377125)

[3.1. 用户接口 3](#_Toc287377126)

[3.2. 外部接口 4](#_Toc287377127)

[3.3. 内部接口 4](#_Toc287377128)

[4. 界面总体设计 4](#_Toc287377129)

[5. 数据结构设计 4](#_Toc287377130)

[5.1. 设计原则 4](#_Toc287377131)

[5.2. 数据库环境说明 4](#_Toc287377132)

[5.3. 数据库命名规则 4](#_Toc287377133)

[5.4. 逻辑结构 4](#_Toc287377134)

[5.5. 物理存储 4](#_Toc287377135)

[5.6. 数据备份和恢复 4](#_Toc287377136)

[6. 系统出错处理设计[可选] 4](#_Toc287377137)

[6.1. 出错信息 4](#_Toc287377138)

[6.2. 补救措施 5](#_Toc287377139)

[6.3. 系统维护设计 5](#_Toc287377140)

[7. 系统安全设计 5](#_Toc287377141)

[7.1. 数据传输安全性设计 5](#_Toc287377142)

[7.2. 应用系统安全性设计 5](#_Toc287377143)

[7.3. 数据存储安全性设计 5](#_Toc287377144)

[8. 系统部署（可选） 6](#_Toc287377145)

# 引言

* 1. 编写目的

本概要设计主要描述的是在轨道交通综合监控系统消息中间件子系统的解决方案。保障系统实时高效、可靠、安全的运行。

* 1. 背景

由于综合监控系统的复杂性，网络通信的不确定性，以及消息数据的多样性，消息中间件需要满足跨网络、高可用、可解耦等要求，同时也能提供接口异步、数据同步等功能，其应用场景应满足如下要求：

1. 业务解耦：应用系统不需要知道消息通知服务的存在，只需要发布消息和接受消息；
2. 削峰填谷：比如上游系统的吞吐能力高于下游系统，在流量洪峰时可能会冲垮下游系统，消息中间件可以在峰值时堆积消息，而在峰值过去后下游系统慢慢消费消息解决流量洪峰的问题；
3. 事件驱动：系统与系统之间可以通过消息传递的形式驱动业务，以流式的模型处理；
4. 消息路由：中央层和车站层的中间件之间消息能互相转发至消费者；
5. 失败队列：当消费者连续处理接收到的消息失败次数超过阈值，则该消息将被缓存至失败队列，产生报警通知系统运维进行人工干预。
   1. 术语

中间件：是一种独立的系统软件或服务程序，分布式应用软件借助这种软件在不同的技术之间共享资源，中间件位于客户机服务器的操作系统之上，管理计算资源和网络通讯。

生产者：队列中消息的发起方，每次发起都会在队列中新建一条消息。

消费者：队列中消息的使用方，当使用完成后，消息从队列中移除。

主服务：高可用性主从热备模式下的功能真实提供方。

从服务：高可用性主从热备模式下的功能备份提供方。

API：Application Programming Interface，应用程序接口。

* 1. 预期读者与阅读建议

|  |  |
| --- | --- |
| **预期读者** | **阅读建议** |
| 系统领导层 | 仔细阅读概述，编写目的，文档约定，系统功能介绍和维度指标说明。 |
| 业务部门、决策部门、具体的使用部门、业务员、系统管理员 | 仔细阅读文档约定，系统功能介绍和维度指标说明。  各个部门可重点阅读与本部门相关的内容。 |
| 参加需求评审的人员 | 仔细阅读全部内容。 |
| 系统设计人员 | 仔细阅读全部内容。 |
| 系统测试人员 | 仔细阅读文档约定，系统功能介绍和维度指标说明。 |

* 1. 参考资料TBD

列出有关的参考资料，如：

a、本项目经核准的计划任务书或合同、上级机关的批文；

b、不属于本项目的其他已发表的文件；

c、本文件中各处引用的文件、资料、包括所要用到的软件开发标准；

d、列出这些文件资料的标题、文件编号、发表日期和出版单位，说明能够得到这些文件资料的来源。

# 总体设计

* 1. 设计概述

### 设计约束

功能需求约束：从整体上看，通信中间件的设计必须实现用户需求说明书中所要求的所有功能，如通信的可靠性级别、异步发送、削峰填谷、Topic多播、消息路由、主从互备等。其实现方式则可由开发人员自由选择。

独立性和通用性约束：通信中间件应当是一个相对独立的子系统，不应当向具体的应用层程序提出过多要求，不应当实现明显只针对特定应用的函数接口。整个中间件必须是一个尽可能通用的平台，各种不同的应用程序能够在上面运行良好。

模块化约束：通信中间件本身是一个比较大的子系统，为了方便开发和调试，中间件必须有良好的模块化。模块之间尽量独立，模块之内功能尽量单一。

### 设计策略

扩展策略：因为通信中间件是一个相对独立和通用的子系统，为了更好地实现通用性，各个功能模块和函数接口都应当是可扩展的。一些功能函数应当定义一些将来可能用到的参数，尽管这些参数所规定的功能在当前版本中不需要实现，但是必须考虑到以后的扩充。

### 设计实现

系统包含两大部分：

1. 独立的中间件守护线程，维护着发送任务队列和接收任务缓冲池，负责主机之间的数据传输、集群之间消息路由、主从之间冗余互备；
2. 程序SDK开发包，包含函数API（声明各个外部接口的头文件）、静态库（.a文件）和运行时动态链接库（.so文件），java客户端（.jar文件，java平台SDK）。应用层的程序经过编译链接以后，消息中间件SDK将成为应用程序的一部分，与消息中间件守护进程交互，交互的过程对应用层透明。

**图 1**描绘了该设计的数据传输过程，以中间件API提供的发送和接收功能为例说明。

应用程序

（调用mq\_send）

中间件接口线程

（调用\_mqsendto）

中间件Redis内核

（调用publish）

内核层协议栈

中间件Redis内核

（触发event）

应用程序

（调用mq\_receive）

中间件接口线程

（调用\_recvfrom）

**图 1数据传输过程**

* 1. 设计目标

异步：生产者发送消息后关闭会话，无需等待。可以通过应答监听或者主动查询应答缓存获得结果。消费者可以根据自己的状态和处理情况判断是否（及如何）获取消息，是否（及如何）执行操作并返回结果，生产者、消费者视作MQ的被调用者，可以自行决定该如何操作，比如可以批处理而无需返回和等待结果。

解耦：多种业务应用相互关联，容易造成底层数据分散，应用系统间的耦合度高。面向不同业务应用提供统一的数据访问服务，使用消息中间件对不同系统间的交互进行解耦。

* 1. 运行环境

|  |  |  |
| --- | --- | --- |
| **类别** | **标准配置** | **最低配置** |
| 计算机硬件 | TBD | Intel Xeon 双核CPU（64  位指令集兼容）1GB内存/40GB硬盘,两块相同的以太百兆网卡 |
| 软件 | TBD | CentOS 7 x64 |
| 网络通信 | TBD | TCP/IP协议栈支持，IP多播支持 |

* 1. 平台架构

整个轨道综合监控系统里，消息中间件在控制中心有主备两个集群，每个集群内的中间件服务是以主、从冗余互备架构；在车站层则是以主、从冗余互备单集群的形式部署。集群内部通过心跳和权值选举主服务，当心跳异常，内部选举失效的情况下，则通过第三方哨兵服务决定主从角色分配。具体从**图 2**中可以看出整个消息中间件的系统架构：

**MQ主**

**MQ从**

心跳

同步

**MQ主**

**MQ从**

心跳

同步

同步

**中心监控哨兵**

监控中心↑

车站

主集群

备用集群

**MQ主**

**MQ从**

心跳

同步

**车站监控哨兵**

消息路由（双通）

**图 2系统架构**

* 1. 总体设计思路和处理流程

本系统通过Redis队列组件缓存消息和应答。生产者与MQ建立连接成功后，自动打开发送Session和应答Session，完成消息发送后可关闭发送Session，业务线程无需阻塞等待，应答Session自动维护应答线程，接收MQ的推送，推送的消息是来自于消费者的应答。生产者和MQ的连接关闭后，发送Session和应答Session自动关闭，并回收资源。

生产者与MQ连接，MQ自动维护三个队列：消息发送队列，应答队列（域内消费者的应答缓存），跨域应答队列（跨域消费者的应答缓存）。消费者与MQ建立连接，MQ自动创建三个队列：消息接收队列（域内生产者消息缓存），应答队列，跨域消息接收队列（跨域生产者消息缓存）。跨域的消息和应答均通过路由服务代理转发，如**图 3**。

**图 3点对点异步消息流转架构**

订阅与广播采用无状态消息复制和异步推送模式，生产者循环向订阅主题的消费者广播Topic消息，且Topic消息并不会因为某个消息没有被处理而阻塞。消费者每次接收到Topic消息后，无需对生产者应答。如**图 4**

**图 4广播异步消息流转架构**

* 1. 制作购买或复用的分析

MQ子系统使用开源Redis作为队列数据缓存，其他模块均自主开发，所有模块采用代码编译集成的形式复用。

* 1. 模块结构设计

消息中间件是一个相对独立的子系统，包括通信守护进程和Redis内核两大部分，每一部分又可以细分为若干个小模块。它的总体结构如**图 5**所示

**系统配置**

**订阅**

**广播**

**消息**

**会话**

**通道**

**队列**

**消息路由**

**消息处理**

**主从互备**

**监控管理**

**日志记录**

**安全管理**

**图 5 MQ的总体结构**

### 消息会话

* 概念

上层应用在请求消息中间件为其进行数据传输之前，必须先与通信守护进程建立连接，这种连接被称为会话。有了会话的概念，便可以实现多个上层应用同时请求通信守护进程为其进行数据传输。当上层应用不再需要守护进程为其服务的时候，应当断开与守护进程的连接，这时候守护进程就会将会话注销。通信守护进程维护着一个会话列表。

* 会话缓存

上层应用与消息中间件建立连接时，会话管理模块自动连接消息中间件的所有主从服务，并将连接的会话缓存，与主服务的会话默认是激活状态。当消息中间件主从发生切换后，自动将从服务的会话激活，减少因主从变换而通信阻塞的时间。整个激活切换的过程对上层应用透明。

* 失效重连

主从服务与上层应用的会话之间存在双向心跳。当与从服务连接的会话失效时，按一定周期自动重连，提供给上层应用的服务正常使用。当与主服务练级的会话失效时，除了周期重连外，还需要查询当前系统是否发生了主从切换，如果接收到了主从切换的事件，则将从服务的会话激活，与原主服务的会话自动降级至从服务会话。如**图 6**所示：

开始

主会话状态？

主从服务

切换？

尝试连接

成功？

主从会话建立

有效

主从会话类型切换

是

否

是

从会话状态？

失效

失效

有效

否

**主会话状态有效时，上层应用能正常使用MQ服务**

**图 6 失效重连流程**

### 通道队列

一个消息能且只能被一个消费者接受。如果当前队列中没有任何的消费者，那么这条消息将会被队里存储起来，直到有一个消费者连接上。另外，如果消费者在接受到消息后，在他断开与MQ连接之前，没有发送ACK信息（可以是客户端手动发送，也可以是自动发送），那么这条消息将被发送给其他消费者。

* 发送通道

生产者通过发送队列将消息发至MQ，等待MQ将消息包进行分发处理：根据消息目标对象，将该消息发送至域内接收队列或者其他域的跨域转发队列。

* 接收通道

即域内工作队列，接收消息处理模块从发送队列、或者跨域接收队列转发过来的消息包。

* 跨域转发通道

处理跨域消息包转发的队列。

* 应答通道

消费者收到消息后进行消息处理，但是整个消息传递的过程还没有完成：视设置情况，消费者在完成某一条消息的处理后，将需要手动的发送一条ACK消息给对应的消息派发模块（默认一般设置为自动发送）。派发模块在收到这条ACK信息后，才会认为这条消息处理成功，并将这条消息从接收队列中移除；如果没有这条消息的ACK信息或者收到NACK信息，这条消息将会重新被派发给另外的并行消费者（如果不存在，则进入失败队列并报警）。

* 失败队列

消息在处理，派发，跨域路由过程，因异常而无法成功，则该消息将被缓存在失败队列，并通知系统触发报警。

* 队列优先级

队列初始化时，具有优先级属性，优先级高的队列中消息在处理、派发、跨域路由等操作中，高优先级会被率先执行，相同优先级之间仍然按照FIFO原则执行。

### 订阅广播

Topic 基于“订阅-广播”模式，当操作者发布一条消息后，所有对这条消息感兴趣的订阅者都可以收到它，即该消息被拷贝成多份，进行分发。只有当前“活动的”订阅者能够收到消息（如果当前Topic主题中没有订阅者，这条消息将被丢弃）。如**图 7**

Produce

Exchange

Broker

Channel

Channel

Channel

**MSG**

**MSG**

**MSG**

**MSG**

**MSG**

Consumer

**MSG**

Consumer

**MSG**

Consumer

**MSG**

Channel

**图 7 订阅与广播**

### 消息处理

* TTL（Time to Live）

生产者在建立MQ队列通信连接时，设置队列中，该生产者所创建消息的TTL，超过生存时间的消息包，自动转移至失败队列。

* 应答方式

生产者在建立MQ队列通信连接时，设置队列中，该生产者所创建消息的应答方式：自动、手动、无应答。消费者在接收消息后，需要按照生产者指定的规则，自动、手动等方式处理每个消息的应答。

* 消息分发

消息分发含有两种模式：点对点消息传输、一对多消息拷贝和广播。

### 消息路由

* 路由表

两个不同域内的MQ之间，通过网络建立连接，双方连接成功后更新各自的路由表。跨域的消费者，通过路由代理，直接监听生产者所在服务端的消息队列。当生产者发送消息时，生产者所在的消息中间件，在路由表中查询目标域的连接，并将消息直接转发。跨域消费者接收消息后，产生应答包，MQ查询路由表获取发送者所在域，并将应答推送至生产者所在域。

* 心跳

建立连接绑定的跨域MQ之间，维持心跳，失效后能自动重连。

* 跨域路由

能够处理消息跨域路由的前提是：至少已经和某个其他域的Exchange形成了绑定关系，并设置好了到这些Exchange的路由规则。在本地Exchange收到消息后，会根据设置的路由规则，将消息发送到符合要求的跨域转发队列中，跨域路由模块将消息转发至异地Exchange。

**图 8 跨域异步消息处理时序图**

### 主从互备

接收来自冗余服务的主从命令，自主切换主从服务。具体参考《系统管理——冗余服务》。无磁盘主从复制。

### 系统配置

以配置文件的形式，将系统级别的个性化描述符文本方式存储，配置文件与程序隔离，方便运维人员调整描述符参数。

* 日志文件：mq.logging.file
* 日志级别：mq.logging.level，默认DEBUG
* 日志数量：mq.logging.max-days，默认7天
* 日志大小：mq.logging.size，默认10M
* 通信协议：mq.transport.connector，如tcp://0.0.0.0:61616
* 最大连接数：mq.transport.max-connections
* 服务主机：mq.host.name，如ISCS-CEN
* 跨域协议(服务端)：mq.romote.connector，如tcp://0.0.0.0:3090
* 跨域绑定：mq.romote.exchages，如tcp://192.168.1.200:3090;udp://192.168.1.100:3090
* 跨域绑定心跳周期：mq.remote.exchage.heartbeat，主动连接方发起心跳
* 消息队列最大消息数：mq.queue.max-messages
* 失败队列最大消息数: mq.dlq.max-messages
* 接口安全账号密码：mq.security.policy，如user1:passwd1;user2:passwd2

### 安全管理

安全管理模块主要用来验证一个应用层连接的有效性，即用户名、密码是否正确。非法账号无法建立连接，并通过接口向应用层返回错误码。

### 日志记录

日志记录子模块主要功能是，将MQ运行的信息记录下来，日志的信息可以按照日志级别过滤，分级输出至日志文件，以便管理人员分析、定位异常。

写日志文件本身也是一种费时的IO操作，因此系统应创建一个日志记录线程负责将日志项写入文件，同时设计了一个日志项环形缓冲池。日志记录线程空闲时处于睡眠状态，程序每到一处需要记录的时候就将日志项写入环形缓冲池，同时唤醒日志记录线程，日志记录线程将缓冲池中的每一项写入文件以后，继续睡眠，等待下一次被唤醒。

### 监控管理

MQ健康监控的指标有：各队列或广播的消息堆积数量、入列数量、出列数量，跨域队列或者主题的路由入数量、路由出数量，死信队列失败消息数量。各指标按照队列分类，以表格方式界面显示。

* 1. 功能需求与程序模块的关系（可选）

本条用一张如下的矩阵图说明各项功能需求的实现同各块程序模块的分配关系，本表可用需求跟踪距阵替代：

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | 程序1 | 程序2 | …… | 程序m |
| 功能需求1 | √ |  |  |  |
| 功能需求2 |  | √ |  |  |
| …… |  |  |  |  |
| 功能需求n |  | √ |  | √ |

* 1. 尚未解决的问题

说明在概要设计过程中尚未解决而设计者认为在系统完成之前必须解决的各个问题。

# 接口设计

* 1. 用户接口
* 简单消息发送接口

interface SimpMessageSendingOperations {

SimpMessageSendingOperations(PooledConnectionFactory connectionFactory);

Boolean send(String destination, Json payload, long timeout=INDEFINITE\_TIMEOUT) throws MessagingException;

Boolean send(String host, String destination, Object payload, long timeout=INDEFINITE\_TIMEOUT) throws MessagingException;

}

* 简单消息接收接口（同步阻塞查询）

interface SimpMessageReceivingOperations{

SimpMessageReceivingOperations(PooledContainerFactory connectionFactory);

Json receiveFrom(String domain, String destination, long timeout=INDEFINITE\_TIMEOUT) throws MessagingException;

}

简单消息发送和接受接口，每次执行时首先从MQ连接池中申请连接，完成发送后释放连接，生产者对消息的接受状态不处理。消费者阻塞接受，完成消息接收后，在返回前释放连接至连接池。消息是无应答模式的单向传输。

* 高级消息发送接口

interface MessageSendingOperations {

MessageSendingOperations(PooledConnectionFactory connectionFactory);

Boolean send(String destination, Message<?> message, long timeout=INDEFINITE\_TIMEOUT) throws MessagingException;

Boolean send(String host, String destination, Message<?> message, long timeout=INDEFINITE\_TIMEOUT) throws MessagingException;

}

* 生产者

class MessageProducer implement MessageSendingOperations {

}

* 高级消息接收接口（消息监听）

interface MessageListener{

void onMessage(Message message) throws MessagingException;

}

* 消费者

class MessageConsumer {

void setMessageListener(MessageListener listener);

Message<?> receive(long timeout=INDEFINITE\_TIMEOUT) throws MessagingException;

}

* 连接工厂

interface ConnectionFactory {

ConnectionFactory(String user, String passwd);

Connection createConnection();

void close();//关闭连接池中所有连接

}

* 连接池工厂

interface PooledConnectionFactory extends ConnectionFactory{

PooledConnectionFactory(String user, String passwd, int poolSize);

Connection createConnection();

void close();//关闭连接池中所有连接

}

* 连接

class Connection {

Session createSession();

}

* 队列

class Queue {

Queue(String destination);

Integer priority {

get; set;

};

}

* 主题广播

class Topic extends Queue {

Topic(String destination);

}

* 会话

class Session {

Producer createProducer(Queue queue);

Consumer createConsumer(Queue queue);

}

* 1. 外部接口

Redis Queue接口依赖：

* PUBLISH(queueOrTopic, value)：向Redis发送一条消息
* SUBSCRIBE(queueOrTopic)：向Redis注册监听接口
* SUBSCRIBE(queueOrTopic)：从Redis中注销监听接口
  1. 内部接口

### 消息包

* 消息基类

class Message<T> {

T payload; //消息载体

Long id; //MQ自动生成uuid的hashcode

Date timestamp；//发送时间

Long ttl; //TTL

Integer version; //消息包版本

String fromRegion; //消息起源服务器唯一标识

String destRegion; //消息目的服务器唯一标识

String destQueue; //消息队列标识

List<String> routingTracks; //消息在域内的路由轨迹

}

* 点对点消息请求

class RequestMessage extends Message<T> {

//auto：MQ自动ack；client：消费者主动ack；non：无需应答

Enum replyMode;

}

* 消息广播

class TopicMessage extends Message<T> {

//auto：MQ自动ack；client：消费者主动ack；non：无需应答

Enum replyMode;

}

* 消息应答

class ResponseMessage extends Message<T> {

Message<T> request;

}

### 发送通道

class MessageChannel {

Boolean send(Message<?> message, long timeout=INDEFINITE\_TIMEOUT);

}

### 接收通道

class PollableChannel {

Message<> receiveFrom(long timeout=INDEFINITE\_TIMEOUT);

}

### 消息分发

* 消息分析协议接口

interface MessageHandler{

//消息处理

void handleMessage(Message<?> message)throws MessagingException;

Boolean subscribe(MessageChannel channel);

Boolean unsubscribe(MessageChannel channel);

}

* 消息分析实现

class ExchangeMessageHandler implement MessageHandler{

}

### 消息路由

interface MessageRemoteHandler extends MessageHandler{

}

* 消息路由

class MessageRemoteRouterHandler implement MessageRemoteHandler{

}

* 1. C语言接口

### MQ客户端初始化

参数hosts:MQ服务器Protocol://IP:Port集合，以“；”隔开。如tcp://192.168.1.120:6378;upd://192.168.1.130:6378。

int mq\_init(const char \*hosts, const char \*username, const char \*passwd, const int poolSize);

### 资源释放接口

MQ客户端资源回收，关闭回话、连接和连接池

int mq\_release();

### 消息发送

消息队列或主题广播名称queueOrTopic 变量内容，以queue://或topic://开头

int mq\_send(const char\* queueOrTopic, const char\* payload, const int size, const long timeout);

### 消息或者广播接收

int mq\_receive(const char\* domain, const char\* queueOrTopic, char\* buffer, const int size, const long timeout);

### 消息或者广播注册监听

int mq\_subscribe(const char\* domain, const char\* queueOrTopic, const MessageListener listener);

### 消息或者广播注销监听

int mq\_unsubscribe(const char\* domain, const char\* queueOrTopic, const MessageListener listener);

# 界面总体设计

说明界面总体布局和风格设计。

# 数据结构设计

可以在本文说明也可以单独使用数据库设计说明书描述

* 1. 设计原则

给出系统数据库的设计原则。

* 1. 数据库环境说明

简单介绍一些数据库直接有关的支持软件，如数据库管理系统、存储定位程序和用于装入、生成、修改、更新数据库的程序等。说明这些软件的名称、版本号和主要功能特性。

* 1. 数据库命名规则

联系用途，详细说明用于唯一地标识该数据库的代码、名称或标识符，附加的描述性信息亦要给出。

* 1. 逻辑结构

数据库设计人员根据需求文档，利用数据建模技术来描述逻辑数据库结构。要求使用PowerDesigner或Visio创建数据库PDM模型。此处只需列出PDM模型名称。

* 1. 物理存储

描述整个逻辑数据模型是如何被转换为数据文件（物理模式）。文件结构类型在这里应清楚的体现。

* 1. 数据备份和恢复

描述数据库的备份和恢复策略。

# 系统出错处理设计[可选]

* 1. 出错信息

用一览表的方式说明出错的类型，以及每种可能的出错或故障情况出现时，系统输出信息的形式、含意及处理方法。例如：

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **错误分类** | **子项及其编码** | **错误名称** | **错误代码** | **备注** |
| 数据库错误 | 连接 | 连接超时 | 100001001 |  |
| 连接断开 | 100001002 |  |
| 数据库本身错误代码 | 数据库本身错误代码 | 100002+数据库错误代码 |  |
| TCP连接错误 | 连接 | 连接超时 | 101001001 |  |
| 连接断开 | 101001002 |  |
| 其它TCP连接错误(socket自身错误代码) |  | 101002+ socket错误代码 |  |
| 配置信息错误 | 未配置输入参数 |  | 102001 |  |
| 未配置输出参数 |  | 102002 |  |

* 1. 补救措施

说明故障出现后可能采取的变通措施，包括：

a．后备技术 说明准备采用的后图示技术，当原始系统数据万一丢失时启用的副本的建立和启动的技术，例如周期性地把磁盘信息记录到磁带上去就是对于磁盘媒体的一种后备技术；

b．降效技术 说明准备采用的后备技术，使用另一个效率稍低的系统或方法来求得所需结果的某些部分，例如一个自动系统的降效技术可以是手工操作和数据的人工记录；

c.恢复及再启动技术 说明将使用的恢复再启动技术，使软件从故障点恢复执行或使软件从头开始重新运行的方法。

* 1. 系统维护设计

说明为了系统维护的方便而在程序内部设计中作出安排，包括在程序中专门安排用于系统的检查与维护的检测点和专用模块。

# 系统安全设计

* 1. 数据传输安全性设计

说明在数据通信和传输过程中安全性设计。

* 1. 应用系统安全性设计

说明在访问应用系统过程中用户以及访问权限、操作等安全性设计。

* 1. 数据存储安全性设计

说明在数据和文件在存储过程中的安全性设计。

# 系统部署（可选）

给出系统部署方案，尽量使用图表的形式，并辅以必要的文字说明。