摘要

时序数据是按时间索引的一系列数据，例如汽车引擎中各类传感器采集到的数据，其中包括油量、速度、喷油量等。这类时序数据通常维数较高，体量较大。高体量意味着海量的时序数据将对存储系统造成很大的压力，而高维度意味着我们必须采用特殊的压缩算法来解决多维度的数据压缩难题并保证可靠的效率。因此本论文先通过特定的数据压缩算法实现多维度数据压缩，并将压缩后的数据进行持久化，同时为了方便直观的提取时序数据压缩过后包含的关键信息，我们使用蜡烛图来进行时序数据的可视化。

关键词： 时序数据；数据压缩；持久化；可视化；蜡烛图

Abstract

Time-series data is a collection of data indexed by time sequence. For instance, there are various types of data collected by a car engine sensor, including oil, speed, fuel injection quantities and so on. Such time-series data usually has high dimensions and a large volume. A large volume means that massive time-series data will cause great pressure to our storage system, while high dimensions mean that we must use special compression algorithms to solve the multi-dimensional data compression problem and keep a controllable efficiency at the same time. Therefore, in this thesis we first use a specific algorithm to implement multi-dimensional data compression, then store our compressed data for persistence. In order to intuitively extract key information from our compressed data, we meanwhile use candle-stick to visualize time-series data.

Keywords: Time-series Data; Data Compression; Persistence; Visualization; Candlestick Charts

目录

第 1 章 引言

时序数据是按时间节点进行记录的数据集合，与非时序数据相比，它的顺序性要求更高，数据的应用范围更广。时序数据产生于生活中的各个方面，例如气象观测中的大气数据、汽车发动机的各项传感器数据。这类数据不仅维度很高，同时产生的数据体量也非常大。目前随着时序数据规模的增大，不加处理的原始数据已经对关系数据库特别是对一些移动平台和其它嵌入式设备造成了很大的存储压力。由于时序数据在一定时间内的重复率比较高，例如汽车的速度在某段时间内都保持在一定速度，油料温度也都稳定在一定范围内，特别是对于发动机磨粒浓度这类数据很有可能长时间都稳定在某固定值，此类数据导致了存储空间的严重浪费。高维时序数据是时序数据在空间上的扩展，相比于一维数据，多维时序数据复杂度更高，数据要求的存储空间更大。

理想的数据压缩效果是实现最小的存储成本，提高数据的传输效率。随着云计算和物联网的不断发展，数据的压缩存储技术成为海量数据研究的重要内容，根据数据压缩后是否能完全还原为原始数据我们将数据压缩分为有损压缩和无损压缩，无损压缩的期望压缩比通常在 1/2 到 1/5，相对于无损压缩，有损压缩通常只要求压缩后的数据不影响特定范围内的正常使用，有损压缩广泛运用于视频和音频的压缩。

单维时序数据已经存在很多的成熟的压缩算法，例如 SDT 算法。相比于单维时序数据多维时序数据的压缩更为复杂，但是生活中需要对高维时序数据进行压缩的场合却是很多，目前存在一些针对于时序数据的压缩算法，例如旋转门，稳态阈值，线性外插值等压缩算法，但是大多属于一维数据的有损压缩算法，不能直接运用于多维时序数据的数据压缩，而且这些压缩算法的复杂度过高，海量的数据更是需要庞大的计算资源，而且这些压缩数据压缩后不能直接存储于关系数据库，这很大程度的影响了这些数据对于业务的使用。因此本论文设计了一种针对于高维时序数据的压缩方法，并提供了一种附加的持久化方式。

对于压缩处理过后的时序数据如何快速的从中找到自己需要的信息这是一个比较通用的业务场景，数据可视化的目的就是将这些晦涩难懂的数据在与用户的交互中展示出来。可视化的图表，更易于我们观察，对于发现数据的特

征和深入的理解数据隐含的行为有更大的帮助, 数据可视化不同于将所有的数据通过图标直接显示, 数据体量超大的时序数据, 直接用可视化系统观测不仅对可视化系统造成了巨大的压力, 更重要的是这通常会直接导致高价值的数据被淹没。

根据现有的绝大部分业务场景, 一般的持久化都是采用的基于关系操作的关系数据库, 例如 PC 端常用的 MySQL, 移动端广泛运用的 SQLite, 当然也不排除一些特殊的业务场景使用了类似于 MongoDB 的非关系数据库。在本论文中我们给出了一种基于蜡烛图的的海量时序数据的可视化方法, 具体的可视化我们使用开源的图形化报表库, 在可视化的过程中我们先对时序数据进行高效的线性压缩, 生成特定的区间, 对于每个区间的统计数据, 我们采用折线图和蜡烛图进行可视化。这种方法不仅能够对原始数据进行大范围的压缩, 还能在压缩的基础之上通过用户的需求分层显示压缩后的效果

本论文依次介绍了相关的应用历史背景, 算法实现, 程序的相关实现等其它步骤, 并在最后给出了一些针对于特定场合的压缩选项和优化。

第 2 章 预备知识

2.1 数据压缩简介

数据压缩的一般前提是不损失有用信息, 理想的效果是达到最小的存储空间, 同时提高传输和存储的效率。我们通过一定的算法重新组织数据, 从而来减少数据的冗余。数据压缩可分成两种类型, 一种叫做无损压缩, 另一种叫做有损压缩。无损压缩通常能保证在压缩数据后我们可以通过目标数据进行重构, 重构出的数据能和原始数据保持一致。常见的无损压缩包括磁盘文件的压缩, 无损压缩的效果一般能把普通文件的数据压缩到原来的 1/2 到 1/5. 常用的无损压缩算法包括霍夫曼算法, LZW(Lenpel-Ziv&Welch) 压缩算法。有损压缩是指在压缩后我们可以通过目标数据重构出原始数据的主要轮廓而这些数据不会影响到我们对他们的使用。

2.2 数据压缩的理论极限

了解了 2.1 中数据压缩的概念和相关分类, 我们给出数据压缩的理论极限, 值得一提的是对于时序数据的数据点压缩也存在压缩极限当数据压缩到一定程度之后, 再提高压缩比例, 带来的结果就是数据的永久性损害。

假设在均匀分布的情况下, 我们采集到的某一维数据的取样值在存储文件中出现的概率是 p, 那么在这个取样值最多可能出现 1/p 种情况。因此我们需要 $\log_2(1/p)$ 个存储单位表示该采样数据。我们把这个结论可以推广到一般情况。假设我们采样的某一维时序数据由 n 个采样值组成, 每个部分的内容在存储文件中的出现概率分别为 p1、p2、...pn。那么, 替代符号占据的存储单位最少为下面这个表达式：

$$\log_2(1/p1) + log_2(1/p2) + ... + log_2(1/pn) = \sum_1^n log_2(1/pn) \quad (2\text{-}1)$$

考虑单个采样值, 它对应的存储单位为：

$$log_2(1/pn)/n = log_2(1/p1)/n + log_2(1/p2)/n + ... + log_2(1/pn)/n \quad (2\text{-}2)$$

2.3 蜡烛图

蜡烛图 (candlestick charts) 也叫 K 线图，蜡烛图来源于对统计数据的分析，例如股价，其中包含股价的开盘价，收盘价，最高价和最低价。

K 线图的状态可以分为翻转形态，整理形态等。K 线图由于其细致明确的表达效果，引入到了股市和期货市场，K 线图由统计区间的开始值，最大值，最小值，结束值组成，假设给定一个统计区间，当我们确定区间的开始值和结束值时，我们把这两者中间的部分用对应的实体矩形画出，当开始值小于结束值时，我们用红色标注，此时 K 线被称为阳线，当开始值大于结束值时我们用黑色标注，此时对应的 K 线我们称之为阴线。

K 线图绘制的技术指标和 K 线图的分析对于我们从中寻找有用的数据有很大的帮助。它可以作为我们从时序数据中寻找奇点数据最有力的工具，根据经典的 K 线图或者图上的某一指标分析和得出的结论不一定完全正确，但是结合我们时序数据压缩过后的数据查询，就能准确而快速的寻找到问题所在。

注意：时序数据绘制过程中的技术指标是对应于某一段时间内的统计分析数据，并不是和原始数据完全一样。因此这些数据的最终效果可能会受到人为因素的控制，例如我们设置的压缩比过大，那么此时的绘图界面可能和原始数据相差较大。此时我们应该结合主要因素来判断图形是否能表达原始数据的真实趋势，以及我们是否需要调整原始数据的压缩比和其它参数，以达到理想的效果。

2.4 JavaScript&Echarts

JavaScript 是一种直译式脚本语言，是一种动态类型、弱类型、基于原型的语言，内置支持类型。它的解释器被称为 JavaScript 引擎，为浏览器的一部分，广泛用于客户端的脚本语言，最早是在 HTML（标准通用标记语言下的一个应用）网页上使用，用来给 HTML 网页增加动态功能。

HTML5 的设计目的是为了在移动设备上支持多媒体。新的语法特征被引进以支持这一点，如 video、audio 和 canvas 标记。HTML5 还引进了新的功能，可以真正改变用户与文档的交互方式，包括：

- 新的解析规则增强了灵活性
- 淘汰过时的或冗余的属性
- 一个 HTML5 文档到另一个文档间的拖放功能
- 多用途互联网邮件扩展（MIME）和协议处理程序注册
- 在 SQL 数据库中存储数据的通用标准（Web SQL）

JavaScript&CSS&HTML. 是前端开发的必备工具, JavaScript 作为事件处理和交互的核心起到了重要的作用，在 HTML5 中新增了 canvas 元素，这为我们绘制高质量的蜡烛图提供了一个非常不错的选择, 结合使用百度提供的 echarts 图形化报表组件，能够快速高质量的帮助我们完成工作。当然作为绘图工具。你依然可以选择其它的工具，例如基于 Java Swing 的 UI 组件库，也可以选择基于 Matlab 的绘图套件，不过这些工具不一定适合在工程下的业务环境。

2.5 JAVA& 设计模式

高维时序数据压缩的研究成果适用于工程中的很多场景, 而 Java 语言也是现阶段业务场合中使用最为频繁的编程语言，论文的设定环境为基于 Java EE 虚拟机的嵌入式平台，因此需要了解 Java EE 相关的一些基础知识。

JDBC 是 Java 语言中与数据库交互最为流行的方式，绝大部分数据库都有特定的 JAR 包来实现这些接口。而现在最流行的持久化框架包括 Hibernate, Ibatis 都是基于 JDBC 完成的，学会使用 JDBC 是完成该论文的重点。

设计模式 (Design Pattern) 是一套反复被使用，多数人知晓，经过分类与编目的代码设计经验的总结, 使用设计模式的目的是使代码的可用度达到最大化, 同时使用良好的设计模式对于代码的维护也有很重要的作用, 在本论文中需要了解的设计模式包括线程安全的单例模式，工厂模式，原型模式，和装饰者模式以及访问者模式。

第 3 章 时序数据压缩算法和可视化

3.1 时序数据压缩算法与实现

3.1.1 问题描述

汽车运行过程中引擎的相关参数我们可以通过传感器获得, 例如汽车发动机的转速, 汽车的进气量和喷油量。这些数据的维度可以达到两个数量级以上, 也就是说我们采集到的数据在数据库当中可以单个表达到 100 列。假设我们的采样频率为 100, 可以计算每小时我们采集数据的次数为 3600*100. 很明显如果汽车长时间运转, 这些数据将对汽车的存储系统造成极大的压力。这些数据全部存储起来将浪费大量的存储资源, 但是如果不加处理的直接丢弃, 那么一旦出现汽车故障, 我们将不能获取即时有效的原始数据。

通过我们对时序数据的观测, 在这些时序数据中存在大量的数据冗余, 原因很直接, 因为大部分的数据通常在某段时间都是稳定的, 例如引擎稳定运行时的温度, 特别是像汽车的档位这类数据, 很有可能长时间不改变, 对于日常的业务场景来说, 维护人员很难从海量的时序数据中快速找到自己需要的异常数据, 因此我们可以允许在保存的过程中与原始数据有一定的误差, 在存储的过程中我们依然采用了关系数据库, 这不仅是因为关系数据库的技术成熟, 而且目前存在的一些早期的时序数据库的实现原型也是基于关系数据库而来。

时间本身充满了多变性, 它可以是一个时间点, 也可以是一个时间间隔, 既可以是线性的, 也可以是周期的。如果考虑时序数据的维数, 可想而知这样的数据它的复杂度是相当高的, 对于这样的数据, 如何可视化也呈现出了多样性, 根据特定的时序数据, 现在已经存在了一些可视化框架, 例如针对模拟数据我们可以采用 SimVis, 而针对新闻数据我们可以采用 ThemeRiver 进行分析。

时序数据可视化从通用的角度来看可以认为是从维度到空间的映射, 一般可视化框架有 XmdvTool 和 Visage 相对标准的可视化技术, 而且平行坐标, 复杂坐标都可以用来可视化时序数据, 但对于海量的时序数据, 这些框架显得捉襟见肘。

对于数据量庞大的时序数据, 我们可以选择基于蜡烛图的可视化方法, 蜡烛图 (candlestick chart) 又称 K 线图, K 线图可以容易的表示统计数据中的若干

内容，关于 K 线图的具体内容请参考背景知识。

3.1.2 算法的基本思路

我们根据时序数据的维数和设定的压缩比例，对输入的时序数据进行单维处理。在单维数据中我们将无明显波动的数据，合并到相同的区间中，注意在合并的过程中，同时更新对应区间的时间点，当我们每个区间的数据点数超过了限制之后，我们采用对应的区间合并算法，将每维数据进行区间合并。处理完单维数据之后，我们将所有维数的数据进行区间合并，来产生对应的最终输出。

如图所示3-1:

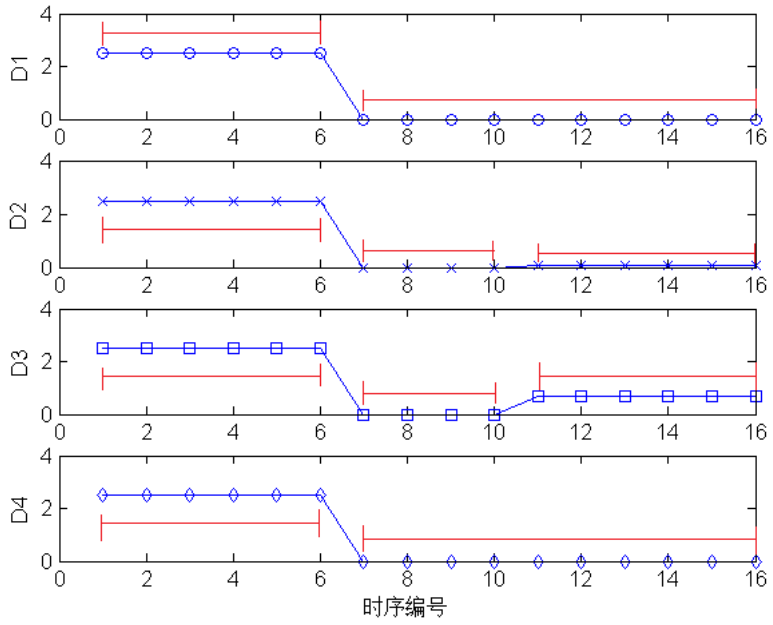

图 3-1　时序输入数据

假设我们采样到的时序数据包含 4 维，D1 ~ D4，其中我们可以看到 D1 从 t1 到 t6 的值未发生变化，t7 到 t16 区间内的值相同，因此我们将 D1 划分为 [1,6] [7,16] 两个区间，对于 D2 ~ D4 我们也采用相同的方法，D2 和 D3 将被划分为 3 个区间 [1,6]、[7,10]、[11,16]. 同时我们将这几个区间的时间标签提取出来并进行排序重排后的时序列表对应为 [1,6,7,10,11,16]，然后我们重新生成时序数据，此时共产生了 6 个时刻对应的 6 条数据，每个时刻对应的值为该区间的

平均值，最后通过压缩后的结果如下表所示。

T	D_1	D_2	D_3	D_4
1	2.5	2.5	2.5	2.5
6	2.5	2.5	2.5	2.5
7	0.0	0.0	0.0	0.0
10	0.0	0.0	0.0	0.0
11	0,0	0.1	0.7	0.0
16	0.0	0.1	0.7	0.0

3.1.3 数据区间的定义

设 D 维时序数据的每一分量为 D_i，压缩后的数据点数我们设置为 CP(D_i)，并将压缩梯度 Grad(D_i) 设置为 CP(D_i)*2^k(k=1,2,3...)，定义 Max(D_i) 为 D_i 已经扫描数据的最小值，Min(D_i) 为 D_i 已经扫描过的数据最大值，IL(D_i) 用于存储已经扫描过的区间列表，列表的每一个区间是一个六元向量：

$$T =< t_s, t_e, v_{max}, v_{min}, v_{avg}, n > \tag{3-1}$$

其中 t_s 和 t_e 分别表示区间的开始时间标签和结束时间标签，v_{max},v_{min},v_{avg} 分别表示区间内采样值的最大值，最小值，和平均值。其中 n 表示所有的数据点数。每个区间是 1 个以上的原始数据点合并之后的结果，同一个区间覆盖的原始数据点基本平稳，而相邻的区间分隔的时间节点应该对应原始数据中比较明显的拐点，在压缩的过程中我们在指定的输入条件 CP(D_i) 下，不断对 D_i 创建区间，扩展区间和合并区间。

新建区间 CI：高维度的时序数据在压缩的过程中必须保持原数据总体变化趋势这些基本特征，在数据处理的过程中，当遇到较大的数据奇点且不在约定的梯度等级时，此时我们需要创建新的区间段，新的区间段取决于奇点数据，也就是：

$$I_n =< t, t, d_{it}, d_{it}, d_{it}, 1 > \tag{3-2}$$

其中 d_{it} 为拐点数据，也就是当前时刻 t 的第 i 维的数据值。

扩展区间 EI: 在扫描数据序列的过程中，若当前数据点 d_{it} 属于当前的数据区间，则应对当前区间进行更新，区间扩展方法为：

$$I_{last} =< t_s, t_e, MAX(v_{max}, d_{it}), MIN(v_{min}, d_{it}), (v_{avg} * n + d_{it})/(n+1), n+1) > \quad (3\text{-}3)$$

其中 I_{last} 为 IL(D_i) 区间的最后一个元组，d_{it} 为当前时刻 t 关于 D_i 的分量

区间合并 MI: 区间合并发生在已有的区间总数超出了设定的数据点数 CP(D_i) 时进行，区间合并是指将原来相邻的区间合并为一个区间，设：

$$I_i =< t_s, t_e, v_{max}, v_{min}, v_{avg}, n > \quad (3\text{-}4)$$

$$I_{i+1} =< t_s^{'}, t_e^{'}, v_{max}^{'}, v_{min}^{'}, v_{avg}^{'}, n^{'} > \quad (3\text{-}5)$$

为数据区间列表中相邻的元组，则合并后数据区间 I_{lr}:

$$\begin{aligned} I_{lf} =< & MIN(t_s, t_s^{'}), MAX(t_e, t_e^{'}), MAX(v_{max}, v_{max}^{'}), MIN(v_{min}, v_{min}^{'}) \\ & , (v_{avg} * n + v_{avg}^{'} * n^{'})/(n + n^{'}), n + n^{'} > \end{aligned} \quad (3\text{-}6)$$

3.1.4 数据压缩算法

在对整个数据序列进行压缩的过程中，顺序扫描对应的数据，分别对每一分量进行区间压缩算法，得到所有不同分量的区间列表序列 IL。在区间的创建和扩展的过程中，对每个分量 D_i 判断 IL(D_i) 中的区间个数，和 CP(D_i) 的大小，如果 |IL(D_i)|>CP(D_i) 则按照区间合并操作 MI，选择性的对相邻区间进行合并，这里的选择性是指选哪些数据，分布较为接近和相似的区间进行合并，最后，取 IL 中所有的 IL(D_i) 中的元组标签并按时间先后进行排序，将排序好的时间标签，放入标签队列中 TL，最终生成分量数据，对应的如下算法：

算法 3.1 高维时序数据压缩算法 CompressInterval

输入： 分量压缩点数 CP(D_i)，原始数据序列 D

输出： 压缩后的数据序列

for ($i = 1$ to N) **do**
  Max(D_i)=0;Min(D_i)=0;IL(D_i)=[];
  Grad(D_i)=CP(Di)*k;
end for
while (未达到输入数据 D 结束位置，且下一条数据记录为 D_t) **do**

  for $i = 1$ to N **do**
    Max(D_i)=MAX(Max(D_i),d_{it});
    Min(D_i)=MIN(Min(D_i),d_{it});
    I_{last}=IL(D_i) 的最后一个区间；
    if ($d_{it} \geq I_{last} \times v_{min}$**and**$d_{it} \leq I_{last} \times v_{max}$) **then**
      $I_{last} = EI(I_{last}, d_{it})$
    else
      $IL(D_i)pushEC(d_{it})$
    end if
    if ($|IL(D_i)| > CP(Di)$) **then**
      $IL(D_i) = MergeInternal(IL(D_i), CP(D_i), Grad(D_i))$
    end if
  end for
end while
IL=$[IL(D_1), IL(D_2...IL(D_n)]$;
return GenData(IL); =0

3.1.5 区间合并算法

时序数据压缩算法 CompresInterval 先对每一分量的相关数据进行初始化，在我们依次读入每条数据之后，对于每条数据我们都逐渐更新最大值最小值，根据读入值的大小我们判断是否需要开辟一个新的区间，如果不需要开辟新的区间我们将直接对最后一个区间进行扩展，当处理完每条输入的数据之后，此时根据我们事先设定的 CP(D_i) 来判断区间内部数据的值是否达到了极限，当区间内部的数据点超过我们设定的值后，我们就可以使用区间合并算法来生成

新的数据。对应的区间合并算法如下所示：

算法 3.2 区间合并算法 IntervalMerge

输入： 需要合并前的区间列表 IL(D_i),CP(D_i),Grad(D_i)

输出： 通过合并算法之后产生的新的区间列表 IL(D_i)

R(D_i)=(Max(D_i)-Min(D_i))/Grad(D_i)
for ($i = 1$ to |IL(D_i)-1|) **do**

  if ($\exists x < Grad(D_i)$**and**$I_i.v_{min} \geq Min(D_i) + x * R(D_i)$**and**$I_i.v_{max} < Min(D_i) + (x+1) * R(D_j)$**and**$I_{i+1}.v_{min} \geq Min(D_i) + x * R(D_i)$**and**$I_{i+1}.v_{max} < Min(D_i) + (x+1) * R(D_i)$) **then**
    $I_i = MI(I_i, I_{i+1})$
  end if
end for
while ($|IL(D_i)| > CP(D_i)$) **do**

  $Grad(D_i) = Grad(D_i/2)$
  $R(D_i) = (Max(D_i) - Min(D_i))/Grad(D_i)$
  for $i = 1to|IL(D_i)|$ **do**

    if ($\exists x < Grad(D_i)$**and**$I_i.v_{min} \geq Min(D_i)+x*R(D_i)$**and**$I_i.v_{max} < Min(D_i)+(x+1)*R(D_j)$**and**$I_{i+1}.v_{min} \geq Min(D_i)+x*R(D_i)$**and**$I_{i+1}.v_{max} < Min(D_i)+(x+1) * R(D_i)$) **then**
      $I_i = MI(I_i, I_{i+1})$
    end if
  end for
end while
return IL(D_i) =0

3.1.6 数据生成算法

根据事先设置的梯度 $Grad(D_i)$, 来判断数据点 D_i 能否插入到设定的区间, 当合并完成之后，如果计算出 IL(D_i) 仍然是大于 CP(D_i), 则通过降低 Grad 的值来重新进行合并，通过循环检测条件来判断区间是否满足最终的要求。

算法 3.3 数据生成算法 GenData

输入： 通过区间合并算法生成的序列 IL

输出： 压缩后的最终数据

```
IL=[]
for (i = 1toN) do

    for (j = 1to|IL(D_i)|) do
        TL.push(IL(D_{i_j}, t_s))
        TL.push(IL(D_{i_j}, t_e))
    end for
end for
对 TL 中的时间标签从小到大按时间排序
删除 TL 中是多余时间标签
for (j = 1to|TL|) do
    对应输出 TL_i 所属的 IL(D_i) 区间的平均值
end for
return VOID =0
```

数据生成算法，首先将所有分量的时间标签放入到一个列表中，经过排序去重之后，通过遍历 IL 来生成新的数据，生成的新的数据结果采用的是计算出的平均值。

3.2 时序数据可视化

时序数据的预处理是指将时序数据转化为数据区间列表的过程，论文中定义的区间 R=< $v_s, v_e, v_{max}, v_{min}, v_{avg}, v_{var}$ >，其中 v_s 表示时序数据的开始值，v_e 表示时序数据的结束值，v_{max} 表示时序数据的最大值，v_{min} 表示时序数据的最小值，v_{avg} 表示时序数据计算出的平均值，v_{var} 表示时序数据的标准差。

对于给定的区间大小 H，时序数据预处理的过程是对连续的 H 个数据点按照顺序进行扫描，在扫描的过程中，计算这些数据的统计信息，我们将连续的 H 个数据点计算出它的最大值最小值平均值和标准差，我们选择区间的开始值，结束值，区间最大值，区间最小值，平均值和标准差作为统计信息是因为这些信息反映了一个区间内数据的基本特征，并且这些统计数据在区间合并的操作过程中具有良好的特性，我们可以根据相邻两个区间的值，直接计算出合并之

后的统计数据，这样就大大减少了算法的耗时。

3.2.1 时序数据预处理

算法 3.4 时序数据预处理算法 PreProcess

输入： 分隔区间的大小 H，原始数据序列 S

输出： 区间列表 R

```
1: COUNTER=0
2: for (d = 1to|S.length|) do
3:    if (COUNTER%H==0) then
4:       创建一个新的区间 I=<d,d,d,d,d,0>
5:    else
6:       num=(counter%H+1)
7:       I =< I.v_s, d, MAX(I.v_max, d), MIN(I.v_min, d), (I.v_avg * num + d)/(num +
         1), i.var + d * d >
8:       if COUNTER%H==H-1 then
9:          I =< I.v_s, d, MAX(I.v_max, d), MIN(I.v_min, d), (I.v_avg * num)/(num +
            1), sqrt(i.var − i.v_avg/H) >
10:         RL.push(I)
11:      end if
12:
13:   end if
14: end for
15: return RL =0
```

3.2.2 时序数据可视化

时序数据首先通过预处理之后，数据会被分隔为多个区间长度为 H 的数据区间，并存储在关系数据库当中，本论文中采取的是 SQLite，之后时序数据的可视化都可以以这些数据为基础来完成，当然考虑到原始数据量的规模很大，因此即使压缩为指定数据区间之后，有可能区间的个数还是很多，如果我们选取区间的总数为 100，并假设所有的数据量为 100 万，那么压缩过后仍然有万条数据，如果全部显示肯定是不现实的，那么换一个思路，假设我们将 H 设置为 10000，那么我们获取到的数据点数变为了 100，此时数据的局部信息丢失的

情况很有可能变得严重，在算法中，我们将合并的粒度设置为灵活可变，也就是最终的区间密度可以由用户指定，

定义初始化处理完的数据为元数据，并且由于默认的区间值为 H。对于每个区间我们定义 R=$< v_s, v_e, v_{max}, v_{min}, v_{avg}, v_{var} >$，设置数据的区间大小为 kH，那么以 kH 为区间大小的区间数据我们设置为:

$$R' =< v'_s, v'_e, v'_{max}, v'_{min}, v_{avg}, v'_{var} > \text{(3-7)}$$

那么需要合并的元数据区间中，第 i 个数据区间的数据为:

$$R=<v_{si}, v_{ei}, v_{maxi}, v_{mini}, v_{avgi}, v_{vari} > \text{(3-8)}$$

存在如下表达式:

$\alpha_s = v_s$

$\alpha_e = v_e$

$\alpha_{max} = max(v_{1max}, v_{2max}...v_{kmax})$

$\alpha_{min} = min(v_{1min}, v_{2min}...v_{kmax})$

$\alpha = 1/k * \sum_{n=1}^{N} v_{javg}$

$\alpha_{var}^2 * 1/k = \sum_{n=1}^{k} v_j avg^2 + \sum_{n=1}^{k} v_j var^2 + \sum_{n=1}^{k} v_j avg^2$

基于上面的公式我们可以计算出 kH 区间长度的数据区间特征值的方法，对于 R′ 中的开始值结束值最大值最小值平均值，计算都非常简洁，对于标准差的计算可通过对应的表达式计算出，通过将 α_{var}^2 与合并的 k 个元数据区间的方差之和做减法计算，消去原始数据即可以得到 R′ 的计算表达式。

第 4 章 程序设计与实现

4.1 开发环境介绍

(1) 开发环境：eclipse

(2) 持久化：SQLite

(3) 项目构建工具：Maven

(4) 语言：Java&JavaScript

4.2 需求分析

(1) 基于 B/S 完成整个项目。

(2) 实现数据压缩算法，并绘制相应的可视化界面。

(3) 实现压缩选项参数的可定制化。

(4) 实现灵活可扩展的接口设计，能实现持久化平台和可视化工具的自由切换。

(5) 保证代码的可维护性，尽可能多的合理准确的增加对相应设计模式的使用。

(6) 通过代码实现，验证算法的可行性，并对算法给出改进。

4.3 时序数据模拟设计

4.3.1 时序数据模拟器接口

业务场景中时序数据来自于物理数据，这些数据具有真实的物理特性，在压缩过程中更能体现出压缩算法的效果。在本论文的实现过程中我们采用了相关的模拟时序算法来生成一些时序数据，这些数据能较好的达到压缩的基本需求。对于一个时序数据模拟器接口，设计时最需要考虑的就是时序数据的接口的合理性，好的时序数据接口能以很少的更改，直接切换到真实的时序数据来源，同时还要降低对我们代码的侵入性，因此一个时序数据接口仅仅包含产生

时序数据，时序数据开始和停止三个方法，任何时序数据接口的实现都必须完全实现这三个方法。我们设计的接口显示如下：

```
public interface DataGen{
  public ArrayList<?> getGenData();
  public void run();
  public void stop();
}
```

注：DataGen 接口含有一个模拟器的所有方法，一个模拟器的实现必须至少含有这些方法。

4.3.2 时序数据模拟器工厂

用户在时序数据压缩的过程中，首先会运行时序数据模拟器，但是用户完全不用知道时序数据模拟器是如何产生的，此时我们引入了时序数据模拟器的抽象工厂，该工厂最大限度的隐藏了时序数据模拟器。并实现了更高程度的代码分离。

```
public class DataGenFactory{
  public DataGen createDataGen();
}
```

注：DataGenFactory 工厂提供出一个时序数据模拟器实例，调用者只需要查看 DataGen 的接口代码即可，不用关心 DataGen 的具体实现。

4.3.3 持久化 JDBC 链接

```
public abstract class Instance{
  public void connection();
  public void startTransaction();
  public void insert();
  public void delete();
}
```

注：时序数据持久化类为抽象类，必须提供事务操作。

4.3.4 时序数据模拟实现

```
public class SeriesDataGen implements DataGen{
  @override
  public ArrayList<?> getGenData();
  @override
```

```
  public void run();
  @override
  public void stop() ;

  private ArrayList arrayList;

  pivate createdate();//该方法产生对应的数据

  public static getDataGen(){//此处使用了一个单例模式
    if(dataGen==null)
      dataGen=new DataGen()

    return dataGen;
  }

  private static DataGen dataGen;
}
```

注：由于篇幅限制，我们给出了时序数据的产生接口设计，同时我们在上述过程中使用了工厂和单例的设计模式。

4.4 数据压缩处理

4.4.1 时序数据压缩接口

时序数据压缩的整个过程，可以分为三个阶段，其中包括时序数据的区间产生，时序数据的区间合并，还有时序数据的最终生成这三个过程，由于这三个过程紧密结合，具有较高程度的耦合性，没有其它的需求需要单独调用这三个方法，因此我们通过一个 process 来完成这所有的操作，并且我们将这三个处理类设计为受保护类型，因此他们的可见性，都停留在自己的包命名空间之内。

```
public class DataCompress{
  private process(ArrayList<?> dataarray){

  }
  public void CompressInterval();//此方法为直接调用方法

  private void IntervalMerage();//区间合并算法

  private void getGenData();//数据生成算法
```

```
  public void react();//此方法为注册方法

}
```

注：此类包含了一个数据处理的单个过程。

4.4.2 时序数据处理中心

```
public class processcore{
  private List<DataCompress> list;

  public void register(DataCompress unit);

  private listIterator;
}
```

注：此类包含了各个处理类的注册中心，当 processCore 接受到特定消息，包括 stop，suspend 等消息，就可以调用相应的 react 方法。这用到了迭代器和监听器模式。

4.4.3 可视化的绘图

盒图是在 1977 年由美国的统计学家约翰·图基 (John Tukey) 发明的。它由五个数值点组成：最小值 (min)，下四分位数 (Q1)，中位数 (median)，上四分位数 (Q3)，最大值，结合 K 线图。我们实现了这两者特点的结合，改进后的 K 线图显示如下。4-1:

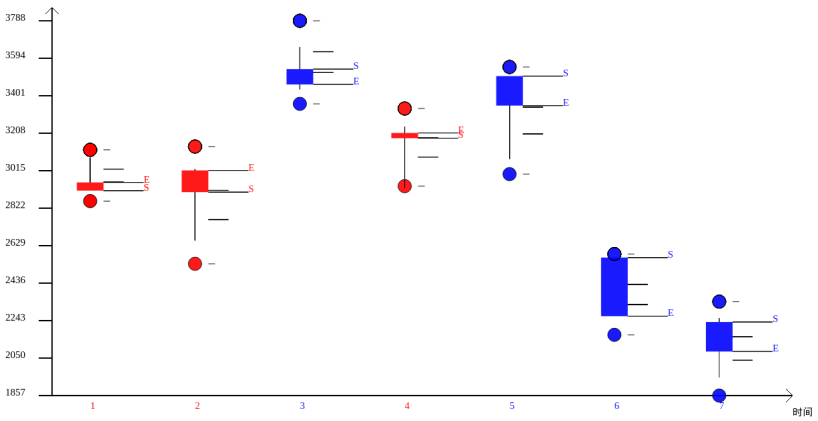

图 4-1 时序输入数据

第 5 章 结果分析

我们通过第三章给出的核心算法，以及第四章的需求和设计，来对时序数据压缩的情况进行模拟与分析。

5.1 示例说明

通过第四章的描述我们知道，压缩模拟的过程中，我们先通过模拟时序数据发生器，来生成时序数据，原始数据的产生时间间隔为 10ms，当原始数据生成以后我们对每一个区间按照区间分隔算法来进行数据区间的分割，当区间分割完毕之后，我们通过数据区间的合并算法对那些数据点过密也就是区间内的数据点数超过限制的区间进行合并，然后我们通过数据生成算法来生成压缩后的数据。最后我们可以通过基于浏览器来显示我们的可视化数据。

5.2 模拟过程

5.2.1 时序数据模拟产生

在程序的实际实现过程中，数据的维数接近 100，此处由于篇幅限制，我们将维数降低以使结果更加明显，程序运行的过程如图所示，5-1此处我们将输出数据重定向到了活动窗口。

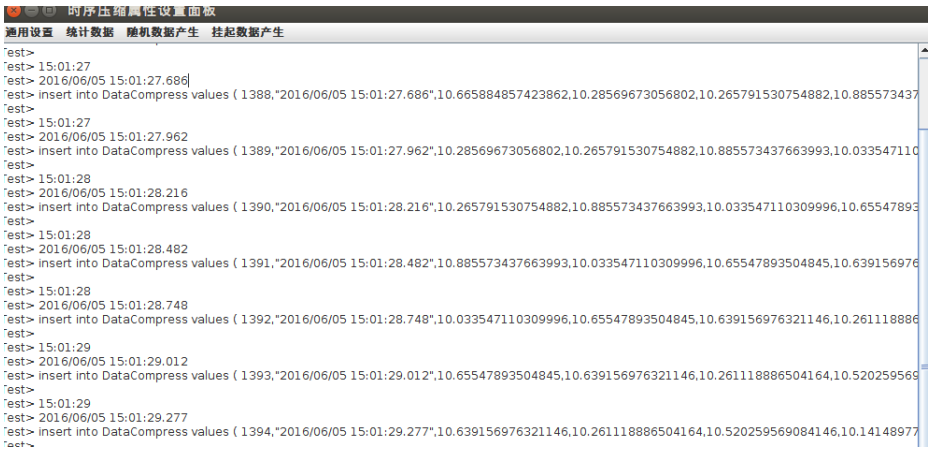

图 5-1 模拟时序数据发生器

如图对应的原始表如下：

T	D_1	D_2	D_3	D_4	D_5	D_6
1	10.6	10.2	10.2	10.8	10.8	10.8
2	10.2	10.2	10.8	10.0	10.0	10.0
3	10.2	10.2	10.8	10.0	10.0	10.0
4	10.8	10.0	10.0	10.1	10.0	10.0
5	10.8	10.0	10.0	10.1	10.1	10.2
6	10.7	10.2	10.2	10.2	10.0	10.2
7	10.9	10.2	10.3	10.2	10.3	11.0

5.2.2 时序数据区间分隔

通过原始数据的每一列我们进行区间划分，划分的规则如下：

```
D1  [1,1]  [2,3]  [4,7]
D2  [1,7]
D3  [1,1]  [2,3]  [4,7]
D4  [1,1]  [2,7]
D5  [1,1]  [2,7]
D6  [1,1]  [2,7]
```

5.2.3 区间合并与数据生成

由时序数据区间分隔算法产生的区间点包括 [1,1],[2,3],[4,7], 由这些时间点的生成的区间如下表所示：

T	D_1	D_2	D_3	D_4	D_5	D_6
1	10.6	10.2	10.2	10.8	10.8	10.8
2	10.2	10.2	10.8	10.0	10.0	10.0
3	10.8	10.1	10.8	10.1	10.1	10.1

5.2.4 可视化示例

时序数据通过可视化后的显示如下5-2，从图中我们可以观察到统计数据的开始值，结束值，最大值，最小值，以及分布情况。

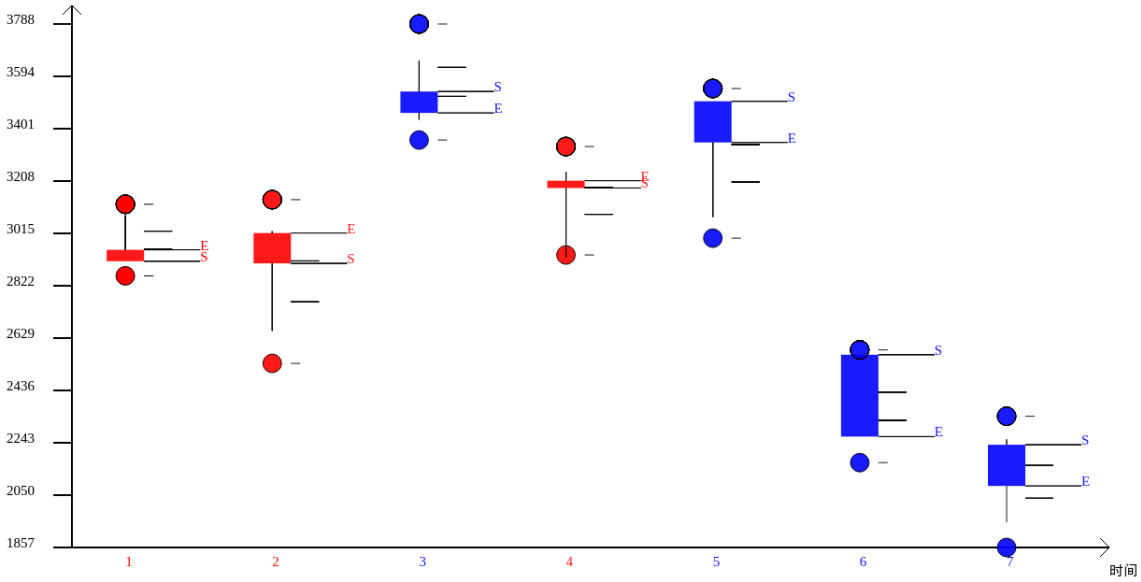

图 5-2 模拟时序数据发生器

图中包含以下有效信息：

(1) 红颜色方块代表开始值大于结束值，蓝色方块为开始值小于结束值。

(2) 开始值和结束值被实体方块包围。

(3) 次短线代表 3/4 位数，和 1/4 位数。

(4) 最短线代表最大值和最小值。

5.2.5 压缩结果分析

通过上述的过程我们可以看出，原始数据的时间点数为 6，而压缩处理过后产生的时间点数为 3，这在存储的过程中减少了一半的存储量，这是在我们

的波动误差约为 0.5 所得出的结论，如果我们的误差设置为 1，那么压缩效果可以达到 1/6，由此可见，在条件允许的情况下，这能节省大量的存储空间。

第 6 章 总结

从全文的整体结构来看, 我们完成了高维时序数据压缩到可视化的一整套流程。首先我们介绍了时序数据的相关特征, 时序数据压缩的相关历史研究, 然后我们就时序数据压缩所需要的工具与算法进行了介绍, 在程序设计与实现部分我们给出了基于 Java 相关平台的设计与实现思路。在实验结果部分我们给出了压缩实例和压缩效果的展示。

从压缩效果来看, 本论文提供了一种切实可行的高维时序数据压缩的可选方案, 从可视化的角度来看我们又提供了对于不同可视化工具的支持设计。我们的原则是基于可扩展的接口设计, 同时强调设计的可重复使用性。总体来说论文中的整体设计与实现是对高维时序数据与可视化的一次有益的探索。对于工程中的使用者来说我们的设计思路也有相当的参考价值。

6.1 未来工作

针对高维时序数据压缩和可视化这一问题, 以及我们的研究结果, 未来还有许多进一步的工作可以继续开展：

(1) 在时序数据的压缩算法上我们应该还有很多需要改进的部分

(2) 时序数据压缩的计算量异常庞大, 我们可以尝试基于分布式系统来提高我们的计算速度

(3) 如何构造一个更为准确的时序数据产生模型

(4) 对于压缩梯度的设置我们是否能够设计的更为准确

(5) 在程序设计部分离工程的标准还有一定的差距, 我们应该以更为合理和优秀的设计为目标。

致谢

近半年的学习和科研工作不仅使我的知识结构和科研能力上了一个新台阶更重要的是各方面的素质得到了提高。而这一切都要归功于刘庆晖老师的深切教诲与热情鼓励。值此论文顺利完成之际我首先要向我尊敬的导师刘庆晖老师表达深深的敬意和无以言表的感谢。同时感谢计卫星老师在我学习期间给予的帮助。感谢和我一起工作的沈卓佳同学, 张露露学姐。沈卓佳灵活考虑问题的方式严谨的解决问题的态度张露露学姐扎实的专业知识功底认真的科研态度都给我留下了深刻的印象。感谢和我同一实验室的高志伟学长。没有他们无私的帮助我是无法完成论文工作的。感谢我的室友王天舒、沈卓佳、彭逸云和他们一起度过了这段美好时光是难以忘记的。最后深深的感谢呵护我成长的父母。每当我遇到困难的时候父母总是第一个给我鼓励的人。回顾 20 多年来走过的路每一个脚印都浸满着他们无私的关爱和谆谆教诲 10 年的在外求学之路寄托着父母对我的殷切期望。他们在精神上和物质上的无私支持坚定了我追求人生理想的信念。父母的爱是天下最无私的最宽厚的爱。大恩无以言报惟有以永无止境的奋斗期待将来辉煌的事业让父母为之骄傲。我亦相信自己能达到目标。

参考文献

[1] 关云鹏，杨静. 高维时序数据的相似搜索 [J]. 贵州大学学报 (自然科学版), 2006/23(1):44-50.DOI:10.3959/j.jssn.1000-5269.2006.01.010

[2] 王成山, 王继东. 基于能量阈值和自适应算术编码的数据压缩方法 [J], 电力系统自动化 2004/28(24):56-60

[3] 黄勇，吕涛. 基于线性插值函数的遥测数据压缩 [J]，计算机工程与应用，2004,38(1640-42

[4] 段培勇，张枚，汤同奎等 SDT 算法及其在局域控制网络中压缩过程数据的应用 [J]，信息与控制，2002,31(2):132-135.DOI:10.3969/j.jssn.1002-0411.2002.02.008

[5] 刘玉葆，李启睿一种基于压缩策略的高维空间子空间 skyline 查询算法 [J]，计算机研究与发展，2013(s1):101-108

[6] 游文杰，吉国力，袁明顺，高维少样本数据的特征压缩，计算机工程与应用，2006/17(1):49-50

[7] Mah R S H ,et al.Process trending with piecewise linear smoothing .Comput Chem Engng,1995,19(2):129-137

[8] Gray R M Vector quantizatio .IEEE ASAP Magazine,Apr,1984,4-29

[9] Macgregor JF.Statisti cal process control of multivariate process IFAC AD-CHEM,Kyoto Japan 1994

[10] ABflag.j.Friegel.HP similary search ontime series based on threshold queries in: Proceedins of the 10thinternationl COnferece on Extending DatabaseTechnolog pp.276-294

[11] Abonyi clustering for fuzzing segmentation of multivariate time-series .Fuzzy sets and Systems Data Mining SSpecial lssue 149:39-56

[12] Oliveira PCF Ahmd K 2004 summarization fo multiodal information in: Proceedings of the Fourth Internationa COnference on Language Resources and Evalutation vol 3.pp 1049-1052

[13] ENrique EH 1999 Qualitative object description:inital reports of the exploration of the frontier. Proceedings of Joint EUROFUSE-SIC99 Interational Conferecne pp:489-490

[14] Sidiropouls Johnson T.Jagadish H.V Faloutsos C Biliris A 2000 online data mining for co-evolving time sequeces in Proceedings of the 16th IEEE International Conference on Data Engineering pp:201-208

[15] Yin Yang Q 2006 Intergratiing Hidden Markov Models and spectral analysis for sensory time series clustering in: Proceedings of the Fifth IEEE international-Conference on Data Mining pp:06-513

[16] Yand.K.Yoon H.Shahabi C. 2005 Clever: a feature subset selection technique for multivariate time seies in procedings of the Ninth Pacific-Asia COnference on Knowledge Discovery and Data Mining pp:516-522

[17] Yang K .Sha on the stationarity of multivariate time seies for correlation-based data analysis in proceedings of the fifth IEEE international conference on Data Mining pp:805-808

附录 A 时序模拟和时序压缩

A.1 GenData.java

```
package cn.BITCS.DataBase;

import java.io.FileNotFoundException;
import java.sql.SQLException;
import java.util.ArrayList;
import java.util.Random;

public class DataGen
{
  private ArrayList<String> Array;

  public void Init()
  {
    int i=0;
    Array=new ArrayList<String> ();

    Array.add(0, "0");
    Array.add(1, "null");

    for(i=2;i<=5;i++)
    {
      Array.add(i, GenerateDouble().toString());
    }

    for(i=6;i<=7;i++)
    {
      Array.add(i, GenerateBoolean().toString());
    }

    Array.add(8, GenerateDouble().toString());

    for(i=9;i<=11;i++)
    {
      Array.add(i, GenerateBoolean().toString());
    }

    for(i=12;i<=51;i++)
```

```
  {
    Array.add(8, GenerateDouble().toString());
  }

  Array.add(52, GenerateInteger().toString());

  for(i=53;i<=57;i++)
  {
    Array.add(8, GenerateDouble().toString());
  }

  for(i=58;i<=63;i++)
  {
    Array.add(i, GenerateBoolean().toString());
  }

  for(i=64;i<=77;i++)
  {
    Array.add(i, GenerateDouble().toString());
  }

  Array.add(78, GenerateBoolean().toString());

  for(i=79;i<=87;i++)
  {
    Array.add(i, GenerateDouble().toString());
  }

}

private int i=0;

public void Insert() throws SQLException, FileNotFoundException
{
  InsertInto PreInsert=new InsertInto();
  PreInsert.insert(Array);
}

private Double GenerateDouble()
{
  Random RandomNumber=new Random(1000);
  Double Number=RandomNumber.nextDouble()*1000;
  if(Number%1000>=970)
```

```
    {
      Number=Number%5+10.0;
    }
    else
    {
      Number=10.0+Number%1.0;
    }

    return Number;

  }

  private Integer GenerateBoolean()
  {
    Random RandomNumber=new Random();
    Integer Number=RandomNumber.nextInt();
    if(Number%1000>990)
      return 1;
    else return 0;
  }

  private Integer GenerateInteger()
  {
    Random RandomNumber=new Random();
    Integer Number=RandomNumber.nextInt();
    return Number%5;

  }
}
```

A.2 Instance.java

```
package cn.BITCS.DataBase;
import java.sql.*;

public class Instance{
  public static void main(String[] args) {
      try {
        Class.forName("org.sqlite.JDBC");
        Connection conn =DriverManager.getConnection("jdbc:sqlite:info");
        Statement stat = conn.createStatement();
        stat.executeUpdate("drop table if exists people;");
        stat.executeUpdate("create table people (name, occupation);");
        PreparedStatement prep = conn.prepareStatement("insert into people values (?,

        prep.setString(1, "Gandhi");
        prep.setString(2, "politics");
        prep.addBatch();
        prep.setString(1, "Turing");
        prep.setString(2, "computers");
        prep.addBatch();
        prep.setString(1, "Wittgenstein");
        prep.setString(2, "smartypants");
        prep.addBatch();

        conn.setAutoCommit(false);
        prep.executeBatch();
        conn.setAutoCommit(true);

        ResultSet rs = stat.executeQuery("select * from people;");
        while (rs.next()) {
          System.out.println("name = " + rs.getString("name"));
          System.out.println("job = " + rs.getString("occupation"));
        }
        rs.close();
        conn.close();
      } catch (ClassNotFoundException e) {
        // TODO Auto-generated catch block
        e.printStackTrace();
      } catch (SQLException e) {
        // TODO Auto-generated catch block
```

```
        e.printStackTrace();
      }
    }
  }
```

A.3 Compress.java

```
package cn.BITCS.SeqDataPre;

import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Statement;
import java.util.ArrayList;

public class Compress{

  public void VictorSet()
  {

  }

  public void Scanner(int i2) throws ClassNotFoundException, SQLException
  {
    Class.forName("org.sqlite.JDBC");
    Connection conn =DriverManager.getConnection("jdbc:sqlite:info");
    Statement stat = conn.createStatement();

       ResultSet rs = stat.executeQuery("select count(*) from DataCompress" );



       if (rs.next())
         {
              Count = rs.getInt(1);
         }

       String SqlStatement="select * from DataCpress";

       int i=0;
       for(i=2;i<=88;i++)
       {
```

```
      stat.executeUpdate(SqlStatement);

      rs=stat.executeQuery(SqlStatement);

      Area area=new Area();

      area.setStart(0);

      while(rs.next())
      {
        if(rs.getDouble(2)>11.0)
        {
          area.setOver(rs.getRow());
          Array.add(area);
          area=new Area();
        }

        if(new Integer(area.getMax())<new Integer(rs.getString(i)))
        {
          area.setMax(rs.getString(i));
        }

        if(new Integer(area.getMax())<new Integer(rs.getString(i)));
        {
          area.setMax(rs.getString(i));
        }

        if(new Integer(area.getMin())>new Integer(rs.getString(i)))
        {
          area.setMin(rs.getString(i));
        }
      }
     }

}

public void insert() throws ClassNotFoundException, SQLException
{
  Class.forName("org.sqlite.JDBC");
  Connection conn =DriverManager.getConnection("jdbc:sqlite:info");
  Statement stat = conn.createStatement();
  int i=0;
```

```
    while(Array.get(i) != null)
    {
      String Data=Array.get(i).getAverage();
      stat.executeQuery("insert into data"+Data);
    }


  }

  private ArrayList <Area> Array;

  private Integer Count;

}
```

A.4 TextAreaOutputSteamTest.java

```
package cn.BITCS.SeqDataPre;
import java.awt.BorderLayout;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.io.PrintStream;
import javax.swing.*;

@SuppressWarnings("serial")
public class TextAreaOutputStreamTest extends JPanel {

   private JTextArea textArea = new JTextArea(15, 30);
   private TextAreaOutputStream taOutputStream = new TextAreaOutputStream(
         textArea, "Test");

   public TextAreaOutputStreamTest() {
      setLayout(new BorderLayout());
      add(new JScrollPane(textArea, JScrollPane.VERTICAL_SCROLLBAR_ALWAYS,
            JScrollPane.HORIZONTAL_SCROLLBAR_NEVER));
      System.setOut(new PrintStream(taOutputStream));

      int timerDelay = 1000;
      new Timer(timerDelay , new ActionListener() {
         int count = 0;
```

```
            @Override
            public void actionPerformed(ActionEvent arg0) {

               // though this outputs via System.out.println, it actually displays
               // in the JTextArea:
               System.out.println("Count is now: " + count + " seconds");
               count++;
            }
         }).start();
      }

      private static void createAndShowGui() {
         JFrame frame = new JFrame("Test");
         frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
         frame.getContentPane().add(new TextAreaOutputStreamTest());
         frame.pack();
         frame.setLocationRelativeTo(null);
         frame.setVisible(true);
      }

      public static void main(String[] args) {
         SwingUtilities.invokeLater(new Runnable() {
            public void run() {
               createAndShowGui();
            }
         });
      }
}
```

附录 B echarts 绘图示例代码

B.1 candlestick.html

```
<!DOCTYPE html>
<html>
<head>
  <meta charset="utf-8">
  <title>时序数据压缩</title>
  <script type="text/javascript" src="jquery-1.12.3.min.js"></script>
</head>

<body>
    <br/>
    <canvas id="maincanvas" height="650px" width="1500px"></canvas>
    <script type="text/javascript">
      var canvas=document.getElementById("maincanvas");
      var ctx=canvas.getContext('2d');
      ctx.font="15px Georgia";
      ctx.fillText("时间",1210,610);


      function paintline(start_x,start_y,end_x,end_y){
        ctx.moveTo(start_x,start_y);
          ctx.lineTo(end_x,end_y);
          ctx.stroke();
      }

      function spalit_data(argument){
        var categrayData=[];
        var values=[];
        for(var i=0;i<argument.length;i++){
          categrayData.push(i+1);
          values.push(argument[i]);
        }

        return {categrayData:categrayData,
            values:values
          };
      }
```

```
//x axis
paintline(100,580,1210,580);

//y axis
paintline(100,580,100,0);

//x axis arrow
paintline(1210,580,1200,570);

paintline(1210,580,1200,590);

//y axis arrow
paintline(100,0,90,10);

paintline(100,0,110,10);


function load_fromdata() {
    var text=document.getElementById("json").value;
    var jsondata=JSON.parse(text);

    data0=spalit_data(jsondata);

    var array=[];
    var ystart=0;
    var yend=0;

    for(var i=0;i<data0.values.length;i++){
      var length=data0.values[i].length;
      var start_value=data0.values[i][0];
      var end_value=data0.values[i][length-1];
      //document.write(data0.values[i]+'#');
      data0.values[i].sort(function(n1,n2){return n1-n2;});
      //document.write(data0.values[i]+'@');
      var min=data0.values[i][0];
      var max=data0.values[i][length-1];
      var Q1=data0.values[i][Math.floor(length/4)];
      var Media=data0.values[i][Math.floor(length/2)];
      var Q3=data0.values[i][Math.floor(length*3/4)];
      var IRQ=Q3-Q1;
      var count_max=Math.floor(Q3+1.5*IRQ);
      var count_min=Math.floor(Q1-1.5*IRQ);
```

```
  if(count_min<0)
    count_min=0;

  var ele={
    start_value:start_value,
    end_value:end_value,
    Q1:Q1,
    Q3:Q3,
    Media:Media,
    max:max,
    min:min,
    count_min:count_min,
    count_max:count_max,
    array:data0.values[i]
  }

  if(count_max>yend)
    yend=count_max;

  if(i==0&&count_min>0)
    ystart=count_min;

  if(i!=0&&count_min<ystart)
    ystart=count_min;

  array.push(ele);
}


var distance=(yend-ystart)/10;

for(var i=0;i<=10;i++){
  paintline(100,580-56*i,80,580-56*i);
  ctx.fillText(Math.floor(ystart+distance*i),30,580-56*i);
}

for(var i=1;i<=data0.values.length;i++){

  var color='#FF0000';

  if(array[i-1].start_value>array[i-1].end_value)
    color='#0000FF';
```

```
var distancex=1100/7;
ctx.fillStyle=color;
ctx.fillText(data0.categrayData[i-1],distancex*i,600);
var xcood=distancex*i;

//var count_up_Q3=0;
//var count_media=0;
//var count_down_Q1=0;

//for(var j=0;j<array[i-1].array.length;j++){
//  if(array[i-1].array[j]>array[i-1].Q3&&array[i-1].array[j]<=array[i-1]
//    count_up_Q3++;
//  else if(array[i-1].array[j]<=array[i-1].Q3&&array[i-1].array[j]>array
//    count_media++;
//  else if(array[i-1].array[j]<=array[i-1].Q1&&array[i-1].array[j]>array
//    count_down_Q1++;
  /*else{
    var ycood_circle=580-(array[i-1].array[j]-ystart)/distance*56;
    if(array[i-1].array[j]>yend)
      ycood_circle=580-(yend-ystart)/distance*56;
    ctx.fillStyle=color;
    ctx.beginPath();
    //document.write(array[i-1].array[j]+'@'+ystart+'@'+xcood+'@'+ycood_ci
    ctx.arc(xcood,ycood_circle,10,0,2*Math.PI);
    ctx.fill();
  }
  */
//}

//var counter=count_up_Q3+count_media+count_down_Q1;


var ycood_max=580-(array[i-1].max-ystart)/distance*56;
//paintline(xcood+20,ycood_max,xcood+60,ycood_max);

var ycood_min=580-(array[i-1].min-ystart)/distance*56;
//paintline(xcood+20,ycood_min,xcood+60,ycood_min);


var ycood_start=580-(array[i-1].start_value-ystart)/distance*56;
//paintline(xcood+20,ycood_start,xcood+80,ycood_start);
//ctx.fillText('S',xcood+80,ycood_start);
```

```
var ycood_end=580-(array[i-1].end_value-ystart)/distance*56;
//paintline(xcood+20,ycood_end,xcood+80,ycood_end);
//ctx.fillText('E',xcood+80,ycood_end);


//document.write(ycood_min+'#');
//document.write(ycood_max+'#');
//document.write(ycood_start+'#');
//document.write(ycood_end+'#@@@@@@@');

var ycood_count_min=580-(array[i-1].count_min-ystart)/distance*56;
//paintline(xcood+20,ycood_count_min,xcood+30,ycood_count_min);
for(var j=0;j<8;j++){
  if(j%2==0)
    paintline(xcood+20+j*5,ycood_count_min,xcood+25+j*5,ycood_count_min);
}
//if(ycood_count_min>ycood_start&&ycood_count_min<ycood_end){
//}
//else if(ycood_count_min<ycood_start&&ycood_count_min>ycood_end){
//}
//else{

//  ctx.fillStyle=color;
//  ctx.beginPath();
//  ctx.arc(xcood,ycood_count_min,10,0,2*Math.PI);
//  ctx.fill();
//}


var ycood_count_max=580-(array[i-1].count_max-ystart)/distance*56;
for(var j=0;j<8;j++){
  if(j%2==0)
    paintline(xcood+20+j*5,ycood_count_max,xcood+25+j*5,ycood_count_max);
}
//paintline(xcood+20,ycood_count_max,xcood+30,ycood_count_max);
//if(ycood_count_max>ycood_start&&ycood_count_max<ycood_end){

//}
//else if(ycood_count_max<ycood_start&&ycood_count_max>ycood_end){
//}
//else{

//  ctx.fillStyle=color;
```

```
//  ctx.beginPath();
//  ctx.arc(xcood,ycood_count_max,10,0,2*Math.PI);
//  ctx.fill();
//}


var ycood_Q3=580-(array[i-1].Q3-ystart)/distance*56;
paintline(xcood+20,ycood_Q3,xcood+50,ycood_Q3);
//ctx.fillText('Q3',xcood+50,ycood_Q3);


var ycood_Q1=580-(array[i-1].Q1-ystart)/distance*56;
paintline(xcood+20,ycood_Q1,xcood+50,ycood_Q1);
//ctx.fillText('Q1',xcood+50,ycood_Q1)




/*ctx.globalAlpha=count_up_Q3/counter*0.8;
ctx.rect(xcood-20,ycood_count_max,40,ycood_Q3-ycood_count_max);

ctx.fillStyle=color;

ctx.fill();


ctx.globalAlpha=count_media/counter*0.8;

ctx.rect(xcood-20,ycood_Q3,40,ycood_Q1-ycood_Q3);
ctx.fillStyle=color;

ctx.fill();

ctx.globalAlpha=count_down_Q1/counter*0.8;
ctx.rect(xcood-20,ycood_Q1,40,ycood_count_min-ycood_Q1);

ctx.fillStyle=color;

ctx.fill();
*/
ctx.globalAlpha=0.9;
ctx.fillStyle=color;
```

```
            if(ycood_start<=10)
              ycood_start=20;
            if(ycood_end<=10)
              ycood_end=20;

            if(array[i-1].start_value>array[i-1].end_value)
              ctx.fillRect(xcood-20,ycood_start,40,ycood_end-ycood_start);
            else
              ctx.fillRect(xcood-20,ycood_end,40,ycood_start-ycood_end);


            if(array[i-1].start_value>array[i-1].end_value){
              paintline(xcood,ycood_max,xcood,ycood_start);
              paintline(xcood,ycood_end,xcood,ycood_min);
            }
            else{
              paintline(xcood,ycood_max,xcood,ycood_end);
              paintline(xcood,ycood_start,xcood,ycood_min);
            }



          }

        }
    </script>
    <br/>
    <br/>
    </body>
    <input type="text" id="json" style="height: 100px;width: 800px"></input>
    <input type="button" onclick="load_fromdata()">更新显示</input>
</body>
</html>
```