**【项目名称】**

**概要设计说明书**

|  |  |  |  |
| --- | --- | --- | --- |
| 文档版本号： | V1.0 | 文档编号： | NG\_TS\_TSDB\_SDS |
| 文档密级： | 内部公开 | 归属部门/项目： | 专业系统部 |
| 编写人： | 张关举 | 生效日期： | 2017-12-13 |

**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 | 2017-12-13 | 张关举 | 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](#_Toc503283825)

[1.1. 编写目的 1](#_Toc503283826)

[1.2. 背景 1](#_Toc503283827)

[1.3. 术语 1](#_Toc503283828)

[1.4. 预期读者与阅读建议 1](#_Toc503283829)

[1.5. 参考资料 1](#_Toc503283830)

[2. 总体设计 2](#_Toc503283831)

[2.1. 设计概述 2](#_Toc503283832)

[2.1.1 设计约束 2](#_Toc503283833)

[2.1.2 设计策略 2](#_Toc503283834)

[2.1.3 设计实现 2](#_Toc503283835)

[2.2. 设计目标 2](#_Toc503283836)

[2.3. 运行环境 3](#_Toc503283837)

[2.4. 平台架构 3](#_Toc503283838)

[2.5. 数据处理流程 4](#_Toc503283839)

[2.5.1数据写入流程 4](#_Toc503283840)

[2.5.2数据读取流程 5](#_Toc503283841)

[2.6. 模块结构设计 6](#_Toc503283842)

[2.6.1 元数据存储模块 6](#_Toc503283843)

[2.6.2 预写日志模块 8](#_Toc503283844)

[2.6.3 Cache模块 10](#_Toc503283845)

[2.6.4 数据存储模块 12](#_Toc503283846)

[2.6.5 数据压缩模块 14](#_Toc503283847)

[2.6.6 文件管理模块 19](#_Toc503283848)

[2.6.7 数据同步模块 19](#_Toc503283849)

[3. 接口设计 19](#_Toc503283850)

[3.1. 用户接口 19](#_Toc503283851)

[3.1.1 用户管理接口 19](#_Toc503283852)

[3.1.2 数据库管理接口 20](#_Toc503283853)

[3.1.3 数据表管理接口 20](#_Toc503283854)

[3.1.4 数据管理接口 20](#_Toc503283855)

[3.2. 外部接口 21](#_Toc503283856)

[3.2.1 基于HTTP协议的接口 21](#_Toc503283857)

[3.2.2 开发语言级别接口 21](#_Toc503283858)

[4. 界面总体设计 22](#_Toc503283859)

[5. 数据备份和恢复 23](#_Toc503283860)

[6. 系统安全设计 23](#_Toc503283861)

[4.1. 数据传输安全性设计 23](#_Toc503283862)

[4.2. 应用系统安全性设计 23](#_Toc503283863)

[4.3. 数据存储安全性设计 23](#_Toc503283864)

# 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历史数据库子系统的解决方案。保障系统实时高效、可靠、安全的运行。

* 1. 背景

轨道交通综合监控系统需要监控各种设备的状态，包括温度、湿度、电压、电流等等，并且需要每时每刻把监控的数据记录下来，用来做后续的场景回放及分析。按照每条线路10万个设备，每个设备每秒钟产生一条数据来计算的话，每秒钟将会产生10万条数据，每天将会有86.4亿条数据。如此巨大的数据量，如果采用传统的关系数据库系统进行存储，存储成本会非常的高，并且随着数据量的增加，传统关系数据库的查询速度会有很明显的下降。

所以，在综合监控系统中使用传统的关系数据库存储设备的历史数据不合理，需要针对业务数据的特征研发新的数据库用来存储历史数据。

* 1. 术语

TSDB ： 基于时间序列的历史数据库

WAL ： 预写日志

* 1. 预期读者与阅读建议

本文档的预期读者及阅读建议如下：

|  |  |
| --- | --- |
| **预期读者** | **阅读建议** |
| 项目管理人员 | 仔细阅读文档的系统范围，编写目的，文档约定，系统功能介绍。 |
| 软件设计人员 | 仔细阅读全部内容。 |
| 软件开发人员 | 仔细阅读全部内容。 |
| 测试人员 | 仔细阅读全部内容。 |

* 1. 参考资料

《综合监控（历史数据库）软件需求规格说明书》

# 总体设计

* 1. 设计概述

### 2.1.1 设计约束

功能需求约束：从整体上看，历史数据库软件的设计必须实现用户需求说明书中所要求的所有功能，如数据的存储与查询、数据同步、SQL支持（部分支持）、访问权限控制、冗余集群等。其实现方式则可由开发人员自由选择。

独立性约束：历史数据库软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。

模块化约束：历史数据库软件本身是一个比较大的子系统，为了方便开发和调试，历史数据库软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。

### 2.1.2 设计策略

扩展策略：历史数据库软件作为轨道交通综合监控系统的数据存储的基础设施，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。

### 2.1.3 设计实现

系统包含三大部分：

1、独立的历史数据库守护进程，负责轨道交通综合监控系统数据的压缩、存储，数据同步、主从设备之间冗余互备。

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）。应用层的程序经过编译链接以后，历史数据库的SDK将成为应用程序的一部分，与历史数据库守护进程交互，交互的过程对应用层透明。

3、独立的历史数据库管理软件，包含基于控制台的数据库管理软件和基于图形界面的数据库管理软件。可实现对历史数据库软件的基本管理，包括数据库的管理、数据表的管理、数据查询、数据更新、数据删除等操作。

* 1. 设计目标

存储容量：历史数据库软件存储的数据应进行压缩存储，其存储容量应只受磁盘容量的限制。

写入速度：在内存和CPU资源充足的条件下，历史数据库软件的写入速度应只受磁盘写入速度的限制。

查询速度：在同等内存和CPU资源下，历史数据库软件的查询速度不应随着数据规模的扩大而剧烈降低。

* 1. 运行环境

|  |  |  |
| --- | --- | --- |
| **类别** | **标准配置** | **最低配置** |
| 计算机硬件 | Intel Xeon 12核CPU（64  位指令集兼容）64GB内存/1000GB硬盘,两块相同的以太千兆网卡 | Intel Xeon 双核CPU（64  位指令集兼容）2GB内存/100GB硬盘,两块相同的以太百兆网卡 |
| 软件 | CentOS 7 x64 | CentOS 7 x64 |
| 网络通信 | TCP/IP协议栈支持 | TCP/IP协议栈支持 |

* 1. 平台架构

整个轨道综合监控系统里，历史数据库在控制中心有主备两个集群，每个集群内的历史数据库服务是以主、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。

具体从下图中可以看出整个历史数据库的系统架构：



* 1. 数据处理流程

### 2.5.1数据写入流程

历史数据库软件数据写入流程如下：



流程描述如下：

1. TSDB通过网络从客户端接收数据；
2. TSDB将接收的数据通过追加方式写入WAL文件，同时将数据复制到Cache中；
3. 当WAL文件大小达到预设值后，TSDB会创建新的WAL文件，原先的WAL文件的属性将变为只读，并将该文件对应的Cache中的内容经过排序和压缩后存放到TSDATA文件中，同时在内存中添加这部分数据的索引信息；
4. 删除原先的WAL文件。

### 2.5.2数据读取流程

TSDB数据读取流程如下：



流程描述如下：

1. TSDB接收客户端的查询指令；
2. TSDB根据指令中的Key及约束条件在Cache中查找，如果查询成功，直接返回结果；
3. 如果在Cache中未查询成功，TSDB会从MemoryIndex中查询；
4. 如果TSDB在MemoryIndex中查询成功，TSDB会根据MemoryIndex提供的信息从TSDATA文件中读取数据，解压后返回查询结果；

如果TSDB在MemoryIndex中未查询成功，则本次查询失败；

* 1. 模块结构设计

TSDB整体结构可分为两部分：

* 元数据存储

元数据存储主要用于存储数据库的数据库结构、表结构等信息。

* 数据存储

数据存储部分主要由Cache模块、WAL存储模块、TSDATA存储模块、数据压缩模块、文件管理模块构成，整体结构如下：



### 2.6.1 元数据存储模块

在TSDB中，元数据是为了描述最终存储数据的数据。元数据能够提供基于用户的信息，如记录数据项的业务描述信息能够帮助用户使用数据。其次，元数据能支持系统对数据的管理和维护。具体来说，在TSDB中，元数据机制主要支持以下几类功能：

描述数据库表结构信息；

描述数据碎片分布情况；

描述数据碎片时间信息；

描述数据库用户信息；

#### 存储结构

元数据文件存储结构如下：



索引区存储索引信息，可用于区别服务器的不同实例；

逻辑结构存储区用于存储数据库的名称、存储策略信息等；

用户信息存储区用于存储数据库的用户信息及相关权限设置；

边界数据存储区用于存储系统中的边界数据，为系统的优化提供支持；

#### 数据结构

索引区和边界数据存储区记录的数据比较简单，这里主要描述下逻辑结构存储区和用户信息存储区的数据结构。

##### 逻辑结构存储区

逻辑结构存储区存储系统中所有数据库的存储策略信息，数据结构如下：

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Name**  **（N bytes）** | **DefaultRetentionPolicy（N bytes）** | **RetentionPolicies**  **（N bytes）** | **…** | **Name**  **（N bytes）** | **…** |

字段解析如下：

* Name

该字段为字符串类型，用于记录数据库名称，支持建立不同的数据库。

* DefaultRetentionPolicy

该字段为字符串类型，用于记录默认存储策略，内容为后续存储策略信息中的名称。

* RetentionPolicies

该字段为数组类型，用于记录数据库可以使用的存储策略的集合。

##### 用户信息存储区

用户信息存储区存储系统中的所有用户的认证信息及权限数据，数据结构如下：

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Name**  **（N bytes）** | **Password**  **（N bytes）** | **AdminFlag**  **（1 byte）** | **Privileges**  **（N Bytes）** | **…** | **Name**  **（N bytes）** | **…** |

字段解析如下：

* Name

该字段为字符串类型，用于记录用户名。

* Password

该字段为字符串类型，用于记录当前用户对应的密码信息。

* AdminFlag

该字段为bool类型，用于记录当前用户是否为管理员。

* Privileges

该字段map类型，用于记录当前用户所拥有的各项权限。

### 2.6.2 预写日志模块

WAL的全称是Write Ahead Log，WAL模块的主要作用是防止数据丢失。

WAL的存储方式在数据写入时执行的是附加操作，将随机写转换为顺序写来大幅提高写入操作的性能。

当TSDB收到写入和删除的操作指令时，并不是立即执行该操作，而是将这些指令存入到WAL文件中，后续一起执行，不会影响数据写入速度。

当WAL文件的大小大于设定的值时，TSDB会关闭当前WAL文件并创建新的WAL文件用于写入。

#### 存储结构

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **DataType**  **（1 byte）** | **DataLen**  **（4 bytes）** | **DataValue**  **（N bytes）** | **…** | **DataType**  **（1 byte）** | **…** |

字段解析如下：

* DataType

用于标记数据操作类型，支持的操作类型有写入、删除、连续删除，由于该存储结构的特性，更新操作可以通过写入实现（相同的key重新写一遍即可）。

* DataLen

后面数据的长度。

* DataValue

具体数据字段。

#### 数据结构

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Type**  **（1 byte）** | **Key Len**  **（2 bytes）** | **Key**  **(N bytes)** | **Count**  **(4 bytes)** | ***Time***  ***(8 bytes)*** | ***Value***  ***(N bytes)*** | **…** | **Type**  **(1 byte)** | **…** |

字段解析如下：

* Type

表示存储数据的类型，支持的类型有：

浮点型，整型，布尔型，字符串型。

* Key Len

指定紧跟其后的Key的长度。

* Key

Key具体内容。

* Count

后面跟的（Time + Value作为一个整体）数据的个数。

* Time

单个value的时间戳。

* Value

需要存储的数据，支持的类型有：

浮点型，整型，布尔型，字符串型。

当Value为字符串类型时，Value 由于两部分构成：

字符串长度（4Bytes） + 字符串内容（长度由前面决定）。

其它类型由于是定长的，无需再标识长度。

### 2.6.3 Cache模块

Cache的主要作用是让WAL文件中的数据变得可查询。



由于WAL文件适合快速写入，但不适合查询，如果直接从WAL查询数据，速度会非常慢，所以每次数据写到WAL文件时，TSDB会同时向Cache中写一份，以加速查询。

Cache分为两个部分：热区域和冷区域。



热区域主要存储最近写入的数据，冷区域存储的是已关闭WAL文件的快照，冷区域具有只读属性，热区域的才可写入，这样可以避免写操作的阻塞。



如果Cache满了，或者空闲超过一定时间，TSDB会对当前Cache做一次快照，当前Cache将被标记成冷区域，同时关闭当前Cache对应的WAL文件。然后TSDB将当前Cache中的数据经过排序、压缩后以特定格式写入TSDATA文件中。写入TSDATA文件成功后，TSDB将释放当前Cache的快照，并清理对应的WAL文件。

TSDB启动时，会将上次运行过程中未处理的WAL映射到Cache中。

#### 存储结构

|  |  |  |
| --- | --- | --- |
| **CacheDataArea**  **（N bytes）** | **SnapshotArea**  **（N bytes）** | **StatisticsArea**  **（N bytes）** |

解释如下：

* CacheDataArea

该部分存储当前cache的内容，即Cache的热区域。

* SnapshotArea

该部分存储cache的快照，即Cache的冷区域，用于写入TSDATA文件，只读属性。

* StatisticsArea

该部分记录一些统计信息。

#### 数据结构

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **CacheData**  **（N bytes）** | **CacheSize**  **（8 bytes）** | **CacheMaxSize**  **（8 bytes）** | **Snapshot**  **（N bytes）** | **SnapshotSize**  **（8 bytes）** | **SnapshotFlag**  **（1 byte）** |

解释如下：

* CacheData

一个字典类型的数据存储结构，用于存储具体的缓存数据的指针。

* CacheSize

用于记录当前Cache的大小，每当 Cache 中的数据达到阀值后，会将当前的 Cache 进行一次快照，之后清空当前 Cache 中的内容。

* CacheMaxSize

Cache中的数据阀值。

* Snapshot

快照的指针。

* SnapshotSize

快照的大小。

* SnapshotFlag

用于标记是否正在处于快照状态。执行快照操作时该值设置为true，清理快照时该值被设置为false。

### 2.6.4 数据存储模块

WAL的存储方式将随机写转换为顺序写来大幅提高写入操作的性能，但是牺牲了部分读的性能。

由于综合监控系统业务数据本身的特性，运用WAL的方式非常合适。持续写入数据量大，数据和时间相关，编码到 key 值中很容易使 key 值有序。读取操作相对来说较少，而且通常不是读取单个 key 的值，而是一段时间范围内的数据，这样就把读取性能差的劣势缩小了，反而由于数据是按照 key 值顺序排列，读取大块连续的数据时效率也很高。

TSDATA中存储的数据就是Cache中的数据在磁盘的有序存储，其内部数据是根据 key 从小到大排列的。通常为了加快查找的速度，需要加入数据索引，可以快读定位到指定的 k-v 数据。

TSDATA通常采用的分级的结构，当数据达到指定阀值后会在 Level 0 层创建一个新的存储结构。当某个 Level 下的文件数超过一定值后，就会将这个 Level 下的一个存储文件和更高一级的存储文件合并，由于存储中的 k-v 数据都是有序的，相当于是一个多路归并排序，所以合并操作相当快速，最终生成一个新的存储文件，将旧的文件删除，这样就完成了一次合并过程。

#### 存储结构

TSDATA文件存储结构如下：

|  |  |  |  |
| --- | --- | --- | --- |
| **Header**  **（5 bytes）** | **Blocks**  **（N bytes）** | **Index**  **（N bytes）** | **Footer**  **（8 bytes）** |

* Header

用于标识文件类型及版本号。

* Blocks

用于存储数据。

* Index

用于存储Blocks的索引信息。

* Footer

用于标记Index在TSDATA文件中的偏移量。

#### 数据结构

##### Header结构

Headr由EngineType和Version构成，其中EngineType用于标记使用的存储引类别，Version用于记录版本号。

存储结构如下：

|  |  |
| --- | --- |
| **EngineType**  **(4 bytes)** | **Version**  **(1 byte)** |

##### Blocks结构

由多个Block构成，每个Block包含CRC32和Data两部分，其中CRC32用于校验Data的内容是否有问题， Data为存储的数据，Data的长度记录在之后的Index部分中。

存储结构如下：

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Block 1 | | Block 2 | | Block N | |
| CRC  4 bytes | Data  N bytes | CRC  4 bytes | Data  N bytes | CRC  4 bytes | Data  N bytes |

##### Index结构

存储结构如下：

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Key Len  (2 bytes) | Key  (N bytes) | Type  (1 byte) | Count  (2 bytes) | Min Time  (8 bytes) | Max Time  (8 bytes) | Offset  (8 bytes) | Size  (4 bytes) | … |

字段解析如下：

* Key Len

代表紧随其后的key的长度

* Key

Key的内容

* Type

存储数据的类型

* Count

后面跟的（Min Time + Max Time + Offset + Size作为一个整体）数据的个数

* Min Time

block中value的最小时间戳

* Max Time

block中value的最大时间戳

* Offset

该block在TSDATA文件中的偏移量

* Size

该block的大小

##### Footer结构

该部分占用8个字节，用于标记Index在TSDATA文件中的偏移量。

### 2.6.5 数据压缩模块

数据压缩模块由数据压缩器和数据压缩管理器构成，数据压缩器负责提供各种算法用于数据压缩，数据压缩管理器会有计划的进行数据压缩及压缩文件的合并。

#### 数据压缩器

压缩是个持续的过程，数据压缩器主要功能如下：

* 将已经关闭的WAL文件转换为TSDATA文件
* 将多个比较小的TSDATA文件合并为一个大的TSDATA文件，进一步提高压缩率

#### 数据压缩算法

为达到较快的压缩和解压速度，同时减少对CPU的消耗，TSDB针对不同的数据类型采用不同的压缩算法。

##### 整型数据压缩

1、压缩流程描述



首先使用ZigZag算法进行编码，如果编码后的值小于 (1 << 60 ) - 1，使用simple8b算法；

如果大于(1 << 60 ) - 1，则不执行不压缩操作。

2、ZigZag算法描述

 ZigZag这个算法使用的基础就是认为在大多数情况下，我们使用的数字都是不大的数字。其编码过程如下：

1）获取int64类型输入X；

2）对X执行左移1位的操作，得到X1；

3）对X执行右移63位的操作，得到X2；

4）对X1和X2执行异或运算，得到ZigZag编码结果；

从编码过程可以看出，该算法的原理是将标志位后移至末尾，如果是负数则保留符号位移过来的1，非负数直接为0（异或操作），去掉编码中多余的前导0，则可以使用更少的字节来存储数据，从而达到压缩效果。

比如int64类型的数字1，其标志位为0，用二进制表示时前面会有63个0，最后一位才是1，执行位移操作后，X1为2，X2为0，执行异或操作后的值为2，前面有62个0，

去掉前面多余的0，仅用最后8位数表示，则编码后的数据为： 00000010 。

标志位后移主要是为了处理负数，比如int64类型的数字 -1 ，其标志位为1，用二进制表示时两端各有一个1，中间有62个0，执行位移操作后，X1为0xfffffffffffffffe，X2为0xffffffffffffffff，执行异或操作后的值为1，前面有62个0，去掉前面多余的0，仅用最后8位数表示，则编码后的数据为： 00000001 。

小整数对应的ZigZag码字短，大整数对应的ZigZag码字长。在特定的场景下，比如，要传输的整数为大整数居多，ZigZag编码的压缩效率就不理想了。而综合监控中的业务数据大部分是小数据，所以非常适合使用该算法。

该算法的解码过程如下：

 1）获取ZigZag编码结果V；

2）对V执行右移1位的操作，得到结果V1；

3）将V与1相与，得到中间值，将中间值左移63位，然后右移63位，得到结果V2；

4）对V1和V2执行异或操作，得到结果X；

3、Simple8b算法描述

Simple8b算法是64位算法，实现将多个整型数据（在 0 和 1<<60 - 1 之间）压缩到一个64位的存储结构中。

其中前4位为选择器，后面60位用于存储数据，数据使用下表进行编码：

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Selector | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
| Bits | 0 | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 10 | 12 | 15 | 20 | 30 | 60 |
| N | 240 | 120 | 60 | 30 | 20 | 15 | 12 | 10 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 |
| Wasted  Bits | 60 | 60 | 0 | 0 | 0 | 0 | 12 | 0 | 4 | 4 | 0 | 0 | 0 | 0 | 0 | 0 |

编码过程如下：



这里以30个3的数组为例描述下编码过程：

1）该数组中的最大数据为3，可以使用2位二进制表示。

2）从表中可以看出，Selector等于0、1、2时都不满足编码条件，当Selector为3时，条件满足：每2个bit存储一个数据，可以存储30个数据。

3）最终数据由选择器和存储数据组成。

其中前4个bit数据为： 0011

后面存储了30个3，则后面60个bit数据为：0xfffffffffffffff

两部分数据合并在一起表示：0x3fffffffffffffff

因此，30个3使用该算法压缩后可表示为： 0x3fffffffffffffff

如果上面的30个3都使用int64进行存储，该算法的压缩后占用空间为原来的 3.3%；如果上面的30个3都使用int32进行存储，该算法的压缩后占用空间为原来的 6.7%；如果上面的30个3都使用int8（即一个Byte）进行存储，该算法的压缩后占用空间为原来的 26.7% 。

由上面的例子可以看出，该算法针对使用int64和int32存储数据的场景压缩效果是比较明显的，如果存储数据的范围波动比较大，需要使用64位或32位的int进行存储，但大部分数据的绝对值比较小（比如可以使用一个字节存储），则使用该算法的压缩效果比较明显。而综合监控系统中的数据刚好满足上述条件，因此作为存储系统的历史数据库，可以使用该算法对最终数据进行压缩后存储，节省磁盘空间。

该算法的解码过程如下：



1）首先获取压缩数据V的前4个bit作为Selector的值；

2）如果Selector的值大于或等于16，直接出错返回；

3）如果Selector的值小于16，执行解码操作：根据不同的Selector值选取不同的解码规则进行解码操作。

##### 时间戳类型数据压缩

时间戳为独立的数据类型，并且具有一定的规律可循，在TSDB中，针对时间戳先执行排序操作后使用差分编码算法进行编码，然后再根据编码结果采用不同的算法。

1、压缩流程描述



解释如下：

1）根据输入的原始数组arrValues计算出差值数组deltaValues；

2）如果差值数组的所有值相同，使用RLE编码算法；

3） 如果差值数组的所有值不同，并且差值数组的最大值大于（1 << 60）- 1，使用Raw编码算法；

4）如果差值数组的所有值不同，并且差值数组的最大值不大于（1 << 60）- 1，使用Packed编码；

2、编码算法描述

1）RLE编码算法描述

使用该算法的前提是差值数组的所有数值都相同。使用该算法进行编码时，其存储结构如下：

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| EncodeType | Divisor | Timestamp | DeltaValue | N |
| 4 bits | 4 bits | 8 Bytes | 8 Bytes | 8 Bytes |

解释如下：

EncodeType ： 记录编码类型，占4个bit；

Divisor ：记录除数的log10值，占4个bit；

Timestamp ： 记录第一个时间戳的值；

DeltaValue ： 记录第一个差值；

N ： 重复次数；

该算法的核心思想是记录数据的重复次数，其存储结构的第一个字节的高4位用于记录该存储结构使用了RLE编码，后4位记录除数的log10值。由于差值数组是相对原始数组的第一个数据计算的，所以原始数组的第一个值（第一个时间戳）必须记录，即上述结构中的Timestamp字段。差值数组的所有值都相同，所以可以在存储结构中可以记录第一个差值和重复次数，即上述结构中的DeltaValue字段和N字段。

2）Raw编码算法描述

使用该算法的前提是差值数组的最大值大于（1 << 60）- 1。使用该算法进行编码时，其存储结构如下：

|  |  |
| --- | --- |
| EncodeType | RawData |
| 4 bits | N Bytes |

解释如下：

EncodeType ：编码类型，和其它结构兼容，第一个字节的前4个bit用于记录编码类型；

RawData ： 原始数组的数据；

该算法数据没有压缩，反而增加了一个字节。为了和其它结构兼容，第一个字节的前4个bit用于记录当前存储的数据使用的是Raw编码类型。

3）Packed编码算法描述

使用该算法的前提是在差值数组的所有数值均不同，并且差值数组中数据的最大值不大于（1 << 60）- 1 。使用该算法进行编码时，其存储结构如下：

|  |  |  |  |
| --- | --- | --- | --- |
| EncodingType | Divisor | Timestamp | Simple8bData |
| 4 bits | 4 bits | 8 Bytes | N Bytes |

解释如下：

EncodeType ：记录编码类型，占4个bit；

Divisor ：记录除数的log10值，站4个bit；

Timestamp ：记录第一个时间戳的值；

Simple8bData ：差值数组使用Simple8b算法编码后的结果；

该算法首先使用差值编码对原始数据进行编码，将编码后的值除于最大共同除数Divisor（10的倍数或1），使差分数组的值尽量缩小。然后将差值数组使用Simple8b算法进行编码，进一步提高压缩效果。

##### 布尔型数据压缩

布尔类型只有2个值，只占1位数据，采用简单的位数据打包策略。

##### 浮点型数据压缩

根据浮点数的存储特征，历史数据库软件使用XOR算法对数据进行压缩存储，其存储格式如下：

|  |  |  |  |
| --- | --- | --- | --- |
| **Flag** | **FirstValue** | **DeltaArray** | **NaN** |
| 8 bits | 64 bits | N bits | 64 bits |

解释如下：

Flag ： 数据压缩标记。

FirstValue ： 第一个数据。

DeltaArray ：压缩后的后续数据。

NaN ：结束标志。

下面描述下浮点数的存储格式及XOR算法。

**1、IEEE754标准的浮点数存储格式**

32位浮点数存储格式：

|  |  |  |
| --- | --- | --- |
| Sign | Exponent | Fraction |
| 符号位  （1 bit） | 阶码  （8 bits） | 有效数字  （23 bits） |

这种格式的非0浮点数的真值为：

64位浮点数存储格式：

|  |  |  |
| --- | --- | --- |
| Sign | Exponent | Fraction |
| 符号位  （1 bit） | 阶码  （11 bits） | 有效数字  （52 bits） |

这种格式的非0浮点数的真值为：

**2、XOR算法描述**

该算法是结合遵循IEEE754标准的浮点数存储格式的数据特征设计的特定算法，数据编码过程如下：

1）第一个值不压缩（记录为v0）；

2）计算后续值v与第一个值v0的异或值vDelta；

3）如果vDelta为0（即：v与v0的值相同），接下来的一个bit存储一个0（占用一个bit）；

4）如果vDelta不为0（即：v与v0的值不相同），接下来的一个bit存储一个1（占用一个bit），然后根据vDelta的值分以下两种情况进行处理：

如果重置前导值或尾数存储空间更优化，则按如下流程处理：

a）接下来的一个bit写入1；

b）接下来的 5 个 bit 写入vDelta值（二进制表示）中前导0的个数leading；

c）接下来的 6 个 bit 写入vDelta值（二进制表示）中有效位大小sigbits；

d）将vDelta值（二进制表示）右移去掉后面多余的0（长度前面有效数字已经标记过）得到vDelta2，写入vDelata2的值（仅有效长度）；

如果重置前导值或尾数存储空间没有达到更优效果，则之前使用之前的参数，按如下流程处理：

a）接下来的一个bit写入0；

b）将vDelta值（二进制表示）右移去掉后面多余的0（长度前面有效数字已经标记过）得到vDelta2，写入vDelata2的值（仅有效长度）；

##### 字符串型数据压缩

历史数据库软件中存储的字符串类型的数据使用Snappy算法进行压缩。Snappy的目标不是最大限度压缩或者兼容其他压缩格式，而是旨在提供高速压缩速度和合理的压缩率。

### 2.6.6 文件管理模块

在TSDB中有大量的数据文件，该模块的作用如下：

1、管理TSDATA文件；

2、保持文件索引和引用的活跃；

### 2.6.7 数据同步模块

该模块主要用于保持双机热备系统数据的一致性，需要配合其它系统一起使用。需要有两台或两台以上的服务器完成相同的功能，共同执行同一服务，让它们具有相同的配置，彼此互为备用，当一台服务器出现故障时，不需要人工介入，可以由另一台服务器自动接替工作，保证系统持续运转，实现高可用性。

数据同步分为以下几种情况：

* 主机到备机之间的数据同步
* 车站服务器到中央服务器的数据同步

# 接口设计

* 1. 用户接口

本软件为历史数据库系统，考虑到系统的通用性和易用性，应给TSDB提供基于SQL的接口。

### 3.1.1 用户管理接口

TSDB提供用户管理相关接口，具体如下：

* 添加用户接口

用于添加数据库用户，该功能需要管理员权限才能操作。

* 删除用户接口

用于删除数据库用户，该功能需要管理权限才能操作。

* 修改用户密码

用于修改数据库用户对应的密码信息，修改前需对用户信息进行认证。该功能对系统管理员和当前用户有效。

* 更新用户权限

用户权限的类别有：无操作权限、数据读取权限、数据写入权限、所有权限。

针对权限的操作有：请求权限、授予权限、撤销权限。

### 3.1.2 数据库管理接口

TSDB提供数据库管理接口，具体如下：

* 创建数据库接口

用于创建数据库，该功能需要管理员权限才能执行。

* 删除数据库接口

用于删除数据库，该功能需要管理员权限才能使用。

* 查询数据库列表接口

用于查看当前TSDB中有那些数据库，该功能所有用户都可使用。

* 数据库切换接口

用于改变当前工作数据库的接口。

### 3.1.3 数据表管理接口

历史数据库软件查询语言支持对数据表进行管理的操作，具体如下：

* 建表接口

数据表建立接口，用于创建数据表。

* 表结构更新接口

表内字段新增操作和表内字段删除操作等。

* 删表接口

删除数据表的接口。

* 查询数据表列表接口

查询当前数据库中有那些表。

### 3.1.4 数据管理接口

TSDB提供数据管理接口，具体如下：

* 数据写入接口

用于向数据表中写入数据。

* 数据更新接口

用于更新数据表中的数据。

* 数据删除接口

用于删除数据表中的数据，支持删除满足条件的数据，支持删除整张表的数据。

* 数据查询接口

用于查询数据表中满足条件的数据，支持排序操作、支持条件查询、支持分页操作。

* 1. 外部接口

### 3.2.1 基于HTTP协议的接口

TSDB提供基于HTTP协议的接口，实现对数据的管理，具体如下：

* 用户认证接口

认证数据库用户信息，保障数据安全。

* 数据写入接口

向数据库写入数据，支持单条数据写入和批量数据写入两种模式。

* 数据查询接口

从数据库中查询数据，支持分页功能。

### 3.2.2 开发语言级别接口

#### 数据结构

##### 状态数据结构

该数据结构用于记录设备上报的原始数据，包含如下字段：

* 子系统id
* 站点id
* 设备id
* 数据点id
* 数据值对应的下标
* 数据值
* 数据值的类型
* unix时间戳（以毫秒为单位）

##### 事件数据结构

该数据结构用于记录报警数据，包含如下字段：

* 子系统id
* 站点id
* 设备id
* 数据点id
* 数据值
* 事件描述
* 事件级别
* 事件类型
* unix时间戳（以毫秒为单位）

#### 数据库访问接口

历史数据库软件提供的数据库访问接口，具体如下：

* 数据库初始化访问接口

初始化数据库访问信息，创建数据库访问对象。

* 数据库反初始化访问接口

销毁数据库访问对象，释放系统资源。

* 创建状态数据的接口

用于创建状态数据结构，并初始化。

* 释放状态数据的接口

用于释放状态数据结构。

* 创建状态数据存储结构的接口

用于创建状态数据存储结构，并初始化。

* 存储状态数据的接口

用于存储状态数据，支持批量模式。

* 查询状态数据的接口

用于查询状态数据，支持分页操作。

* 创建事件数据的接口

用于创建事件数据结构，并初始化。

* 释放事件数据的接口

用于释放事件数据结构。

* 创建事件数据存储结构的接口

用于创建事件数据存储结构，并初始化。

* 存储事件数据的接口

用于存储事件数据，支持批量模式。

* 查询状态数据的接口

用于查询事件数据，支持分页操作。

# 界面总体设计

历史数据库管理工具界面总体风格如下所示：

 可分为如下几个部分：

* 菜单栏

数据库的管理工具的集合，包括对数据库连接的管理，数据查询语句的编辑，数据导入导出工具等；

* 工具栏

数据库快捷管理工具的集合。

* 连接管理区

以树形控件的形式显示数据库连接、数据库、数据表之间的关系。

* 数据查询区

编辑数据库查询语句。

* 数据显示区

显示数据库查询结果。

* 查询日志显示区

显示数据库查询日志。

# 5. 数据备份和恢复

历史数据库软件支持备份与恢复操作，具体如下：

1、支持数据表级备份与恢复操作；

2、支持数据库级备份与恢复操作；

3、支持全量备份与恢复；

4、支持增量备份与恢复；

5、支持差异备份与恢复；

# 6. 系统安全设计

系统安全设计方面，主要通过访问控制模块，从系统安全的角度对历史数据库软件的用户及权限进行管理，具体如下：

* 基于用户名和密码的访问控制策略

通过用户名和密码进行认证后根据对应的权限对数据库进行访问。

* 基于IP地址的访问控制列表

历史数据库软件支持访问控制列表功能，未授权的IP地址不能访问数据库。

6.1 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。

6.2 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。

6.3 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。