多维时间序列数据综合管理系统

王　哲

院　（系）：计算机科学与技术学院　　专　　业：信息安全

学　　号：1130320116　　指导教师：韩琦

2017 年 **6** 月

哈爾濱工業大學

毕业设计（论文）

题　　目 多维时间序列数据综合管理系统

专　　业 信息安全

学　　号 1130320116

学　　生 王哲

指 导 教 师 韩琦

答 辩 日 期 2017.06.26

摘　要

数据的管理是数据时代的重要支撑技术，而随着信息技术的发展，时间序列数据的存储和管理变得越来越重要，如流程工业数据、社会经济数据、心电图数据以及磁场测量数据等。由于磁场测量数据有其自身的特点，采用传统的关系型数据库对这些时序数据进行处理，在检索和查询上很难满足需求。因此，有必要针对磁场数据的特点和使用环境开发一套相应的数据管理系统。

本文在分析了时间序列数据特点的基础上，研究了时间序列数据在使用各种不同数据存储方式为内核下的优缺点，选择采用时间序列数据库 InfluxDB 为内核进行存储数据。分析多种可能时序数据样例的数据格式特点，使用 Python 的便捷的数据处理能力，对多种时序数据格式进行统一格式化处理，提升数据存储效率，有效地管理变长度的磁场测量时间序列数据。根据磁场测量数据各个属性的测量频率，设计 json 格式的数据存储结构，方便数据存储和网络传输。定义多个 Python 函数作为系统接口，可以同时为其他时序数据使用程序提供数据传输的高效性和调用便捷性。使用 Flask 网页架构提供了良好的数据操作界面。最后使用了三种典型的磁场测量数据样例，在网页操作界面进行数据存储和管理测试并与其他时序数据使用程序完成联合测试。

测试结果表明，以时间序列数据库 InfluxDB 为内核，所开发的时间序列数据库能够很好的完成数据管理以及与其他程序的结合使用。InfluxDB 数据库相对于其他数据库有着对时序数据更好的存储设计和数据处理性能。时间序列数据库系统采用分块设计思想，按照功能需求开发模块，组建数据库，设计的数据库具有良好的扩展性和通用性。

关键词：时间序列；InfluxDB；数据管理；数据处理；接口设计

Abstract

The management of data is an important base technologe of data time, with the development of message technology, the storage and the management of time series data is becoming more and more important, such as process industry, social economy data, Electrocardiogram data, megnetic field measurement data and so on. Because of the feature of megnetic field measurement data itself, if we use traditional relational database to process these time series data, it will be hard to meet the need of query function. So it is necessary to develop a time series data management system for the feature and using environment of megnetic field data.

This article mainly research the advantages and disadvantages of using different data storage methods as the kernel of storing time series data based on the analysis of features of time series data, finally we choose InfluxDB as the kernel of database system. After analysing deatures of data formats of many probable time series data, we use Python's function of data process to process many formats of different time series data and to improve the efficiency of data storage, it can manage those variable length time series data efficiently. Based on the measuring frequency of each attribute of time series data, we design the data storage structure of json format to make the network transpotation and data storage convinent. We also define many Python functions as the interface of system, it can provide the efficiency of data tranpotation and convinency of data call for other time series data at the same time. This system uses Flask web structure to provide a good data operating interface. Finally we use three typical megnetic field measuring data as the example of testing to finish the test for system on the system platform and the test with other time series data using program.

The result of test indicate that the time series data management system which use InfluxDB as kernel can achive the need of data management and the combining use with other programs. Comparing with other databases, InfluxDB have better storage design and data process performance for time series data. This time series database system use the thought of block design to develop functions modules and to build databases, these designed databases have good scalability and versatility.

Keywords: time serise, InfluxDB, data management, data process, interface design

目　录

第 1 章 绪论

1.1 课题背景及研究的目的和意义

1.1.1 课题背景

地磁场是地球的基础资源之一，与人类的生活紧密相关，它在地球科学、航空航天、资源勘探、交通通讯、国防建设、地震预报等领域有着广泛的用途。正是因为有如此重要的应用价值，使得人们对地磁场的测量有着迫切的需求。磁场的测量已经成为当下的热门课题。随着信息技术的飞速发展，磁场测量的方法日新月异。根据磁场测量的特点，以及对测量环境和测量精度的特殊要求，磁场测量出现了测量数据全球化，更新周期快速化和数据处理自动化的趋势。这就意味着所要分析的磁场测量数据越来越多，合理且高效的数据管理系统的需求越来越迫切。磁场测量数据是一种多维时间序列数据。时间序列数据与文本、声音、图像一样是一种无法使用统一结构进行表示的非结构化的数据。非结构化数据必须借助对应的解释软件才能打开并直观浏览，因此无法从数据本身直接获取其表达的物理属性，非常不易于理解。非结构化数据包含的信息量非常大，如果直接存储于数据库中，除大幅加大数据库的容量外，还会降低维护和应用的效率。数据的管理是数据时代的重要支撑技术，面向磁场测量数据的定制化管理是为了有效积累数据并为相关研究提供支持。磁场测量数据有其自身的特点，磁场数据包括标量数据和矢量数据，数据本身又具有时间和空间属性，同时，磁场数据的获取和使用也有其独特的上下文环境，对磁场测量数据使用普通数据库很难进行高效的研究和管理，在磁场数据多维度的属性的数据管理上往往顾此失彼。因此，有必要针对磁场数据的特点和使用环境开发一套相应的数据管理系统。

1.1.2 研究的目的和意义

磁场测量数据多维度多属性，查找、浏览、使用都很不方便。本课题将开发一套面向磁场数据的多维时间序列数据管理系统，针对磁场数据的特点设计合理的存储结构和高效的存储方案，使得磁场数据的存储更有针对性且更方便；开发面向网络的数据库管理系统，使得磁场测量数据可以随时通过网络进行实时存储、分类查询；结合应用背景开发相关的接口，使得磁场测量数据能够与用于其他数

据分析处理的系统进行有效的交互，提供便捷的数据管理服务；有效地管理变长度的磁场测量时间序列数据，使得复杂磁场数据的管理更灵活；并提供良好的数据操作界面及程序数据接口，从而提高使用者的工作效率，良好的数据接口可以拓展数据管理系统的功能，为其他程序提供合理且高效的数据服务。

1.2 时间序列数据库技术的研究现状

1.2.1 时间序列数据的特点

时间序列数据是具有时间特性的数据。在现实中主要在一些具有实时性的测量中会产生时间序列数据，例如地磁场数据测量、电力行业数据测量、化工行业数据测量等。这种时间序列数据具有一些典型特点：产生频率快、严重依赖于系统的实际采集时间、测量点多、信息量较大等。常规监测点较多，每次测量时需要每隔 0.1s 记录一次数据，因此每天产生的测量数据量是异常巨大的，单点单次测量数据一般都在 10MB 左右。如此看来，如果直接采用传统关系型数据库来进行时间序列数据的处理，那么面对时序数据的这种特点，在数据的存储、索引和查询等方面很难有良好的表现。

1.2.2 时间序列数据存储研究现状

目前的数据库系统通常是基于关系型数据库进行的开发，这种数据库系统能够对结构化的的关系型数据进行高效处理，但是对于非结构化以及半结构化的数据处理起来往往显得力不从心。时间序列数据也是一种非结构化数据，显然是不适合用关系型数据库进行直接存储的。

在对时间序列数据的存储研究上，最初还是以关系型数据库为基础，使用合适的方法对时间序列数据进行预处理，对序列进行合理建模，实现对时间序列数据这种非结构化数据的结构化处理。建模后的数据会变得有利于关系型数据库的存储和查询，从而能够充分发挥出关系型数据库在数据处理上的强大功能，进而对时间序列数据库进行高效处理。关于数据建模的方法，研究者们曾提出过例如：离散傅立叶变换、离散小波变换、奇异值分解法以及分段线性表示法等一系列用于针对时间序列数据特征提取从而进行特征建模的方法。但是这些方法往往都是针对一类数据的解决方案，并不具有普适性。况且有些算法的可行性仅存在于原理上，实际使用时十分复杂，会大大降低系统的数据处理效率。

对于这些时间序列数据的传统处理方法的缺点，时间序列数据库的出现很好的

弥补了不足。今年来在商业时间序列数据库的开发和应用方面，国外的时序数据库研究发展异常迅猛。从较早期的美国 Honeywell 公司开发的 PHD(Process History Database)系统，到 OSI 公司开发的 PI(Plant Information)时间序列数据库系统，再到当今使用比较广泛的 InfluxDB 和 OpenTSDB 时间序列数据库。时间序列数据库的发展日新月异。

1.3 本文的主要研究内容

本文以地磁场测量得到的时间序列数据的管理为背景，主要研究了磁场测量数据特点的分析方法和合理的磁场测量数据存储结构的设计方法；研究了基于 python 语言及其数据库系统，开发支持网络访问的数据库系统的方法；研究了数据库系统操作界面开发方法以及数据库系统交互接口的开发方法，用来配合数据可视化模块的对接。

课题研究的主要内容：针对磁场测量数据的特点，设计高效的数据存储结构；基于 python 语言及其数据库模块，开发数据库系统；开发数据库交互接口，配合可视化模块的对接。

第 2 章 多维时间序列数据的存储研究

2.1 引言

针对磁场测量数据的时间序列数据库系统的开发依赖于对多维时间序列数据这种特殊的非结构化数据的特点的研究以及存储结构的设计。这些关键技术的研究非常影响数据库系统最终的使用效率。而除了存储效率的需求，数据查询功能、数据处理功能、数据管理功能等一系列的系统需求也是重要的研究内容。

本章具体考虑了系统需求分析和数据库系统的关键技术的原理，给出了合理的针对性解决思路以及对关键技术的合理取舍和使用思路。

2.2 系统需求分析

本课题需要设计的时间序列数据库系统用于管理具有时间特性的数据。时间序列数据一般具有非结构化、数据类型多样化以及海量性的特点，所以对多种不同数据类型的统一化处理、合理的数据存储格式设计以及存储和查询的性能往往有比较高的要求。根据以上要求、数据库系统开发的常见需求和与数据可视化程序开发者达成的统一意见，时间序列数据库服务主要分为以下几类：多类型时序数据统一化处理、时序数据合理高效存储、时序数据高效查询、时序数据的条件管理以及合理数据传递接口提供。

2.3 数据库系统研究

数据库系统的关键技术的研究包括对多类型数据处理的研究、时间序列数据存储方式的研究以及时间序列数据传递效率的研究。

2.3.1 多类型数据处理研究

由于磁场测量数据是非结构化的时间序列数据，所以这类数据并不都具有相同的数据格式。通常情况下，使用不同的测量仪器，以不同的测量标准测量会产生不同属性类型和属性个数的时序数据。例如：

第一种类型数据：

2017 年 04 月 02 日 11:2:41.20 31.66N 119.79E 6.30 -3411.865234 -32156.372070 37384.033203 49350.613659 0.000000

这一列数据对应的属性依次是：日期、时间、纬度、经度、高度、X 轴分量、Y 轴分量、Z 轴分量、通道 1 总场、通道 2 总场。

第一种类型的数据文件的存储形式是所有属性的数据同时保存在一个文件名为无特殊意义的文件中，文件扩展名为“.txt”。

第二种类型数据：

57334759 00000000 +05100.33 -25134.40 +50324.88 56933.0306 32629.9834 56805.6218 37528.0245 +0.0000 -4.4476 +0.0000 +2.1237 56933.0306 32629.9834 56805.6218 37528.0244 53432.4804 32638.7918 56805.6218 37521.4997 +24303.0472 -12024.1147 +19277.5973 +23477.3907 -12024.1147 +19277.5973 15:55:34.759

这一列数据对应的属性依次是：扫描数值、事件输入标签、x 轴分量、y 轴分量、z 轴分量、4 个原始总场数据、4 个四阶差分数据、4 个原始总场未补偿数据、4 个原始总场补偿后数据、3 个梯度未补偿数据、3 个梯度补偿后数据。

第二种类型数据的数据文件存储形式和第一种类型数据类似，也是所有属性的数据同时保存在一个文件名仅为文件编号的文件中，文件没有扩展名。

第三种类型数据：

这种类型的数据包含了三种补偿系数属性：coeff_bpf、coeff_bpf_diff、coeff_diff，滤波前“_raw”属性：he 总场、total 总场、x 轴分量、y 轴分量、z 轴分量，滤波后“_cal”属性：he 总场、total 总场、res_cal_bd 补偿结果、res_cal_bpf 补偿结果、res_cal_diff 补偿结果、x 轴分量、y 轴分量、z 轴分量。

第三种类型数据文件的存储形式是每种属性的数据保存在一个单独的文件中，所有的属性数据文件保存在一个表明测量起始时间的压缩包中，文件扩展名为“.rar”。

由此可见，对于目前的三种数据类型的时间序列数据形式，使用统一的格式进行存储在关系型数据库中是不现实的。对于每种类型的数据，可以考虑对文件内容进行一定的预处理。利用 python 强大的字符处理功能进行处理。首先不分数据类型，整体上考虑将数据属性分为时间戳数据和其他数据两类。每条数据必定包含这两部分，时间戳数据的格式处理成固定的格式，而其他数据不管是类型、个数、长度都不需要特别的要求，因为在时间序列数据中时间属性是必不可少的。其次将不同存储形式的数据统一成数据流的形式，属性与属性之间是相关关系，存放在不同属性文件中的数据必须组合成新的数据，每条数据流由时间和其他属性数据组成。

多类型数据的处理必须具体考虑每一种数据与其他类型数据的具体不同之处，然后设计出统一而不固定的数据格式，这样能够一定程度上限制过于复杂的

数据结构在数据存储时效率低下，这种对不同类型数据的预处理从总体上提高了数据存储的速度，也提高了数据意义的可读性。

2.3.2 时序数据存储研究

对于时间序列数据的存储来说，从最基本的文本存储到最新的时间序列数据库都可以存储，但是不同的存储形式对数据的使用和管理有不同影响。时间序列数据本就是有着自身特点的非结构型数据，如何选择最合适的数据存储结构和存储载体会大大影响数据库系统的数据管理效率。关于时序数据的存储研究也是一直以来研究者们都在探索的问题。

(1) 平面文件

平面文件是最简单的数据存储形式。使用平面文件存储时，可以通过列存储格式 Parquet 来扩展原本非常简单的设计，如图 2-1 就是使用 Parquet 格式存储时间序列数据可能的两种 schema.左边的使用固定的类型名称将问题域确定了，所以存储的时间序列个数是固定的；而右边的 schema 更加灵活，可以增加新的时间序列。当需要分析的时间序列数量较小，并且所感兴趣的时间范围相对与单个文件所存储数据的时间跨度很大时，这种时间序列数据存储方式是非常有用的。

```
message simpleSeries {
  repeated group sample {
    required float t;
    optional float tempIn;
    optional float pressureIn;
    optional float tempOut;
    optional float pressureOut;
  }
}
```

```
message fancySeries {
  repeated group block {
    repeated group tags {
      optional string name;
      optional string value;
    }
    repeated float time;
    repeated float value;
  }
}
```

图 2-1 两种 Parquet 存储格式

系统最初使用平面文件来实现是一种非常普遍的情况，而且不久之后这种简单的实现不再适应快速增长的数据的情况也是很普遍的。其基本问题是单一文件的时间序列数量增加了，任何特定的查询中，真正有用的数据占所读取数据的比例就下降了，因为多数读取到的数据其实是属于其他时间序列的。为了避免大量时间序列数据造成的文件数量大幅增长、文件数量过多导致的严重稳定性问题和过多小文件造成的搜索时间增加的问题，应该使用真正的数据库来存储这些数据。所以得出的结论是平面文件对时间序列数据的存储来说虽然形式简单，但却是受

限的工具，不适合快速增长的数据。

(2) 关系型数据库

使用关系型数据库存储时间序列数据时，由于时间序列数据是非关系型数据，所以不能直接对数据进行存储，而需要使用星型模式设计将时间序列数据结构化。在这种数据库存储设计中，核心数据存放在事实表中，而序列的细节存放在维表中。如图 2-2 所示。

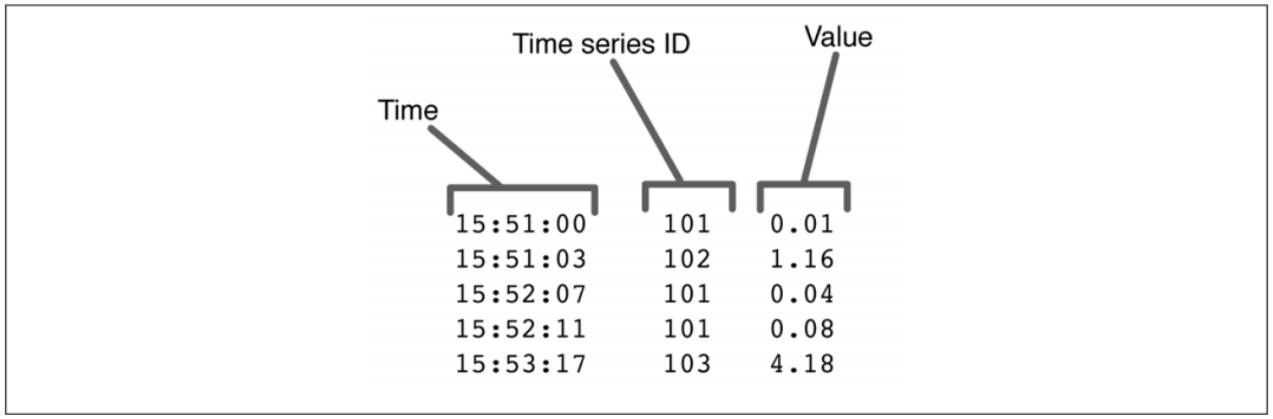


图 2-2 星型模式设计的数据表模式

星型模式中一个表存储主要数据，并会引用其他表。这个设计的核心假定是维表要相对小巧，而且不经常变动。从这种模式的设计可以很容易的看出，简单的将这些时间序列数据存储是没有问题的，但是检索和处理效率却非常低。数据规模越大，基于关系数据库的应用程序就越不适合处理这样的时间序列数据。所以考虑对时间序列数据进行预处理建模然后存储在关系数据库中的方案是不适合的，因为磁场测量数据是有着较大存储规模需求的时间序列数据。所以得出结论关系型数据库有着不够好的扩展性，常见的星型模式存储不适合处理时间序列数据。

(3) 非关系型 NoSQL 数据库

非关系型 NoSQL 数据库是存储时间序列数据的首选方案，因为可扩展性好、高效、能快速相应基于时间段的查询。

最终最适合存储时间序列数据的方案是使用 NoSQL 数据库。目前有三种 NoSQL 数据库的存储设计思路：使用宽表的 NoSQL 数据库、混合模式设计的 NoSQL 数据库以及 blob 直写设计。但是这些设计都基于最基本的设计：使用包含时间序列 ID 的唯一 row key，列是不同时间偏移的数值，并且可存储多于一个时间序列。

在使用宽表的 NoSQL 数据库中，用到了宽表这样的存储结构如图 2-3 所示，尽量减少行的数量，而列的数量几乎是不受限制的，这样每行可以存放尽量多的数值。扫描数据的最大数据部分取决于需要扫描的行的数量，所以这种存储结构可以大幅减小检索开销，提升检索速度。使用宽表来减少时间序列数据行数量的技术和开源时间序列数据库 OpenTSDB 中使用的默认表结构很相似。

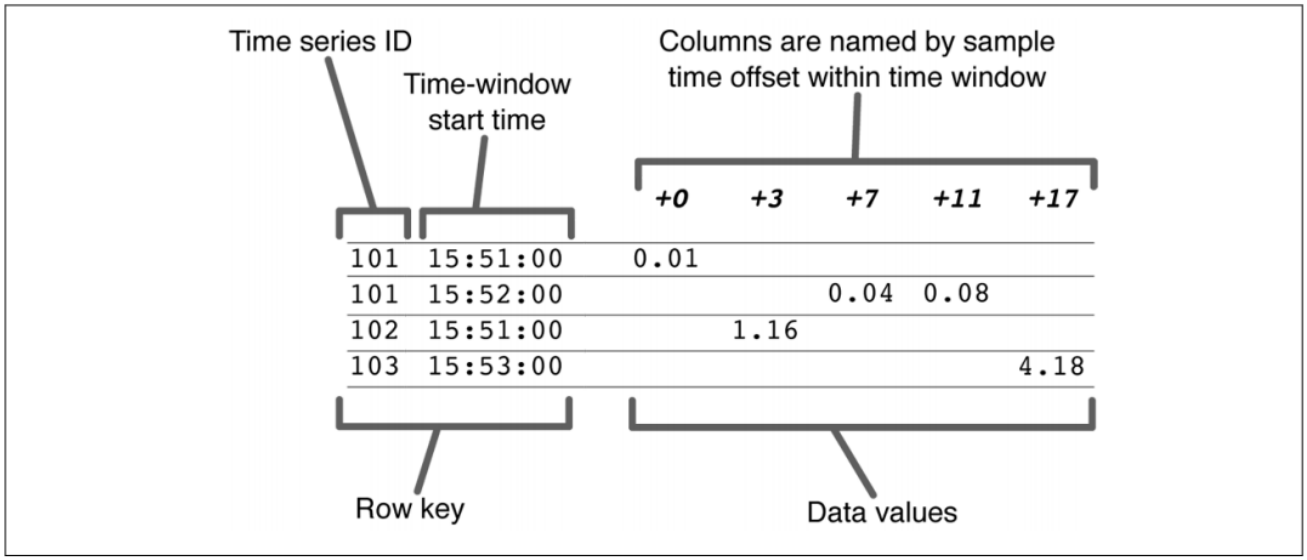


图 2-3 使用宽表的存储结构

在使用混合模式设计的 NoSQL 数据库中，这种混合模式设计其实是对宽表存储模式的改进。如图 2-4 所示，通过将一行中的所有数据压缩成一个单一的被称做 blob 的数据结构。Blob 可以高度压缩，这样需要从磁盘读取的数据量更少。而如果使用 HBase 来存储时间序列数据，每行只有一列的情况会减少每列数据在 HBase 所使用的磁盘文件格式上的开销，进一步提高了性能。

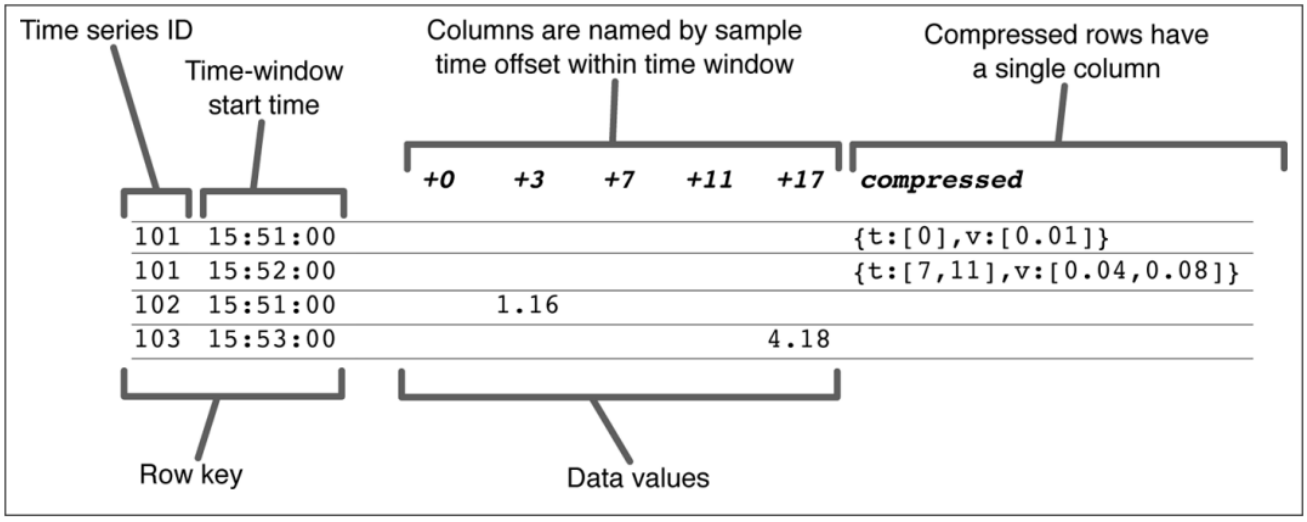


图 2-4 使用混合模式的存储结构

混合式时间序列数据库的数据流如图 2-5 所示。数据从数据源到达 catcher，然后被插入到 NoSQL 数据库中。之后 blob maker 在后台定时将数据压缩成 blob 格式。数据又 renderer 检索和格式化。

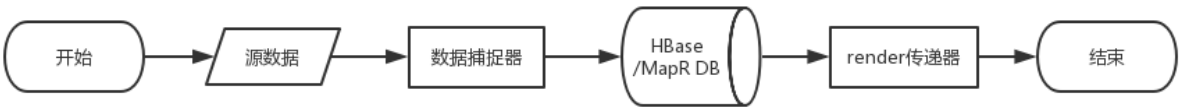


图 2-5 混合式时间序列数据库的数据流程

更进一步提升数据存储效率的方式是 blob 直写，简单来说相对于混合模式设计的时间序列数据库把 blob maker 的步骤放在了 data catcher 和 HBase/MapR DB 之间。在混合模式设计中，数据在存入数据库中之后再被 blob maker 读出然后压缩为 blob 格式，然而在 blob 直写设计中，数据在存入数据库之前就被 blob maker 压缩为了 blob 格式再存入数据库。Blob 直写设计减少了原始数据写入数据库和从数据库中读取原始数据的步骤，完整的数据流只写入到内存中，而不是写入到数据库中，数据的总体写入速度大幅提升。设计流程图如图 2-6 所示。

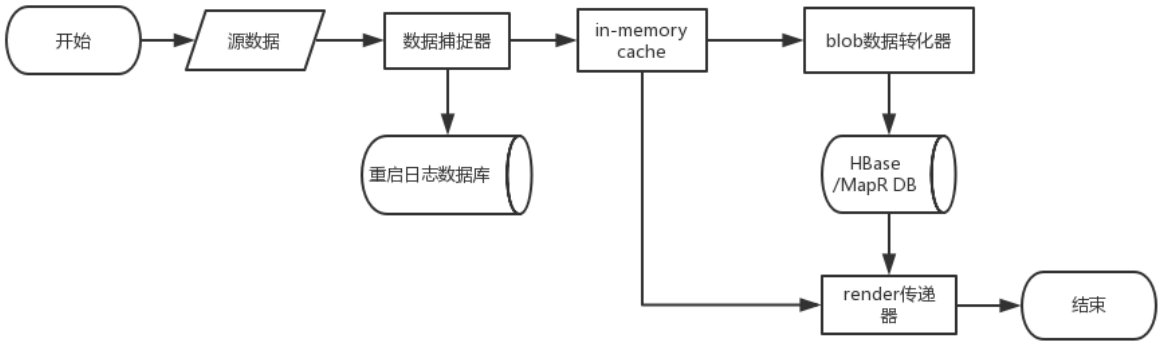


图 2-6 包含 blob 直写设计的混合模式数据流程

所以总体上来看这种数据存储设计才是最适合于时间序列数据存储的设计，因为它有着适合时间序列数据的存储结构、很高的数据存储效率和数据检索效率、数据压缩存储也解决了大规模数据存储占用空间大的问题。这样的 NoSQL 数据库比关系数据库更适合时间序列数据，因为关系数据库主要解决的问题不是提高插入和检索数据的效率，如果使用关系数据库的 blob 格式存储数据，就意味着要放弃大多数其他好处。

基于以上的研究工作，综合考虑了关系数据库和 NoSQL 数据库的优缺点，选择 NoSQL 数据库。这些宽表、混合表与 NoSQL 数据库的结合设计，产生了专门的成熟的时间序列数据库，例如：OpenTSDB、Influxdb。考虑到项目的完成效果和完成效率，选择了 Influxdb 时间序列数据库作为最终使用的存储数据库，Influxdb 是以上的研究工作中说到的高效的时间序列数据存储方案+NoSQL 数据的具体实现，是一种 key-value 型数据库，兼具了时间序列数据库高效的数据存储和检索以及类似 SQL 语句的控制语句。

2.4 本章小结

本章主要针对该数据库系统的需求进行了分析，对系统所用到的时间序列数据存储有关的多类型数据处理、时序数据存储结构设计与载体选择等关键技术进行了介绍，并在研究中提出了自己的看法，选择了最适合的方法进行时序数据库系统的开发。

第 3 章 系统结构的设计

3.1 引言

本系统设计的目的主要是针对时间序列数据的管理，对时间序列数据的存储是数据管理的前提，而系统整体框架设计又是系统各个模块设计的前提。在整体框架的设计上需要充分考虑用户需求，给出多种使用情景。而在对时间序列的存储上，使用了第二章关于时间序列数据存储技术的结论，针对不同类型数据进行相应的预处理。合理的整体框架设计和时间序列数据存储的设计是系统设计的基础。

本章分析设计了系统所需要的功能框架，给出了合理的整体框架设计，并根据各个功能模块的需求，给出了模块化实现需要实现的关键技术和具体的实现方法。

3.2 系统整体框架设计

从整体上看，任何普通数据库系统都具备的基本功能有数据的存储、数据的查询、数据的删除、数据的修改。但是考虑到时间序列数据的特殊性，时间序列数据的存储是大量连续的数据流的存储，在大量连续的时间流数据中只修改某个数据的操作是多余且无意义的。况且在实际的数据测量中，如果测量时出现错误，那么得到的是整个临时记录文件的测量结果，整体数据删除是比数据修改更有意义和效率的处理方法，所以该系统的数据修改模块被并入了数据删除模块。系统整体上看来目前需要数据的存储、数据的查询以及数据的删除模块。

根据系统的使用需求，该数据库系统需要对多种不同类型的时间序列数据进行统一化的处理，所以系统内部对各种类型数据的针对化处理模块也是必不可少的。而该系统的一个主要用途是为其他时间序列数据使用程序提供数据支持，所以系统需要提供一些良好的、有通用性的程序数据接口，故而系统的整体框架中还需要数据库系统接口模块。除了以上的各种功能模块，需求中指出需要提供良好的操作系统界面，所以数据库操作界面的设计也是必需的设计。

综合考虑这些功能模块可以得出如图 3-1 所示的系统整体模块架构图

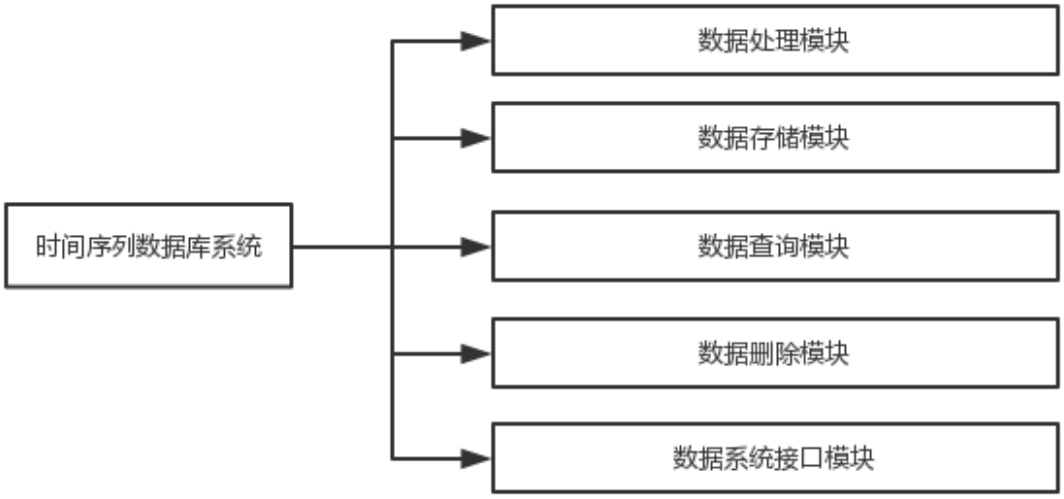


图 3-1 系统整体模块结构图

而在系统整体功能的流程设计上考虑了各个系统功能模块之间的关系，从用户请求开始一步步设计了数据流在系统内的传递过程。系统数据流程图如图 3-2 所示。从图中可以看出系统的操作界面的设计上是功能选择的模式，存在数据存储操作、数据查询操作和数据删除操作的选择。每种选择的功能对应其功能模块。在数据接口模块的设计上，必须考虑其他程序在使用上的方便性和数据传递的效率。所以系统单独提供可以直接调用的接口函数，在提供方便使用性的同时，请求的数据都是通过系统的查询模块得到的，保证了数据传递的效率。

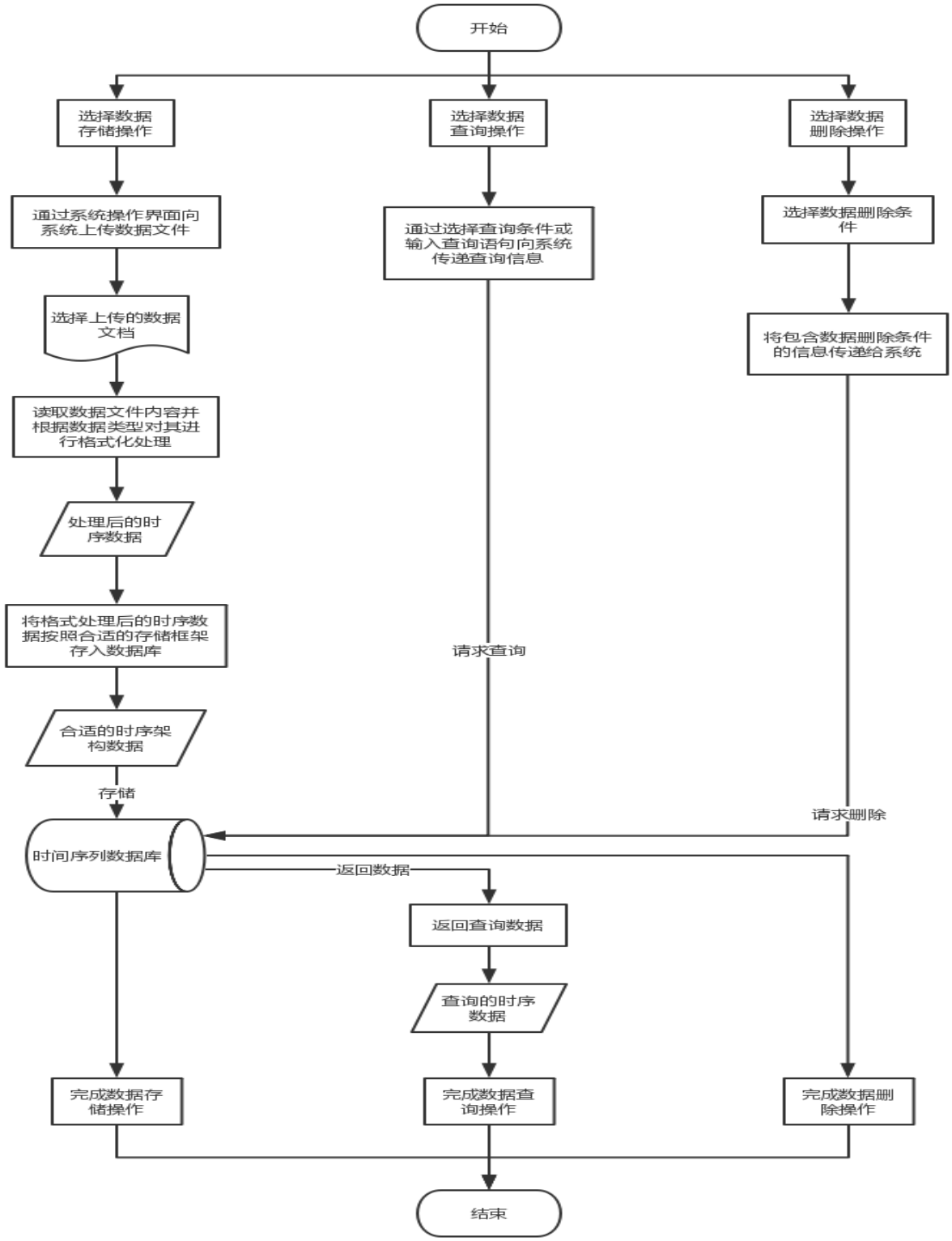


图 3-2 系统整体功能流程图

3.3 系统模块设计

系统由各个独立的功能模块组成，主要有数据处理模块、数据存储模块、数据查询模块、数据删除模块、数据库系统接口模块和系统操作界面模块。其中数据删除模块和系统操作界面模块的设计没有复杂的功能且和时间序列数据的相关性不大，所以主要介绍数据处理模块、数据存储模块、数据查询模块和数据删除模块的模块细节设计。

3.3.1 数据处理模块

在数据处理模块中需要针对多种类型的数据进行相应处理，具体来说数据类型有三种，这三种类型的数据在第二章中有所介绍。

第一种类型数据：

2017 年 04 月 02 日 11:2:41.20 31.66N 119.79E 6.30 -3411.865234 -32156.372070 37384.033203 49350.613659 0.000000

这一列数据对应的属性依次是：日期、时间、纬度、经度、高度、X 轴分量、Y 轴分量、Z 轴分量、通道 1 总场、通道 2 总场。

第一种类型的数据文件的存储形式是所有属性的数据同时保存在一个文件名为无特殊意义的文件中，文件扩展名为“.txt”。

针对这种类型数据的处理，首先需要分析数据有哪些属性，有多少个属性。由于第一种类型数据和第二种类型数据都是数据存储在一个文件中的类型，所以在读取文件内容后直观上区别就是属性个数有明显的区别，第一种数据的属性个数是固定的这 10 个，如果判断读取的数据属性个数小于或等于 10 个，就可以将其确定为第一种类型数据。第一种类型数据需要进行的最大格式处理就是时间格式的统一，规定统一的时间存储格式为“%Y-%m-%d %H:%M:%S.%f”，所以首先需要将日期中的中文转化为对应的“-”字符。这种时序数据在测量时连接了 GPS，所以每条数据流都给出了对应测量时间，本不需要考虑具体时间数值的改变，但是由于 GPS 的时间每隔 0.2S 才刷新一次，所以得到的测量数据是相邻的两个数据时间相同，而每两个数据时间间隔为 0.2S。所以需要对其进行修改，每隔一个数据其时间就需要自动加 0.1S。

第二种类型数据：

57334759 00000000 +05100.33 -25134.40 +50324.88 56933.0306 32629.9834 56805.6218 37528.0245 +0.0000 -4.4476 +0.0000 +2.1237 56933.0306 32629.9834 56805.6218 37528.0244 53432.4804 32638.7918 56805.6218 37521.4997 +24303.0472

-12024.1147 +19277.5973 +23477.3907 -12024.1147 +19277.5973 15:55:34.759

这一列数据对应的属性依次是：扫描数值、事件输入标签、x 轴分量、y 轴分量、z 轴分量、4 个原始总场数据、4 个四阶差分数据、4 个原始总场未补偿数据、4 个原始总场补偿后数据、3 个梯度未补偿数据、3 个梯度补偿后数据。

第二种类型数据的数据文件存储形式和第一中类型数据类似，也是所有属性的数据同时保存在一个文件名仅为文件编号的文件中，文件没有扩展名。

第二种类型数据的处理不同于第一类数据，它的数据有两个特点：属性比较多和时间属性数据缺少日期。而且这类数据有一个最特殊的地方，那就是数据属性的个数是不定的，其原始总场个数可以为 2、3、4 个，所以对应的一系列属性：四阶差分数据、原始总场补偿数据等的个数也会发生相应变化。对此，设计分类处理的模式。由于属性个数都与原始总场的个数有关，所以可以计算出第二种类型数据的属性个数可能为 28 个（4 个原始总场）、22 个（3 个原始总场）、16（2 个原始总场）。分类之后可以对同中类型的不同属性集中存储。由于日期数据的缺失，最合理的解决方案是在数据存储时手动输入日期数据作为时间数据存储。

第三种类型数据：

这种类型的数据包含了三种补偿系数属性：coeff_bpf、coeff_bpf_diff、coeff_diff，滤波前“_raw”属性：he 总场、total 总场、x 轴分量、y 轴分量、z 轴分量，滤波后“_cal”属性：he 总场、total 总场、res_cal_bd 补偿结果、res_cal_bpf 补偿结果、res_cal_diff 补偿结果、x 轴分量、y 轴分量、z 轴分量。

第三种类型数据文件的存储形式是每种属性的数据保存在一个单独的文件中，所有的属性数据文件保存在一个表明测量起始时间的压缩包中，文件扩展名为“.rar”。

第三种类型数据的处理是最复杂的一种，因为其压缩包的数据文件存储方式和其他类型的数据文件相差较大。首先需要对文件进行解压，然后依次读取每个属性数据文件的内容并将数据由纵向组合转化为横向组合，所有属性的对应数据组合成连续的数据流。文件名中只给了数据测量的起始时间，由数据测量的时间间隔是 0.1S 可以自行通过计算为数据添加每条数据的具体测量时间。

3.3.2 数据存储模块

在数据存储模块中也需要具体针对每种类型的数据进行存储结构的设计，但是总体上来说，根据所选用的时间序列数据库 InfluxDB 本身的存储结构，所有的数据属性可以大体上分为三部分：time 属性、tag 属性和 field 属性。time 属性数据

设为存储的每条数据流的测量时间，如果不进行赋值则默认成为数据存入数据库的时间。具体设计为第一种类型数据的 time 属性存储的是每条数据中的时间，第二种类型数据的 time 属性存储的是手动输入的日期与数据中自带时间的组合数据，第三种类型数据的 time 属性存储的是根据数据测量起始时间和测量时间间隔计算出的每条数据的时间;tag 属性存储的是系统数据中那些需要添加索引的属性，这些属性的数据在是数据中查询频率较高的属性，将这些属性添加为索引后，查询的速度会显著高于一般属性。具体来说第一种类型数据中的 x 轴分量、y 轴分量、z 轴分量、通道 1 总场、通道 2 总场和文件名应该添加索引，其他的数据查询的频率相对较低。第二种类型数据中的扫描数值、事件输入标签、梯度未补偿数据、梯度补偿数据和文件名应该添加索引。第三种类型数据中的数据存储相比于第一种和第二种类型比较特殊，对于滤波前 raw 数据 he 总场、total 总场和文件名需要添加索引，对于滤波后数据 he 总场、total 总场、bd 补偿、bpf 补偿、diff 补偿和文件名需要添加索引。

数据的存储除了需要考虑数据属性的分配外，还要考虑数据库和数据表的分配。数据类型的分类上，由于第一种类型数据、第二种类型数据形式相似且与第三种类型数据有很大区别，设计第一种类型数据和第二种类型数据存储在统一个数据库的不同数据表中，而第三种类型数据存储在另一个数据库中的多个数据表中如图 3-3 所示。

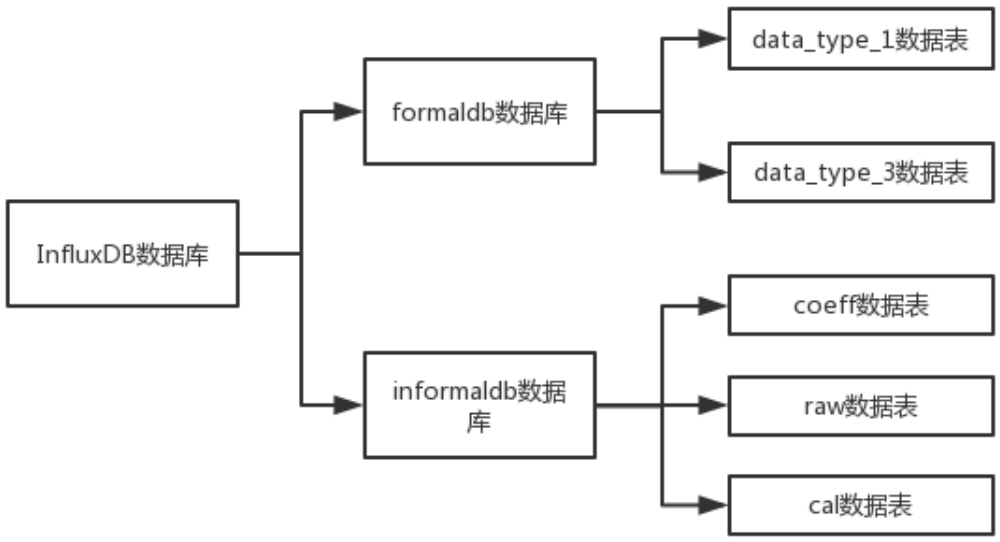


图 3-3 数据库中数据存储结构

InfluxDB 数据库是总数据库，formaldb 数据库用于存储类型一和类型二数据，分别存储在 data_type_1 数据表和 data_type_2 数据表中。informaldb 数据库用于存储类型三数据，分别用数据表 coeff、raw、cal 来存储类型三数据的参数、滤波前数据和滤波后数据。

3.3.3 数据查询模块

数据查询模块负责数据查询功能的实现，根据 InfluxDB 数据库的特点，数据库可以输入类 sql 进行查询，但是数据查询的方式有很多种：组合查询、条件查询等。所以在数据库系统的数据查询模块可以设计多种查询形式，并且在考虑查询效率的同时考虑系统使用的便捷性。系统的查询形式包含两种：选择数据库并直接输入查询语句查询和逐步选择条件查询。第一种查询形式比较直接，这种查询方式的设计也是利用了 InfluxDB 支持类 sql 语句的性质，直接向系统输入选择的数据库和查询语句就能得到数据库系统返回的查询结果。第二种查询形式是更适合一般情况下使用的查询形式，因为用户在初次接触到数据库系统时不知道有哪些数据库，每个数据库中包含哪些表，每个表中具体包含哪些数据属性，这时就需要用到第二种查询形式。逐步条件查询模式首先会提供所有的数据库选项供用户选择，选择一个数据库后系统会提供这个数据库内的所有表选项，选择一个表后系统会提供这个表内的所有属性选项，选择多个需要查询的属性，系统会将查询结果返回。逐步条件查询模式的流程图如图 3-4 所示。

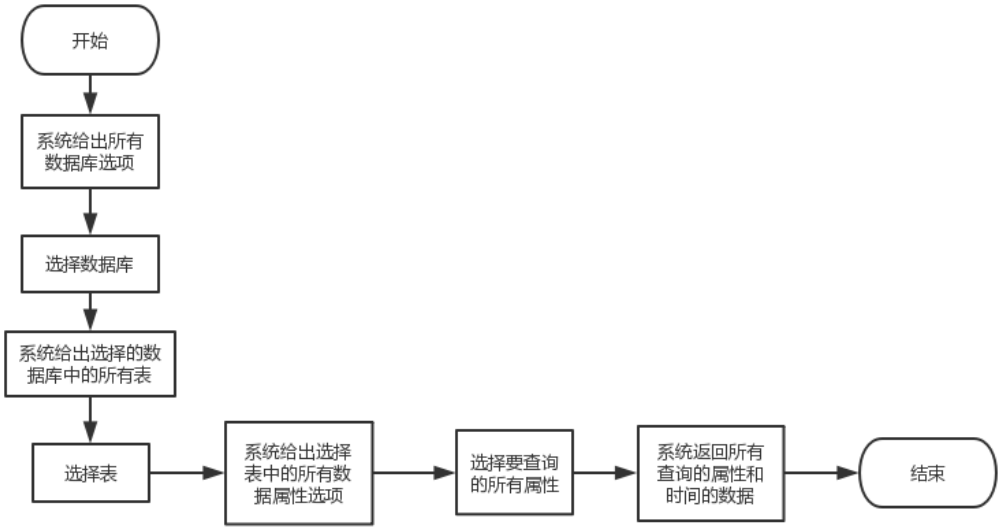


图 3-4 逐步条件查询模式流程图

3.3.4 数据库系统接口模块

数据库系统接口模块是为了方便为其他时间序列使用程序提供数据服务而设计。为了同时满足数据传递的高效性和接口使用的便捷性，选择使用接口函数的形式提供数据服务。系统总共设计了多个接口函数，按功能分类包括：获取数据库中存储所有文件名的函数 getalldatafilename(...)、根据文件名获取所有数据的函数 getdatabyname(...)、将返回的经过滤波处理的数据存入数据库的函数 writedata(...)、根据文件名获取文件属性的函数 getattrbyfilename(...)。这些函数的设计实现了数据库系统与时序数据使用函数之间的数据连接。接口函数可以直接被其他程序调用，确保了接口数据连接的便捷性。通过数据库系统获取需要处理的数据而不是直接自身对数据进行处理和选择，确保了数据处理和查询的高效性。系统数据库接口模块功能设计如图 3-5 所示。

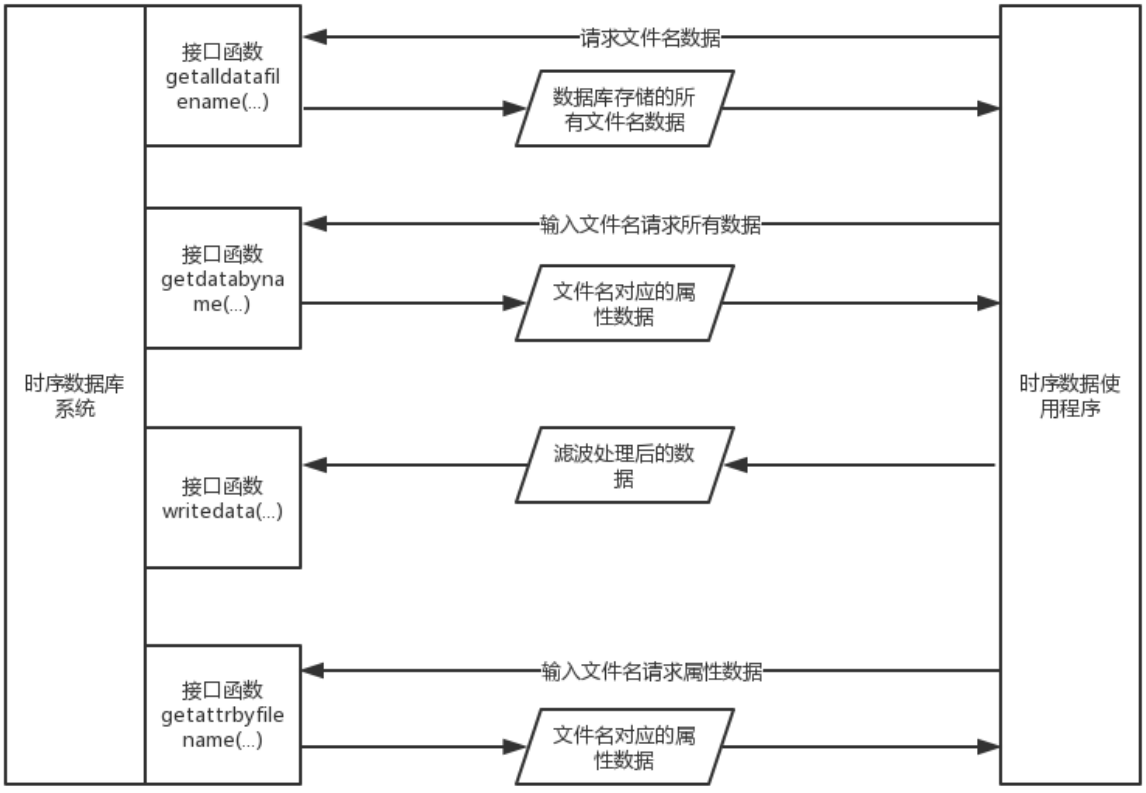


图 3-5 数据库系统接口模块功能设计图

3.4 本章小结

本章主要讨论了系统的整体设计以及主要功能模块的设计。系统整体设计考虑了使用的便捷性以及数据传输的效率。主要设计了数据处理模块、数据存储模块、数据查询模块、系统接口模块等。本章的设计为后面系统各个功能的实现提供了思路。

第 4 章 系统功能的实现

4.1 引言

上一章讨论了系统整体架构以及各个主要功能模块的设计思路，本章将重点介绍各个功能模块的具体实现方法。

4.2 系统模块实现

系统的主要功能模块基于是基于时间序列数据库 InfluxDB 的使用而实现的。系统在 Ubuntu 14.04 操作系统中使用 Python 2.7，在 Flask 网页架构和时间序列数据库 InfluxDB 的支持下实现了该数据库系统。

4.2.1 数据处理模块

该模块的设计思路已经在第三章有了详细的介绍，而该模块的具体实现方法是基于 Python 的数据操作功能以及时间操作函数库来完成的。具体的实现需要根据具体的数据类别分类处理。

对于第一类数据：

2017 年 04 月 02 日 11:2:41.20 31.66N 119.79E 6.30 -3411.865234 -32156.372070 37384.033203 49350.613659 0.000000

主要需要处理的是对其时间数据的格式统一以及正确测量时间的更改。在使用 Python 文件读取功能读取完文件中所有数据后，首先需要根据一小串数据获取文件内容的编码格式，由于测量仪器的不同，第一类测量数据中包含中文字符，需要进行替换。根据文件的编码格式将所有数据编码为能够支持所有字符的 Uncode 格式，然后利用 python 的字符串处理方法将每条数据的对应中文字符“年”、“月”、“日”等统一替换为“-”表示。为了减少数据读取的次数，在数据日期字符替换的同时，修改由于 GPS 时间刷新频率造成的测量时间计量错误，利用 Python 的 datetime 函数库每间隔一个数据就在其记录的时间数值上加 0.1S。通过这两步处理，将处理的结果用一个列表存储起来就得到了格式化的包含正确测量时间的时序数据。这个数据列表可以直接传递给第一类数据存储函数将时序数据存入数据库。

对与第二类数据：

57334759 00000000 +05100.33 -25134.40 +50324.88 56933.0306 32629.9834 56805.6218 37528.0245 +0.0000 -4.4476 +0.0000 +2.1237 56933.0306 32629.9834 56805.6218 37528.0244 53432.4804 32638.7918 56805.6218 37521.4997 +24303.0472 -12024.1147 +19277.5973 +23477.3907 -12024.1147 +19277.5973 15:55:34.759

主要需要处理的是数据的时间问题。这类数据的其他属性有这较为规范的格式，但是时间数据缺少了日期数据。在该模块的设计中，需要在用户上传此类数据文件进行存储时手动输入数据测量日期。该处理的具体实现是利用不同的条件判断是否进行过日期数据的输入，未输入过日期数据的返回日期数据输入网页，输入过日期数据的将日期数据和文件内数据一同传递给第二类数据存储函数进行数据存储。

对与第三类数据，由于其文件格式的特殊性，根据第三章介绍的设计思路，首先通过调用 os 函数库，通过 Python 语句执行系统命令，将压缩文件解压，然后依次读取所有文件并根据文件名中标记的文件类型分为三类：coeff、raw、cal。使用 Python 列表功能将每一类数据的属性文件同一测量时间的对应数据组合成测量数据流，并再次使用 Python 的 datetime 函数库根据文件夹名作为测量起始时间和数据测量间隔为 0.1S 的原则为每一条测量数据流添加测量时间数据。完成这些处理的三类数据全部传递给第三类数据存储函数进行存储。

4.2.2 数据存储模块

数据存储模块的存储主要是基于 Python 与 InfluxDB 的连接，通过 Python 对数据库进行直接操作。在安装了 Python-InfluxDB 连接模块 influxdb 后，在 Python 程序中调用 influxdb 函数库中的 InfluxDBClient 类。然后用 Python 语句创建与数据库的连接。通过调用 InfluxDBClient 中的子函数，对输入的数据进行存储操作。数据存储的格式是 json 格式，在 Python 中表现为字典的形式。具体的数据存储操作需要分类别介绍。

对于每一类数据，其 json 格式中都包含了四个主要部分：measurement（代表数据存入的数据表名，如果表不存在会自动创建表）、time（代表存储数据的时间，可以给这个时间数据赋值，如果不赋值则默认保存为数据存入系统的时间）、tags（包含要存储的数据属性中需要建立索引的部分属性数据，这些数据在检索时速度更快且条件查询时选择的只能以 tags 中的属性作为选择条件）、fields（包含除了要存储的数据属性中需要建立索引的其他属性数据，这些数据在查询时相对速度较慢，但是在任何数据查询时必须包含至少一个 fields 属性的查询）。

在第一类数据中 measurement 被赋予了“data_type_1”作为值，表明存储第一类

数据的表名。time 被赋予了传入存储函数的经过了数据处理的每条数据中记录的测量时间。tags 中包含了 num（用来记录该条 json 数据的数据类型）、filename（用来记录该条数据来自的文件名），x 轴分量、y 轴分量、z 轴分量、通道 1 总场和通道 2 总场都是第一类数据中可能作为数据查询条件以及查询频率较高的属性。fields 中包含了经度、维度、高度这些查询频率较低的数据以及数据的备注属性 remark。

在第二类数据中 measurement 被赋予了“data_type_2”作为值，表明存储第一类数据的表名。time 被赋予了传入存储函数的经过数据处理的每条数据中的时间数据以及文件上传时手动输入的数据测量日期的组合数据。tags 中包含了 num（用来记录该条 json 数据的数据类型）、filename（用来记录该条数据来自的文件名）、扫描数值、事件输入标签，1-3 个未补偿梯度数据、1-3 个补偿后梯度数据都是第二类数据中可能作为查询条件。fields 中包含了其他的 x 轴分量、y 轴分量、z 轴分量、2-4 个原始总场、2-4 个四阶差分数据、2-4 个未补偿总场数据、2-4 个补偿后总场数据以及数据的备注属性 remark。

在第三类数据的存储中，使用三个 measurement 来存储经过数据处理后的三类数据：参数数据 coeff、滤波处理前数据 raw、滤波处理后数据 cal。这三个表的 measurement 分别为 coeff、raw 和 cal，而 time 则都是相同的经过数据处理的时间数据。在 coeff 表中的 tags 包含了 num（用来记录该条 json 数据的数据类型）、filename（用来记录该条数据来自的文件名）。coeff 中存储的参数查询的频率很低，所以三个参数数据被放在 fields 部分存储。而 raw 表的 tags 部分中包含了 num、filename、he 总场数据和 total 总场数据，这些属性都是可能的查询条件属性，fields 部分包含了其他的 x 轴数据、y 轴数据、z 轴数据、备注属性 remark 和滤波处理标记 process。cal 表的 tags 部分包含了 num、filename、he 总场、total 总场、bd 补偿数据、bpf 补偿数据、diff 补偿数据，fields 部分包含了 x 轴数据、y 轴数据、z 轴数据、备注属性 remark 和滤波处理标记 process。

最后在根据设计好的 json 格式将数据存入数据库中后，为了方便数据查询需要另外创建一个数据库 message 用表 filename_filetype 用来存储每次存储数据的信息，包括数据类型、文件名和备注信息。

4.2.3 数据查询模块

数据库查询模块的实现也是基于 Python 的数据库操作，调用 InfluxDBClient 类中的 query 函数，在参数中输入类 sql 查询语句，得到查询结果。查询结果一般

为 json 格式的数据，通过使用 list 函数进行处理，将其转化为列表数据从而使用或查看。

在两种查询模式的实现中，直接输入查询语句查询是通过在选择查询方式后简单地向用户提供数据库和查询语句的输入界面，通过定义一个叫 QueryForm 的类来实现，这种方式适合对数据库系统比较熟悉的用户的直接操作。而另一种查询模式的实现是通过不断根据用户的选择组织合适的查询语句一步步返回查询结果给用户继续选择，直到最后得到选择属性的查询结果。实现具体方式是将页面接收的参数作为查询条件，不断向定义的查询函数传递并更新查询语句以及页面上的可选项，从而实现逐步条件查询。

4.2.4 数据库接口模块

数据库接口模块的实现主要基于 Python 方便的函数调用功能。根据数据库接口模块的设计，四个功能接口函数：获取数据库中存储所有文件名的函数 getalldatafilename(...)、根据文件名获取所有数据的函数 getdatabyname(...)、将返回的经过滤波处理的数据存入数据库的函数 writedata(...)、根据文件名获取文件属性的函数 getattrbyfilename(...)，这些函数的功能实现都是通过合理的利用了数据查询模块和数据存储模块的功能。

获取数据库中存储所有文件名的函数 getalldatafilename()是在被其他时序数据使用程序调用后，主动查询数据库系统中的 message 数据库，得到数据库中存储的所有文件名。

根据文件名获取所有数据的函数 getdatabyname(filename, datatype, chs)是在被其他程序调用后，根据输入的参数文件名、数据类型、以及如果是第三类数据则是否经过滤波处理的标记值，生成数据条件查询语句对数据进行查询得到包含该文件名属性的所有数据。首先根据 datatype 选择连接不同的数据库和数据库中不同的表，然后以 filename 作为查询条件组织查询语句，最后根据固定的函数返回格式对数据进行处理。

将返回的经过滤波处理的数据存入数据库的函数 writedata(filename, datatype, data)是纯粹接收其他程序传入的数据，并将其存入数据库系统的函数。函数接收参数 filename 用于给数据添加新的文件名属性，标记其为经过滤波处理后的文件。函数接收 datatype 函数用于选择连接的数据库和数据库的表来进行在适当位置的存储。函数接收的 data 是程序传递的固定格式的需要存储的数据。在函数根据 datatype 判断出数据类型之后，函数会调用数据存储模块的三种数据存储函数进行

数据存储操作。

4.3 本章小结

本章主要通过关键技术的具体运用、系统运行环境的具体介绍以及重要函数的参数功能介绍对数据库系统的各个主要功能模块的实现方法和实现细节进行了完整的展示。

第 5 章 系统测试验证

5.1 引言

前面的章节已经详细介绍了系统的整体及各个功能模块设计和系统的具体实现方法，本章将具体着眼于几个有代表性的测试案例对系统进行本身的功能测试和与时序数据可视化系统的联合测试。

5.2 系统各模块功能测试

本次测试同时针对本系统自身功能的测试和与时序数据可视化系统的联合测试，所以选择的测试案例要具有代表性。测试选取三类数据每类数据一个作为测试案例。第一类测试数据为数据文件“data_cal.txt”这是实验室中的采集器采集的测量数据，数据保存在这种扩展名为“.txt”的文本文档中。第二类测试数据为数据文件“d081555”这是加拿大的一个采集器产生的测量数据，数据保存在这种无扩展名的文本文件中。第三类测试数据是数据文件压缩包“05-20-201611-54-35.rar”，这是实验室的采集器以前的旧的测量数据，数据记录形式和其他两类数据都不同，每种属性的数据单独保存在一个文本文档中，所有的属性文档保存在这个压缩包里。这三类测量数据满足了之前在数据处理模块提到的需要处理的不同的数据格式问题，作为测试数据非常具有典型性。

5.2.1 文件上传存储功能测试

文件上传存储的界面如图 5-1 所示，包括浏览文件按钮、是否添加备注选项和上传确认按钮。

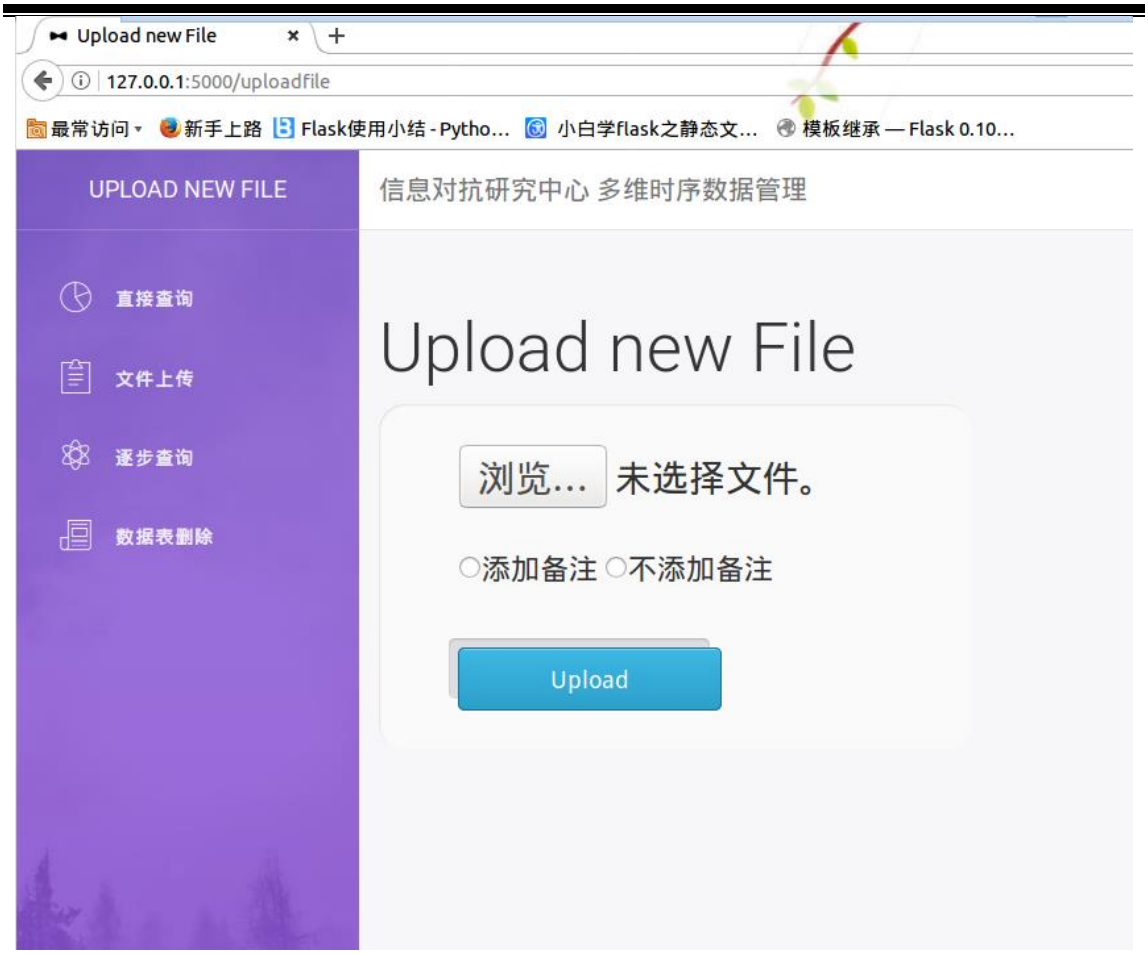


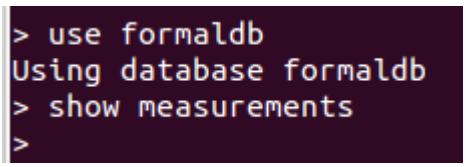
图 5-1 文件上传界面图

为了直接测试文件上传存储功能，使用 InfluxDB 数据库自带的命令行管理功能查看文件数据的存储是否成功。其使用界面如图 5-2 所示，展示了数据库中所有的已经存在的数据库。

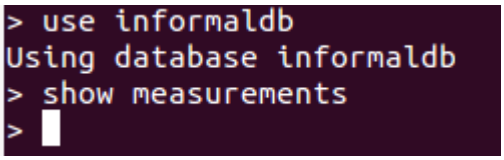
```
rosyphoton@ubuntu:~/myproject/flaskcode/flaskr$ influx
Connected to http://localhost:8086 version 1.2.2
InfluxDB shell version: 1.2.2
> show databases
name: databases
name
----
_internal
formaldb
informaldb
message

>
```

图 5-2 InfluxDB 命令行管理展示所有数据库

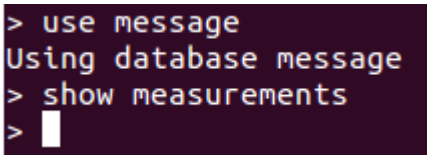
直接使用查询语句查询为存储创建的数据库 formaldb、informaldb 和 message 中的内容如图 5-3 所示。



a) 数据库 formaldb 中的内容



b) 数据库 informaldb 中的内容

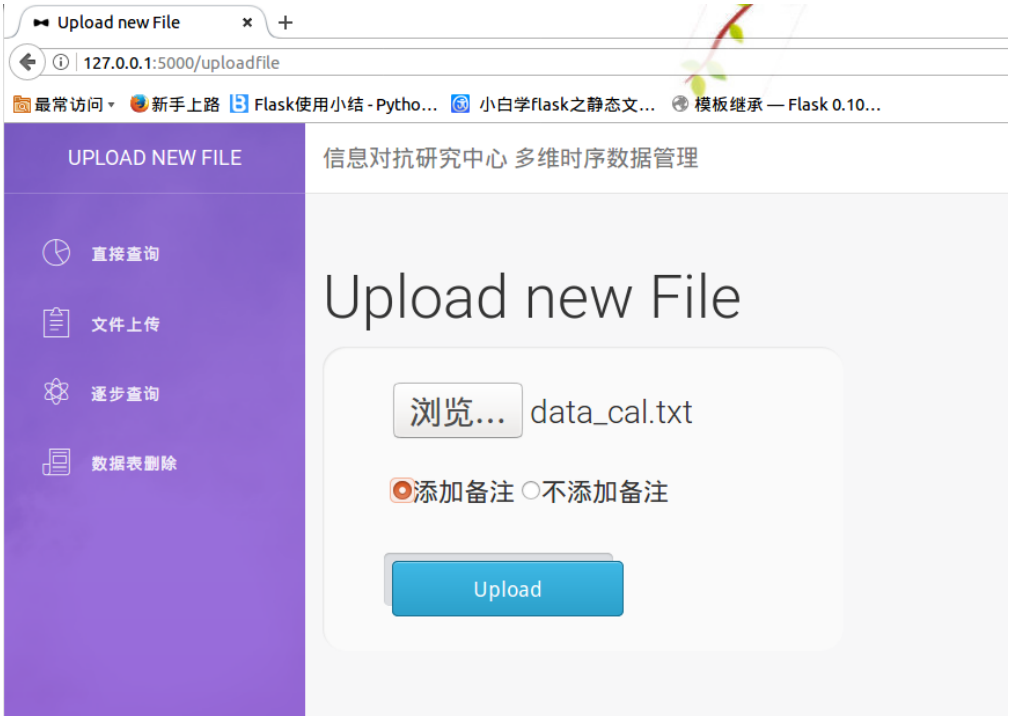


c) 数据库 message 中的内容

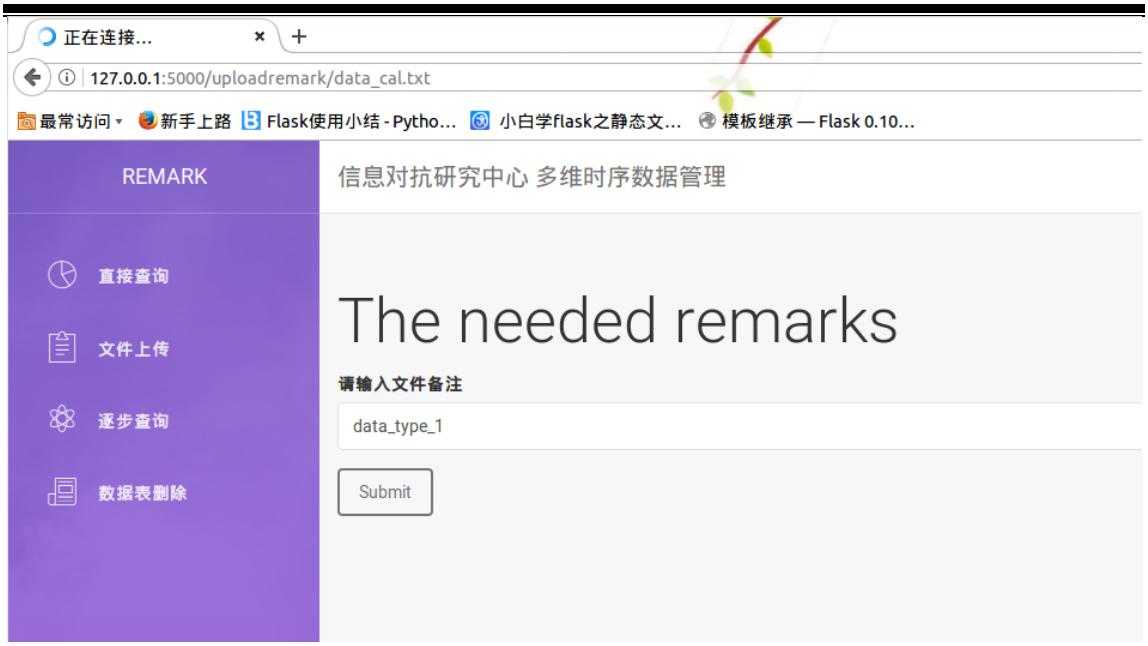
图 5-2 查询多个数据库中的内容

从结果可以看出这三个数据库中都没有数据表存在即数据库没有任何内容。接下来使用数据库的文件上传存储功能依次上传三种类型的数据文件。

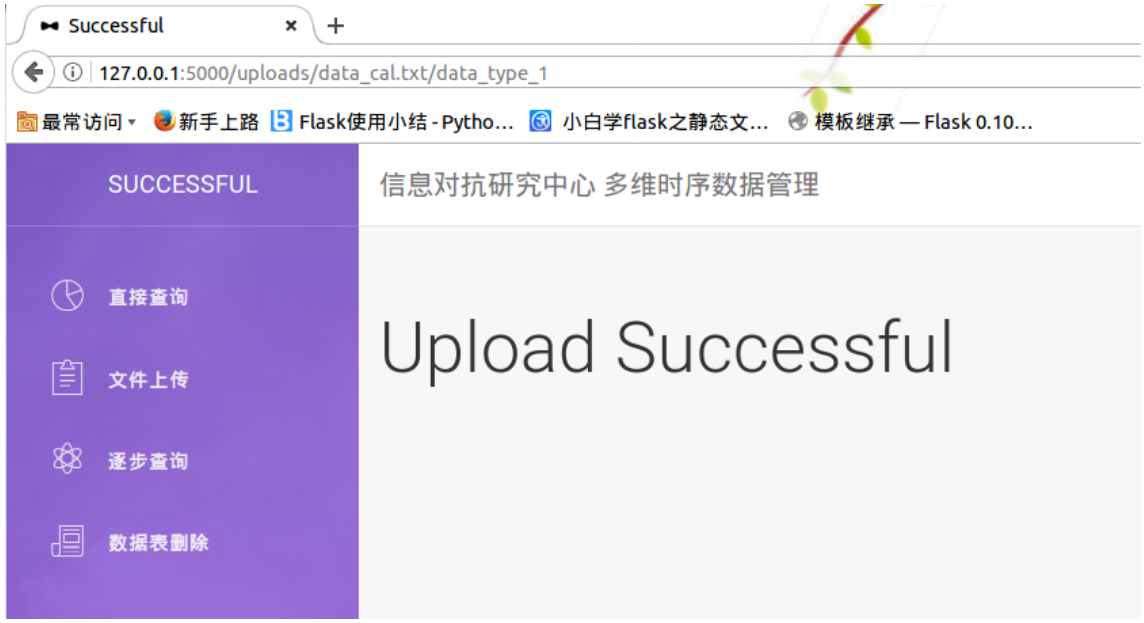
第一类数据文件的上传过程如图 5-3 所示。



a) 浏览并选择上传文件，选择添加备注选项，点击 Upload 按钮完成文件上传



b) 为文件数据添加备注



c) 文件上传并存储成功

图 5-3 数据文件上传过程

文件上传存储成功后使用 InfluxDB 的命令行功能进行数据查询，查询过程以及查询结果如图 5-4 所示。

```
> use formaldb
Using database formaldb
> show measurements
name: measurements
name
----
data_type_1

>
```

a) 显示数据库中所有的表

rosyphoton@ubuntu: ~ | rosyphoton@ubuntu: ~/myproject/flaskcode/flaskr | rosyphoton@ubuntu: ~/myproject/flaskcod

1491131827000000000	49351.51362	0.0	data_cal.txt	3.60	31.66N	119.79E	1	data_type_1	-3439.331055	-32189.941406	37374.87793
1491131827100000000	49352.884329	0.0	data_cal.txt	3.60	31.66N	119.79E	1	data_type_1	-3442.382812	-32202.148438	37371.826172
1491131827200000000	49352.119688	0.0	data_cal.txt	3.70	31.66N	119.79E	1	data_type_1	-3445.43457	-32217.407227	37368.774414
1491131827300000000	49350.498938	0.0	data_cal.txt	3.70	31.66N	119.79E	1	data_type_1	-3445.43457	-32208.251953	37374.87793
1491131827400000000	49350.851833	0.0	data_cal.txt	3.60	31.66N	119.79E	1	data_type_1	-3442.382812	-32189.941406	37374.87793
1491131827500000000	49352.589401	0.0	data_cal.txt	3.60	31.66N	119.79E	1	data_type_1	-3439.331055	-32192.993164	37374.87793
1491131827600000000	49353.608405	0.0	data_cal.txt	3.70	31.66N	119.79E	1	data_type_1	-3439.331055	-32205.200195	37371.826172
1491131827800000000	49350.625023	0.0	data_cal.txt	3.70	31.66N	119.79E	1	data_type_1	-3445.43457	-32205.200195	37377.929688
1491131827900000000	49350.435181	0.0	data_cal.txt	3.70	31.66N	119.79E	1	data_type_1	-3442.382812	-32189.941406	37377.929688
1491131827900000000	49353.205705	0.0	data_cal.txt	3.70	31.66N	119.79E	1	data_type_1	-3445.43457	-32217.407227	37371.826172
1491131828000000000	49351.640785	0.0	data_cal.txt	3.70	31.66N	119.79E	1	data_type_1	-3439.331055	-32192.993164	37374.87793
1491131828000000000	49352.983902	0.0	data_cal.txt	3.70	31.66N	119.79E	1	data_type_1	-3445.43457	-32217.407227	37371.826172
1491131828100000000	49350.596971	0.0	data_cal.txt	3.70	31.66N	119.79E	1	data_type_1	-3445.43457	-32208.251953	37374.87793
1491131828100000000	49353.653765	0.0	data_cal.txt	3.70	31.66N	119.79E	1	data_type_1	-3442.382812	-32202.148438	37371.826172
1491131828400000000	49350.030318	0.0	data_cal.txt	3.70	31.66N	119.79E	1	data_type_1	-3445.43457	-32192.993164	37377.929688
1491131828500000000	49350.774919	0.0	data_cal.txt	3.70	31.66N	119.79E	1	data_type_1	-3439.331055	-32189.941406	37374.87793
1491131828600000000	49353.583417	0.0	data_cal.txt	3.70	31.66N	119.79E	1	data_type_1	-3439.331055	-32199.09668	37371.826172
1491131828600000000	49351.141816	0.0	data_cal.txt	3.70	31.66N	119.79E	1	data_type_1	-3448.486328	-32214.355469	37374.87793
1491131828700000000	49353.485019	0.0	data_cal.txt	3.70	31.66N	119.79E	1	data_type_1	-3445.43457	-32211.303711	37371.826172
1491131828700000000	49349.880558	0.0	data_cal.txt	3.70	31.66N	119.79E	1	data_type_1	-3448.486328	-32199.09668	37374.87793
1491131829000000000	49350.494068	0.0	data_cal.txt	3.70	31.66N	119.79E	1	data_type_1	-3445.43457	-32189.941406	37374.87793

b) 查询该表中的所有数据

图 5-4 查询数据文件上传的结果

从测试结果来看第一类数据上传成功，可以得出数据文件上传存储功能对第一类数据文件的测试案例测试成功。

对与第二类数据和第三类数据测试案例的测试过程和测试结果与第一类数据类似，不同的是第二类数据在文件上传时需要手动输入日期数据，会显示一个日期输入的页面如图 5-5 所示。第三类数据会存储在 informaldb 数据库中且数据会分成三个表存储。在三类数据文件存储的同时会将文件信息存储在 message 数据库中，这一步骤是默认进行的操作，查看其数据结果如图 5-6 所示。

图 5-5 第二类数据文件上传日期数据输入

```
> select * from filename_filetype
name: filename_filetype
time                           filename     num remark
----                           --------     --- ------
2017-06-17T13:35:20.376854532Z data_cal.txt 1   data_type_1
2017-06-17T14:38:25.661337924Z d081555      2   Nothing

>
```

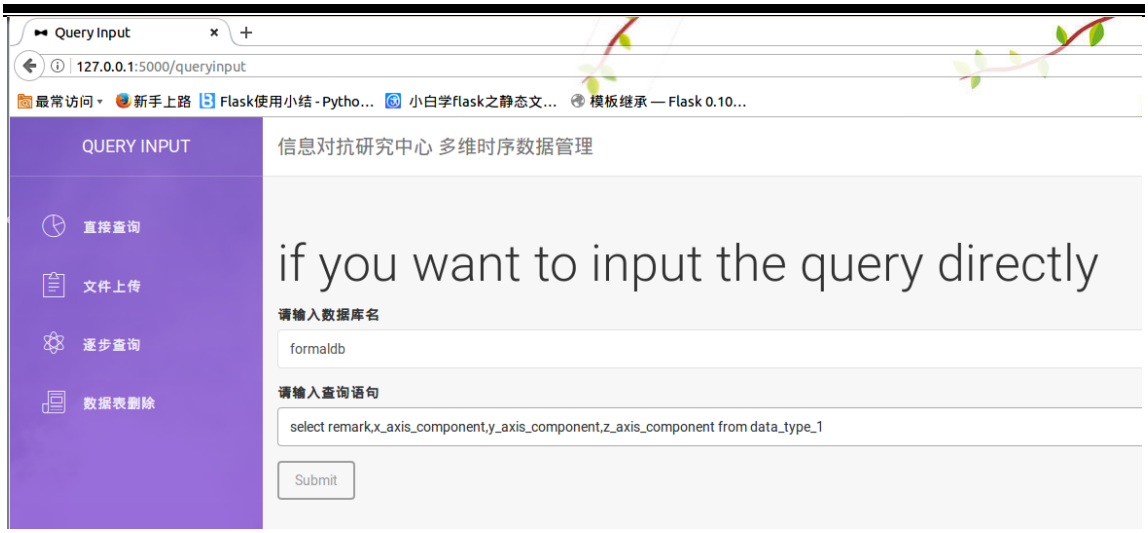
图 5-6 查询 message 数据库中存储的文件信息数据图

由以上测试结果数据可以看出系统的数据文件上传存储功能测试成功。

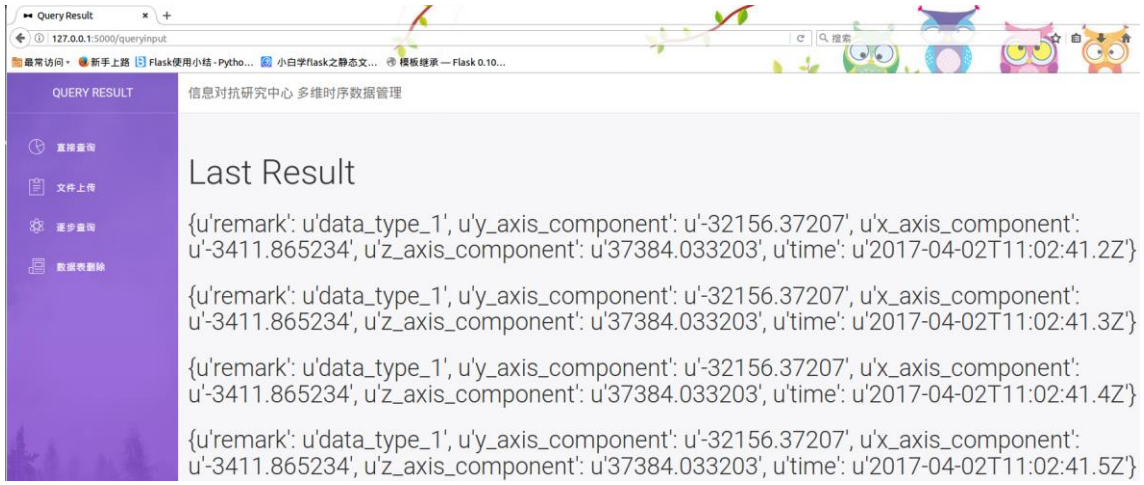
5.2.2 数据直接查询功能测试

数据直接查询功能是直接输入需要查询的数据库和查询语句进行数据查询的功能模块。测试样例直接使用第一步测试存如数据库的三种类型的数据。测试查询第一类数据的备注和 x 轴分量、y 轴分量、z 轴分量数据，依次在输入页面中输入：“formaldb”和“select remark, x_axis_component, y_axis_component, z_axis_component from data_type_1”。

其测试结果如图 5-7 所示。



a) 查询信息输入页面

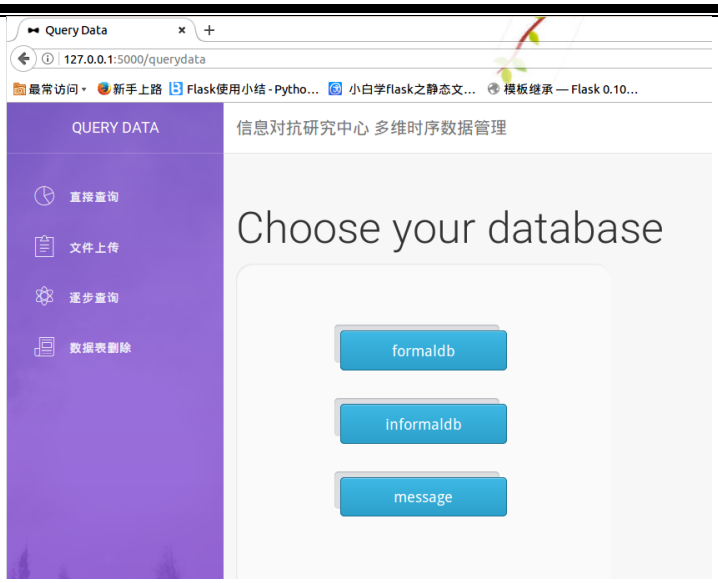


b) 查询结果页面

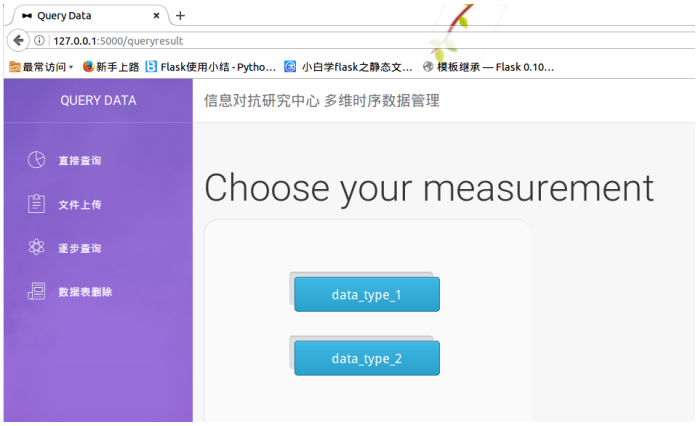
图 5-7 数据直接查询功能测试图

5.2.3 数据逐步查询功能测试

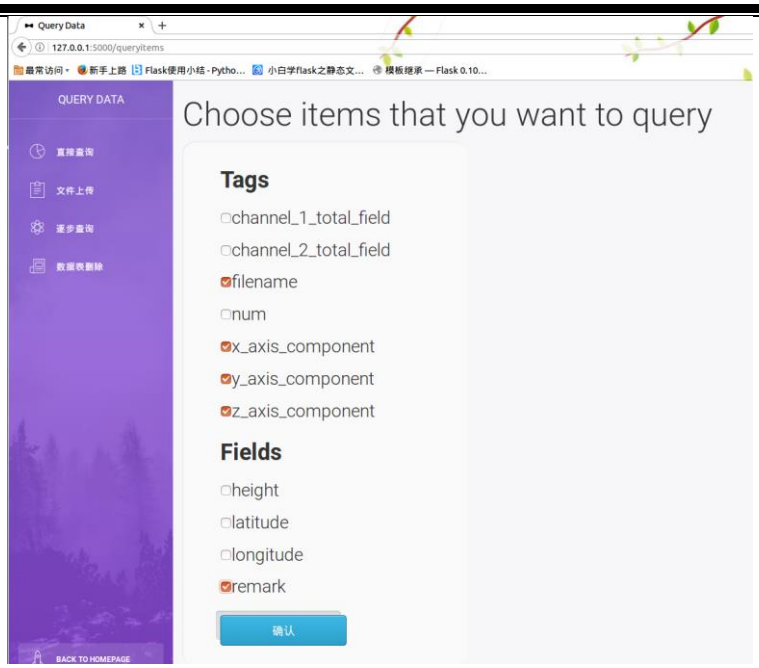
数据逐步条件查询功能测试是从所有数据库开始逐步进行选择最终选择需要查询的几个数据属性得到这几个属性的查询结果。查询过程如图 5-8 所示，先在页面给出的几个数据库中选择要查询的数据库，然后在给出的数据中的所有表中选择要查询的表，最后在这个表中所有的属性中选择要查询的所有属性提交查询能够得到查询数据。



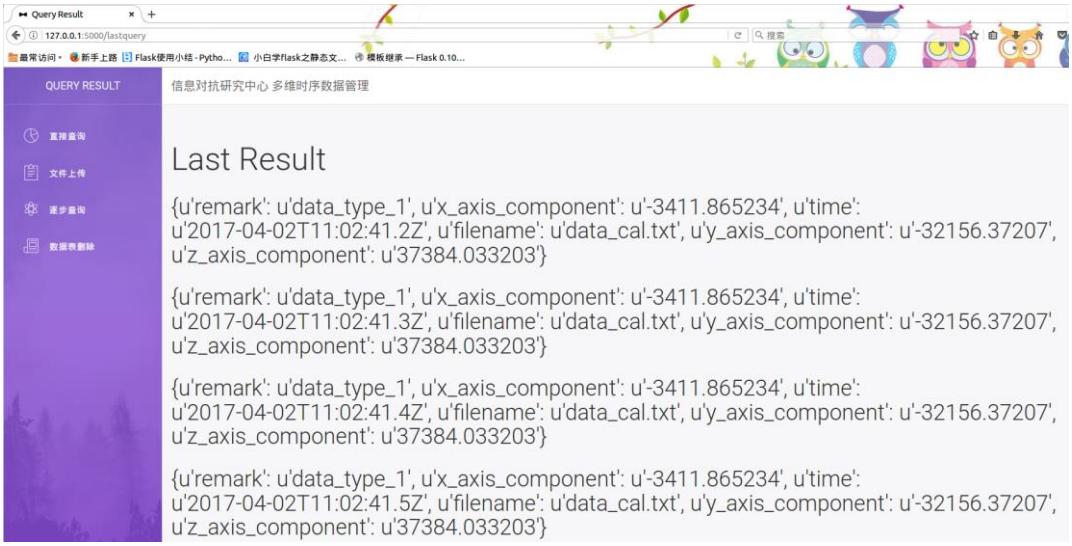
a) 显示所有数据库选项页面



b) 显示数据库中所有表选项页面



c) 选择查询数据属性页面



d) 查询结果显示页面

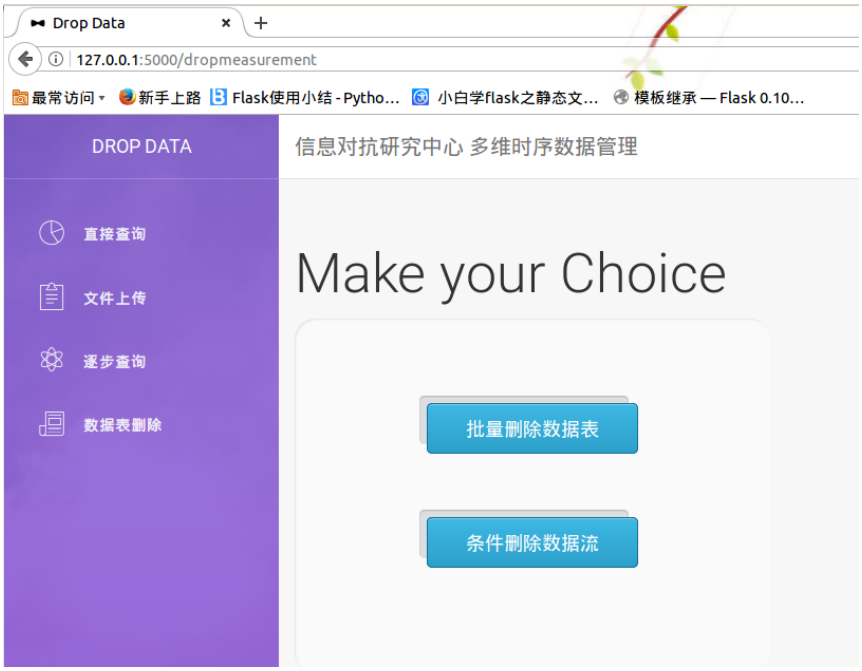
图 5-8 数据逐步条件查询测试过程图

由查询结果可以看出数据逐步条件查询功能测试成功，系统功能正常。

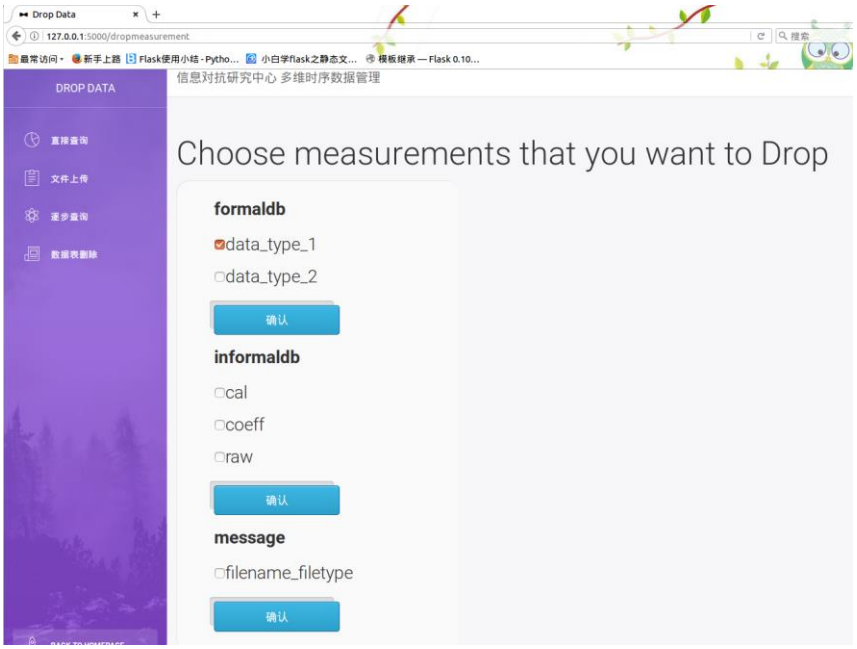
5.2.4 数据表删除功能测试

数据表的查询功能包含了两种表的删除模式，第一种删除模式中在所有表中

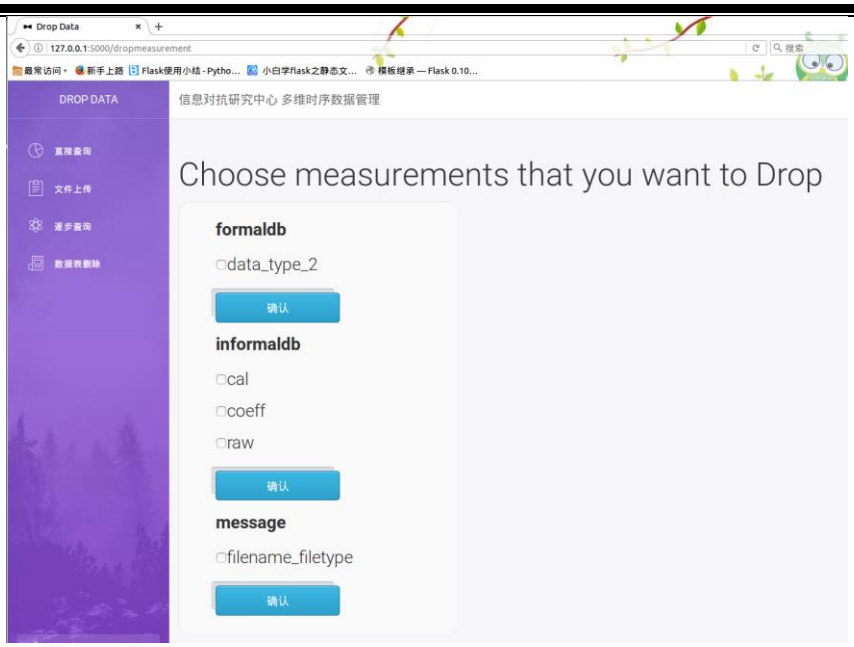
选择直接选择要删除的几个在同一个数据库中的表可以同时批量删除，删除操作测试如图 5-9 所示。第二种删除模式需要选择存储过的数据的文件名进行删除，删除操作测试如图 5-10 所示。



a) 数据表删除模式选择页面

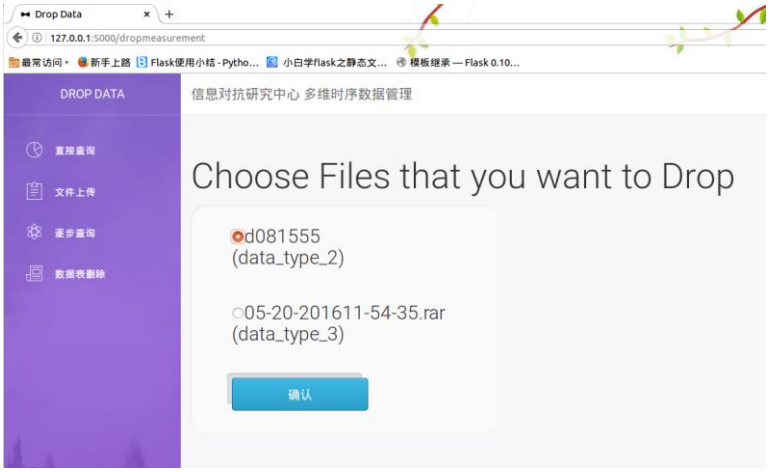


b) 第一种删除模式下的数据表选择页面

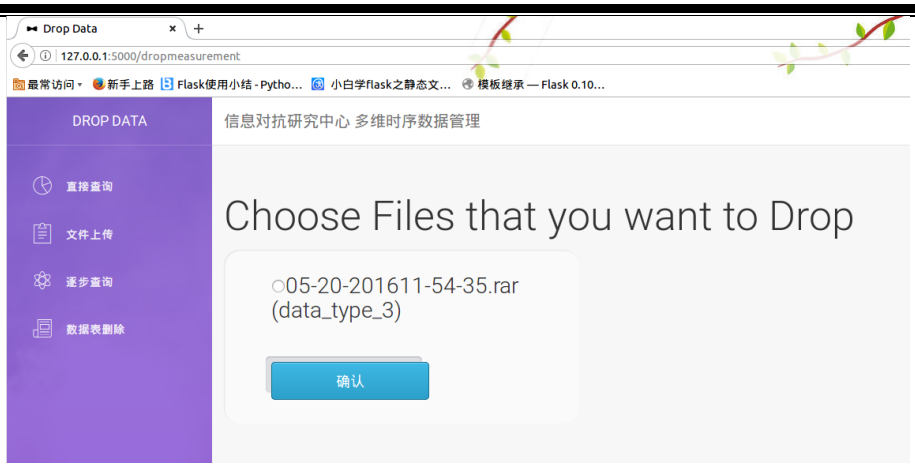


c) 选择的数据表删除后页面

图 5-9 第一种删除模式的测试过程



a) 第二种删除模式的文件名选择页面



b) 根据文件名删除数据后页面

图 5-10 第二种删除模式的测试功能

从两种删除模式的测试结果可以看出数据都按照选择条件被删除，可以得到系统的删除功能模块测试成功的结论。

5.3 本章小结

在本章中主要介绍了系统的各个功能模块在设计好的几个测试案例下的测试状况，从测试结果来看都成功实现了系统设计以及系统实现所期望的功能需求，测试过程中遇到了一些小问题，但都进行了改进，最终的测试结果完整满足了系统需求。

结　论

本文在分析了目前时间序列数据库现状和时间序列数据特征的基础上，以时间序列数据库 InfluxDB 为内核，设计了面向对多种时间序列数据管理和能为其他程序提供数据服务的时序数据库系统，根据实际的应用需求，开发完成了相关的功能模块和网页操作界面，并使用典型的时序数据完成了对设计的数据库模块的功能测试。

通过对时间序列数据的存储研究和对系统自身功能模块测试结果得到了以下结论：

(1) 在研究了时间序列数据的多种存储方式：文本存储、关系数据库存储和非关系型数据库存储后，采用了时间序列数据库 InfluxDB 这种非关系型数据库作为系统的存储内核。InfluxDB 能够有针对性地存储时间序列数据，并且支持类 sql 语句操作，极大方便数据管理。

(2) 针对多种类型的磁场测量数据，经过对其不同数据格式，不同属性个数，不同数据长度的特征分析，设计出了统一的数据处理格式和数据存储结构。使系统能够自动存储和处理多种不同类型的数据。

(3) 实现了基于 Python 的数据库通用访问接口函数的开发，利用 Python 函数调用的便捷性为其他程序提供了数据使用的便利。优化了 Python 语言对数据库的相关操作，有效地提升了系统开发效率。

(4) 分析 NoSQL 数据库的特点，使用了 Flask 作为系统的网页开发框架，利用 Flask 对 NoSQL 数据库的强大支持，有效降低了系统的开发难度，并提高了数据的网络处理效率。

(5) 结合具体的功能需求，独立开发了相应的功能模块，设计的模块之间满足高内聚低耦合的特点，确保了系统的稳定性和可靠性。在虚拟环境中进行系统开发和运行测试，保证了系统环境的独立并验证了所设计的数据库系统功能模块的可用性和合理性。

(6) 系统具有可扩展性，未来可以针对更多的数据类型和不同的数据存储文件进行数据处理方式和功能的扩展开发。

参考文献

[1] 余涛. 面向测控系统数据管理的时间序列数据库设计与开发[D]. 北京化工大学, 2016.

[2] 韩骅宇. 时序数据的高效存储与检索[D]. 北京交通大学, 2014.

[3] 张青. 一种多对象时间序列数据存储设计[J]. 计算机光盘软件与应用, 2013(15):61-62.

[4] 刘城成. 时间序列数据建模与存储研究[D]. 华中科技大学, 2007.

[5] 王建光. 大规模时间序列数据存储系统的研究与实现[D]. 华中科技大学, 2013.

[6] 邹同春. 海量时序数据的压缩存储方法研究[D]. 哈尔滨工业大学, 2016.

[7] 金鑫. 海量时序数据高可用性实时存储技术研究与应用[D]. 浙江大学, 2012.

[8] 郑毅. 时间序列数据分类、检索方法及应用研究[D]. 中国科学技术大学, 2015.

[9] 曾碧贵, 叶少珍. 时间序列数据摘要与索引机制[J]. 工业控制计算机, 2017, 30(1):59-60.

[10] Rafiei D. On Similarity-Based Queries for Time Series Data[J]. Acm Sigmod Record, 1999, 26(2):13--25.

[11] Database F F. Time series database[J]. 2015.

[12] Whistler D. The Design of a Data Base Management System for Economic Time Series Data.[C]// Statistical and Scientific Database Management. 1986:197-202.

[13] Chilingaryan S, Beglarian A, Kopmann A, et al. Advanced data extraction infrastructure: Web based system for management of time series data[C]// 2010:042034.

哈尔滨工业大学本科毕业设计（论文）原创性声明

本人郑重声明：在哈尔滨工业大学攻读学士学位期间，所提交的毕业设计（论文）《多维时间序列数据综合管理系统》，是本人在导师指导下独立进行研究工作所取得的成果。对本文的研究工作做出重要贡献的个人和集体，均已在文中以明确方式注明，其它未注明部分不包含他人已发表或撰写过的研究成果，不存在购买、由他人代写、剽窃和伪造数据等作假行为。

本人愿为此声明承担法律责任。

作者签名：　　日期：　　年　　月　　日

致　谢

衷心感谢韩琦老师对本人的精心指导。在毕业设计的完成过程中对我提出了很多宝贵的指导性意见，同时也对我在完成毕业设计过程中遇到的困难耐心的进行解答，帮助我最终完成了毕业设计，他的言传身教将使我终生受益。

感谢实验室的赵冠一学长，在我的毕业设计完成过程中为我提供了测试数据支持和设计方向指导

感谢项目同组的史双玮同学，在我们的毕设项目联合调试时帮我的程序发现了很多问题，并最终实现了项目的完整集成。

感谢实验室全体老师和同窗们的热情帮助！

感谢 2013 级信息安全专业全体同学的支持！