南京大学

研 究 生 毕 业 论 文

（申请硕士专业学位）

论　文　题　目　基于矢量瓦片和优先点树的相似轨迹检索和可视化服务的设计和实

作　者　姓　名　韩淳

学　科、专　业　软件工程

研　究　方　向　软件工程

指　导　教　师　刘海涛 讲师

年 月 日

学　　号：**MF1732038**

论文答辩日期：**年 月 日**

指 导 教 师：　　**（签字）**

The Design and Implementation of Similar Path Search and Visualization Service based on vector-tile and vp-tree

By

Han Chun

Supervised by

Advisor Title**Si Li**

A Thesis
Submitted to the Software Institute
and the Graduate School
of Nanjing University
in Partial Fulfillment of the Requirements
for the Degree of
Master of Engineering

Software Institute

May 2019

南京大学研究生毕业论文中文摘要首页用纸

毕业论文题目：基于矢量瓦片和优先点树的相似轨迹检索和可视化服务的设计和实

软件工程 专业 2017 级硕士生姓名： 韩淳

指导教师（姓名、职称）： 刘海涛 讲师

摘 要

地理数据，是直接或间接关联着相对于地球的某个地点的数据，是表示地理位置、分布特点的自然现象和社会现象的诸要素数据。地理数据包括轨迹数据和瓦片数据。其中轨迹数据指的是一个物体的运动轨迹在空间中经过的点的集合。瓦片数据指的是用于展示的地图数据。在地理数据用户的业务中，一个非常有价值的场景就是犯罪同伙分析，通过框选选定目标嫌疑人轨迹，再通过检索与目标嫌疑人活动轨迹相似的其他人员的轨迹，从而找到目标嫌疑人的潜在同伙。针对以上这样的使用场景，本文构建了一个能够提供地图路径展示和相似路径检索的服务。本文主要关注两部分，第一部分是如何建立高效的轨迹索引结构，以提高良好的相似性检索的性能。第二部分是如何构建一个通用的瓦片数据服务，用来高效地提供地图数据。

该数据服务的主体思路是分别构建轨迹数据和地图数据的存储检索结构，然后分别从轨迹数据服务和地图数据服务中提取相关数据，再利用前端库整合两份数据共同显示在浏览器上，从而让用户方便快捷地看到轨迹检索结果。

文章的主要工作就是介绍以上两个数据服务的设计和实现。首先，本文介绍了目前在地理数据服务中主要流行的技术，包括通用的技术标准和本文的关键数据结构优先点树和矢量瓦片，并分析了这两种结构在地理数据服务的作用和优势。在后续章节，通过整个系统的需求以及检索端的流程来详细介绍这两种结构在系统中的角色。接着介绍了整个服务的设计架构，各个模块的职责划分和各个模块的详细设计和实现。在该过程中还讨论并分析了系统中出现的一些性能以及模型上的问题，提供了更好的解决方案，并且通过具体的性能数据来说明解决方案的提升点。最后，详细分析了整个服务对地理轨迹数据检索方面的效果提升。

地理轨迹数据服务，（Geographic trajectory data service, 以下简称GTDS)，通过构建独立的索引结构和通用的瓦片数据服务，实现了相似轨迹的检索和展示功能，解决了用户应用中一些难题，提高了系统的可用性。文章的最后部分，通过总结和展望，对该技术以及应用前景进行了一些分析。

关键词： 地理数据，轨迹数据，瓦片数据，优先点树,矢量瓦片，轨迹索引结构，轨迹相似检索，可视化

南京大学研究生毕业论文英文摘要首页用纸

THESIS: The Design and Implementation of Similar Path Search and Visualization Service based on vector-tile and vp-tree

SPECIALIZATION: Software Engineering

POSTGRADUATE: Han Chun

MENTOR: Advisor TitleSi Li

Abstract

Geographic data is data that is directly or indirectly related to a certain location on the earth. It is the data of natural phenomena and social phenomena that represent geographic location and distribution characteristics. Geographic data includes trajectory data and tile data. Where trajectory data refers to a collection of points through which an object's motion trajectory passes in space. Tile data refers to the map data used for display. In the business of geographic data users, a very valuable scenario is the analysis of criminal accomplices. By selecting the target suspect's trajectory, and by searching the trajectory of other people similar to the target suspect's trajectory, the target suspect is found. Potential associates. In view of the above usage scenarios, this paper constructs a service that can provide map path display and similar path retrieval. This article focuses on two parts. The first part is how to build an efficient trajectory index structure to improve the performance of good similarity retrieval. The second part is how to build a generic tile data service to efficiently provide map data. The main idea of the data service is to separately construct the storage retrieval structure of the trajectory data and the map data , and then extract the relevant data from the trajectory data service and the map data service respectively, and then integrate the two pieces of data into the browser by using the front-end library, thereby Let users see the track search results quickly and easily. The main work of the article is to introduce the design and implementation of the above two data services. First of all, this paper introduces the current popular technologies in geographic data services, including common technical standards and the key data structures of this paper, vp-tree and vector tile, and analyzes the roles and advantages of these two structures in geographic data services. In the following chapters, the roles of the two systems in the system are described in

detail through the requirements of the entire system and the processes at the search end. Then it introduces the design structure of the whole service, the division of duties of each module and the detailed design and implementation of each module. In the process, some performance and model problems appearing in the system are also discussed and analyzed, which provides a better solution and shows the solution's lifting point through specific performance data. Finally, the effect of the whole service on the retrieval of geographic trajectory data is analyzed in detail. The geographic trajectory data service realizes the retrieval and display function of similar trajectories by constructing an independent index structure and a common tile data service, solving some problems in the user application and improving the availability of the system. In the last part of the article, through the summary and outlook, some analysis of the technology and application prospects.

Keywords: English,Geographic data, trajectory data, tile data, vantage point tree, vector tile, trajectory index structure, trajectory similarity retrieval, visualization

目　录

表　格

插　　图

第一章　引言

1.1　项目背景

在移动互联网、卫星定位技术、LBS技术高速发展的背景下，无时无刻不在产生轨迹数据，轨迹数据包括交通数据、人类移动数据、动物迁移数据和自然现象轨迹数据等。海量的轨迹数据潜在性地暴露了个人的行为特征、兴趣爱好和社会关系等信息。[1]这种细节信息的暴露，很大程度上是由于轨迹数据本身存在位置特征和时空特征上的关联性。这种关联性是很多高价值商业场景的运行基础。而在所有这些的关联性中，最直观，最有利用价值的，就是轨迹的相似性。

与轨迹相似有关的高价值应用场景很多，例如基于轨迹相似的用户分类，交通路线预测，犯罪同伙分析等等。因此，一个稳定高效的相似轨迹检索系统是符合商业发展要求的必需产品。而要构建这样一个相似轨迹检索系统，必需考虑以下三个方面的问题。

第一，海量轨迹的存储。轨迹本身是具有时空特性的几何图形，而轨迹数据的量级一般都在千万级甚至亿级以上。这对于数据存储提出了很高的要求，传统的单机关系型数据库显然无法满足这一要求。

第二，检索的数据结构和检索行为的定义。由于数据量很大，传统的预处理，排序，过滤等方法即使发挥到最大效应，也很难提供让用户满意的检索性能。因此，必须为轨迹数据建立定制的索引结构，并根据这种索引结构定义检索行为，才能在检索性能满足商用需求。

第三，可视化运行。在当今大数据行业的发展背景下，数据可视化几乎是所有商业用户的共同需求。数据可视化很大程度上降低了系统的使用门槛，提高了系统的可用性，扩大了系统的适用范围。而在轨迹相似检索系统下，需要可视化的数据包括两部分。除了轨迹数据之外，地图数据也必须要实现可视化。否则的话，只有轨迹数据，没有其在对应地图上的状态显示，那轨迹本身失去了空间特性，退化为简单的几何图形，那么轨迹可视化本身也失去了意义。因此轨迹+地图的展示模式，才是一个完整的轨迹数据服务的可视化模式。

针对以上三个方面问题，本文设计并实现了基于优先点树和矢量瓦片的相似轨迹检索系统，为用户提供高效，稳定的相似检索服务。

1.2　国内外相关系统的发展概况

1.2.1　国内(外)轨迹数据系统发展概况

目前国内外轨迹相关系统所提供的轨迹分析功能着重于对速度变化和停靠点处理两方面，本文主要调查了百度鹰眼和ArcGis这两个系统。

百度鹰眼是一套集轨迹追踪、存储、运算、查询的完整轨迹开放服务，可帮助开发者管理多达100万人/车轨迹。[2]

百度鹰眼支持持续的轨迹追踪，鹰眼SDK可以实时地采集终端的地理位置，持续回传轨迹。其采集回传的频率一般在2s到5min这个区间内。

百度鹰眼支持轨迹存储，提供数据访问隔离和分布式存储，保证数据安全。其存储的内容包括坐标，速度，图片，视频和用户自定义字段。在数据存储量上，鹰眼目前支持100万终端，储存1年的轨迹数据。

百度鹰眼支持轨迹查询和展示，提供历史轨迹查询服务，开发者可以毫无延时地查询终端的实时位置，并回放轨迹。

百度鹰眼支持轨迹数据分析。目前已提供的轨迹分析包括驾驶行为分析（急速，超速判断），停留点分析（是否违法停车）等。

ArcGis是由ESRI公司开发的地理信息系统系列软件，其ArcGis1.0是世界上第一个现代意义上的GIS软件，第一个商品化的GIS软件。在轨迹数据服务方面ArcGis提供了路径图层创建，障碍创建，停靠点编辑，轨迹运动方向生成，运动轨迹展示，轨迹运动分析。其中轨迹运动分析包括所有停靠点最佳访问方式路径生成和展示。

1.2.2　国内（外）地图瓦片数据系统发展概况

TileServer-GL是一个针对矢量瓦片的开源地图服务器。它能够在服务器端使用MapBox GL内置引擎对矢量瓦片进行栅格化，进而为web应用和移动应用提供提供地图数据。它支持Mapbox GL JS,Android SDK,IOS SDK,Leaflet, OpenLayers, HighDPI/Retina, GIS via WMTS等众多前端库的数据调用[3]。

TileServer-GL不仅能够提供瓦片数据，还提供了基于Mapbox GL Style的地图渲染。用户只要提供了有效的Mapbox GL style文件，tileServer就能够按照指定风格渲染地图数据，并返回给浏览器或移动端。

尽管TileServer-GL对外服务有良好而良好的适应性，但是它在商业应用领域存在以下两方面明显的短板。

首先，它的数据是保存在mbtiles文件中，而mbtiles是sqlite数据库的一种文件格式。TileServer-GL强耦合了这种文件格式使得其对不同数据源的扩展性

几乎为零，面对那些数据保存在传统关系型数据库或是列数据库中的用户，TileServer-GL将无能为力。

其次，TileServer-GL只能提供瓦片读取服务，而不能提供地图数据的实时更新。而某些商业场景下，地图数据发生更新变化的可能性是非常大的。对于这种有更新要求的商业场景,TileServer也无法胜任。

1.3　本文的主要工作

本文设计和实现了基于优先点树和矢量瓦片的相似轨迹检索服务，其主要功能是在千万级别的轨迹数据量中，在规定时间内检索出与目标轨迹最相似的K条轨迹，并将这些轨迹可视化地展示在地图背景之下。主要工作有以下两个方面。

第一，设计和实现了ElasticSearch Geometry vp-tree这个索引结构，也就是优先点树结构。这个数据结构是原生ElasticSearch没有的。其主要功能是在Geometry数据之上，建立一个多分查找的树结构，以最大限度地做到搜索剪枝，将搜索时间控制在nlog(n)这个级别上。

第二，设计和实现了地图矢量瓦片服务Map Vector Tile Service(简称MVTS)。MVTS是一个提供了瓦片读取，检索，更新和风格渲染功能的矢量瓦片服务器，能够为OpenLayer,Leaflet，GIS via wmts等多个前端库提供瓦片数据。相比于开源的TileServer-GL，MVTS能够提供地图更新功能，并且具有良好的数据扩展性，提供多种数据存储方式的支持，能够实现与多种数据库的无缝功能对接。

1.4　本文的组织结构

本文的组织结构如下：

第一章引言部分。介绍了项目背景，国内外相关系统的研究现状。

第二章技术综述。介绍了项目中使用到的优先点树结构，ElasticSearch全文检索系统,lucene引擎，瓦片数据，nodejs express框架，MapBox标准等。

第三章系统需求分析和概要设计。通过需求列表展示了项目的具体需求，用例图介绍了分析需求的结果，并针对重要的，操作复杂的用例使用用例描述表的形式进行重点介绍。还对项目进行了概要设计，以组件图介绍整体架构，并介绍了各个组件的功能和作用。

第四章系统详细设计与实现。在概要设计的基础上，分别对相似轨迹检索服务和地图瓦片数据服务这两个模块进行详细设计和具体描述，以类图展示了类关系，并展示了关键部分代码。

第五章总结与展望。总结论文期间所做的工作，并就相似轨迹检索服务的未来方向作了进一步展望。

第二章　相关技术概念综述

2.1　优先点树

2.1.1　NN问题

Nearest Neighbour 问题，即最近邻居问题。指的是针对空间中的一个点集，定义一个距离函数d（这里的距离函数d包括但不仅限于欧几里得距离），那么对于一个给定的目标搜索点q，找到距离q 点的使距离d 最小一个点，这就是最近邻居问题。相对应的，要找到与q点距离最近的K个点，就是KNN问题。

针对NN问题，如果采用线性遍历的方式考虑所有点，将会造成很大的性能损耗。而一个比较合理的思路是，将二分查找的逻辑应用于点集合的检索，从而能将时间损耗降低为log(N)级别。也就是说，如果能够实现以nlog(N)的时间消耗将点集合建立成某种有序的数据结构。那么在搜索时，就可以通过类似于二分查找的方式达到log(N)级别的速度[4]。

2.1.2　优先点树概述

vp-tree(vantage point tree),中文名称，优先点树，正是上述思路的一种实现。vp-tree从原理上说，是基于三角不等式进行递归分解的剪枝技术，其核心思想奠基了两种情况下的正确性。第一种，是在检索过程中，对于那些远远超出搜索范围的分支，就不需要进行搜索了。第二种，是当搜索目标点显然在某一个范围内的时候，外部的其他分支就都不必搜索了[5]。基于这两个原则，搜索点的数量和点的距离计算的次数都被大幅度地减少，从而显著提升了性能。

2.1.3　优先点树的基本原理

vp-tree的基本思路就是对点集合进行空间划分。第一步，要选择一个点作为vantage point，也就是优先点。第二步，集合中的所有点要计算自己与vp的距离。第三步，根据距离值的大小将点集合均分为两支，距离小于等于中值的为left/inside子集合，距离大于等于中值为right/outside子集合。第四步，以left/inside集合作为左子树的根节点，right/outside集合为右子树的根节点，再针对这两棵子树分别递归地进行上述划分，从而形成一颗平衡的二叉树。综上可知，vp-tree最终实现了整个点集合内部的一个球状分割。

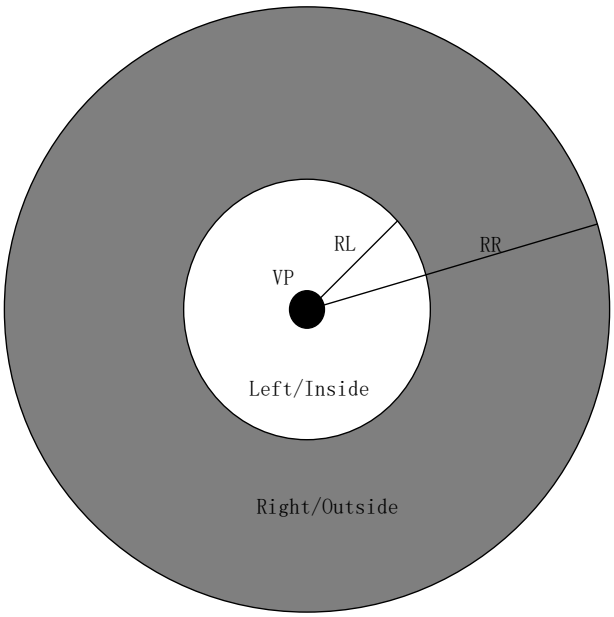

图 2.1: vp-tree点集合分割示意图

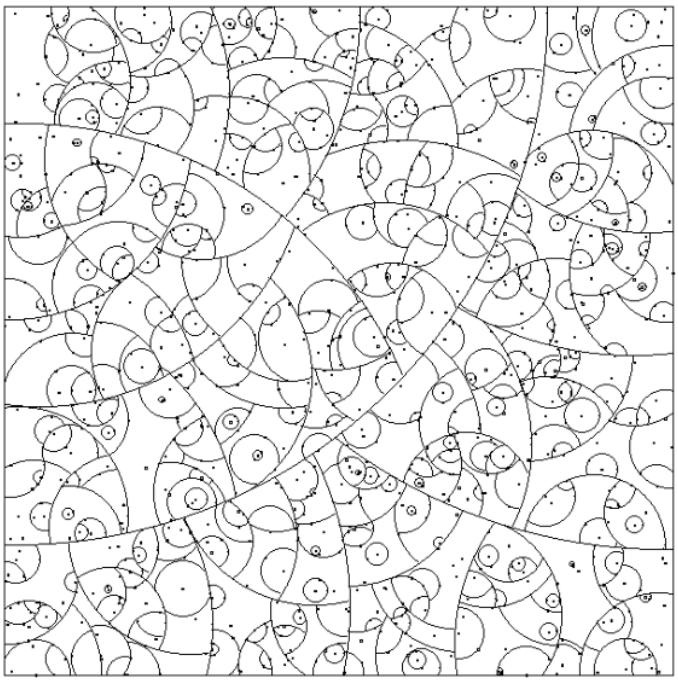

图 2.2: vp-tree空间分割示意图

2.1.4　最简单优先点树的结构和搜索过程

如图2.3所示，即为一个最简单的vp-tree的内部结构，其包括一个用于只是优先点的VP-ID，一个中值mu和分别左右子树的两个指针。

搜索算法的思路是这样的。对于一次检索的目标点q和当前vp-tree的节点node，我们会设置一个容忍阈值u。首先计算q与node的vp之间的距离d,如果距离d¿=mu+u,就舍弃左子树，只搜索右子树。反之，如果d¡=mu-u,就舍弃右子树，只搜索左子树。如果mu-u¡d¡mu+u,那么无法完成剪枝，左右子树都要搜索。

这里显而易见的是，**容忍阈值u越小，剪枝的可能性越大，搜索性能越好**。因此u的选择应该是随着递归过程的推进而不断代之以距离q最小的距离，因为既然u是目前最小的距离，那么比u距离更大的点也就不必考虑了。因此，**如何实现算法使得容忍阈值以较快的速度收敛**，是代码实现的重点，我们将在第四章具体讨论。

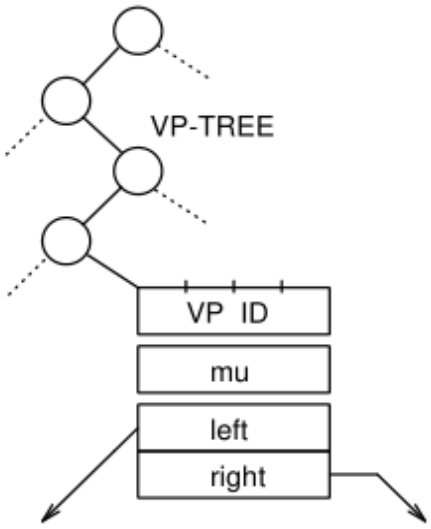

图 2.3: 最简单vp-tree的结构示意图

Algorithm 1: 最简单vp-tree的搜索过程search

```
1  n ← currentNode;
2  if n!=null then
3      return
4  x ← distance(q,n.vp);
5  if x < u then
6      u ← x;
7      best ← n.vp;
8  if x ≥ mu-u then
9      search(n.right);
10 if x ≤ mu+u then
11     search(n.left);
```

2.2 Lucene与ElasticSearch

2.2.1 概述

Lucene是一套用于全文检索和搜索的开放源代码程序库，由Apache软件基金会支持和提供。Lucene提供了一个简单却强大的应用程序接口，能够做全文索引和搜索，在Java开发环境里Lucene是一个成熟的免费开放源代码工具；就其本身而论，Lucene是现在并且是这几年，最受欢迎的免费Java信息检索程序库。

Elasticsearch是一个基于Lucene库的搜索引擎。它提供了一个分布式、支持多租户的全文搜索引擎，具有HTTP Web接口和无模式JSON文档。Elasticsearch可以用于搜索各种文档。它提供可扩展的搜索，具有接近实时的搜索，并支持多租户。Elasticsearch是分布式的，这意味着索引可以被分成分片，每个分片可以有0 个或多个副本。每个节点托管一个或多个分片，并充当协调器将操作委托给正确的分片。再平衡和路由是自动完成的。[6]相关数据通常存储在同一个索引中，该索引由一个或多个主分片和零个或多个复制分片组成。一旦创建了索引，就不能更改主分片的数量。[7]

2.2.2 核心数据结构

正排索引：正排表是以文档的ID为关键字，表中记录文档中每个字的位置信息，查找时扫描表中每个文档中字的信息直到找出所有包含查询关键字的文

档。这种组织方法在建立索引的时候结构比较简单，建立比较方便且易于维护;因为索引是基于文档建立的，若是有新的文档加入，直接为该文档建立一个新的索引块，挂接在原来索引文件的后面。若是有文档删除，则直接找到该文档号文档对应的索引信息，将其直接删除。但是在查询的时候需对所有的文档进行扫描以确保没有遗漏，这样就使得检索时间大大延长，检索效率低下。

倒排索引：倒排索引（英语：Inverted index），也常被称为反向索引、置入档案或反向档案，是一种索引方法，被用来存储在全文搜索下某个单词在一个文档或者一组文档中的存储位置的映射。它是文档检索系统中最常用的数据结构。

2.2.3　选择的原因

天然分布式属性和自称服务的体系，节省了单独开发WebService的时间。ES-Geometry的良好兼容性。性能良好，安全稳定。

2.3　地理相关技术概念介绍

2.3.1　地图瓦片数据

瓦片地图金字塔模型是一种多分辨率层次模型，从瓦片金字塔的底层到顶层，分辨率越来越低，但表示的地理范围不变。首先确定地图服务平台所要提供的缩放级别的数量N，把缩放级别最高、地图比例尺最大的地图图片作为金字塔的底层，即第0层，并对其进行分块，从地图图片的左上角开始，从左至右、从上到下进行切割，分割成相同大小(比如256x256像素)的正方形地图瓦片，形成第0层瓦片矩阵;在第0层地图图片的基础上，按每2x2像素合成为一个像素的方法生成第1层地图图片，并对其进行分块，分割成与下一层相同大小的正方形地图瓦片，形成第1层瓦片矩阵;采用同样的方法生成第2层瓦片矩阵;…;如此下去，直到第N一1层，构成整个瓦片金字塔。

2.3.2　墨卡托坐标

墨卡托投影法（英语：Mercator projection），又称麦卡托投影法、正轴等角圆柱投影，是一种等角的圆柱形地图投影法。本投影法得名于法兰德斯出身的地理学家杰拉杜斯・墨卡托，他于1569年发表长202公分、宽124公分以此方式绘制的世界地图。在以此投影法绘制的地图上，经纬线于任何位置皆垂直相交，使世界地图可以绘制在一个长方形上。由于可显示任两点间的正确方位，航海用途的海图、航路图大都以此方式绘制。在该投影中线型比例尺在图中任意一

点周围都保持不变，从而可以保持大陆轮廓投影后的角度和形状不变（即等角）；但墨卡托投影会使面积产生变形，极点的比例甚至达到了无穷大。

2.4　Nodejs Express框架

Express 是一个基于Node.js 平台的极简、灵活的web 应用开发框架，它提供一系列强大的特性，帮助你创建各种Web 和移动设备应用。

第三章　系统需求分析与概要设计

3.1　GTDS系统概述

地理轨迹数据服务总体上分为三个部分，轨迹数据服务，瓦片数据服务和业务展示服务。其中业务展示服务是用户直接操作的前端，完成轨迹数据和瓦片数据的可视化功能。**注意：由于业务展示服务只有前端库的调用，没有有价值的实现，因此本文不做介绍。**瓦片数据服务主要面向外部数据源，支持瓦片数据源的配置和瓦片的增，删，改，查等功能。轨迹数据服务主要面向数据库管理员，主要功能是负责轨迹存储和相似性检索。

3.2　轨迹数据服务需求分析

3.2.1　轨迹数据服务的功能需求

整个地理轨迹数据服务的核心功能就是对相似路径的检索，因此轨迹数据服务的功能需求有两个。一个是对批量轨迹数据导入建立索引的功能，另一个是Trajectory KNN功能，即根据目标轨迹的ID，执行检索算法，找出与目标轨迹最相近的K 条轨迹。

表 3.1: 轨迹数据服务功能需求列表

需求ID	需求名称	需求描述
R1	轨迹索引批量初始化	服务调用方能够通过上传批量的轨迹数据， 在规定时间内完成轨迹索引的建立，并返回结果
R2	相似轨迹knn检索	服务调用方能够通过传递目标轨迹ID和检索量K， 在规定时间内返回K个与目标轨迹最相似的K个轨迹

3.2.2　轨迹数据服务的非功能需求

Trajectory KNN问题是一个计算密集型问题，用户对响应时间一定会有要求。同时，并发量也是必须考虑的问题

表 3.2: 轨迹数据服务非功能需求列表

时间特性	对于百万级别的轨迹数据量，服务应该在**2s**之内返回检索结果
负载特性	服务应该能应对**10w**以上的并发访问

3.3　瓦片数据服务需求分析

3.3.1　瓦片数据服务的需求综述

在轨迹检索服务中，对于地图瓦片数据的要求有增加，删除，修改，查询，数据源配置。其中以查询和更新这两部分功能的细分功能最多。对于查询而言，可能被用户需要的有整张地图的全量查询，局部bounding-box渲染查询，单个瓦片数据的查询。对于更新而言，可能需要局部地图的更新，bounding-box更新。**注意：在GTDS的功能需求中，对于瓦片的增加和删除都是以整张地图为单位的，局部的增加和删除这种需求并不存在，所以此处不列为功能需求。** 而对于非功能需求，瓦片数据服务涉及到的计算主要是坐标转换，请求解析，缓存处理。计算量不大，所以并不是计算密集型应用，而是IO密集型应用，高并发和快速响应是其必须满足的非功能特性。除此之外，由于瓦片服务本身不存储瓦片数据，用户的瓦片数据可能是存在各种不同的数据库中的，因此服务还应该独立于不同的数据库，做到高可用性，高扩展性。

3.3.2　瓦片数据服务功能需求

表 3.3: 瓦片数据服务功能需求列表

需求ID	需求名称	需求描述
R1	新增地图瓦片数据	服务调用方能够通过瓦片服务，参数为地图名称和图瓦片数据，增加一个地区的完整地图瓦片数据
R2	删除地图瓦片数据	服务调用方能够通过瓦片服务，参数为地图ID，删除一个地区的全部地图瓦片数据
R3	局部更新地图瓦片数据	服务调用方能够通过瓦片服务，参数为地图ID和地图数据，更新一张大地图中某一个地区的地图瓦片数据
R4	bounding-box更新地图瓦片数据	服务调用方能够通过瓦片服务，参数为地图ID，经纬度范围，地图数据，更新一张大地图中某一个地区某一经纬度矩形范围内的地图瓦片数据
R5	局部更新地图瓦片数据	服务调用方能够通过瓦片服务，参数为地图ID和地图数据，更新一张大地图中某一个地区的地图瓦片数据
R6	地图瓦片数据全量查询	服务调用方能够通过瓦片服务，通过获取全量数据的JSON文件，作为输入的数据源，获取整张地图的全量数据
R7	单个地图瓦片数据查询	服务调用方能够通过瓦片服务，参数为地图ID和栅格坐标zxy，获取指定的瓦片
R8	局部bounding-box的渲染查询	服务调用方能够通过瓦片服务，参数为地图ID和经纬度矩形范围，获取这一部分的地图渲染结果

表 3.4: 瓦片数据服务非功能需求列表

时间特性	在正常负载情况下，服务的平均响应时间应在**1s**之内
负载特性	服务能稳定应对百万级别的并发访问，不会出现延迟超过**10s**或服务崩溃的情况
高可用性	服务能通过设置中间件的方式，便捷地对接到各种不同的数据库，并保证运行正常

3.3.3　瓦片数据服务非功能需求

3.3.4　瓦片数据服务用例图

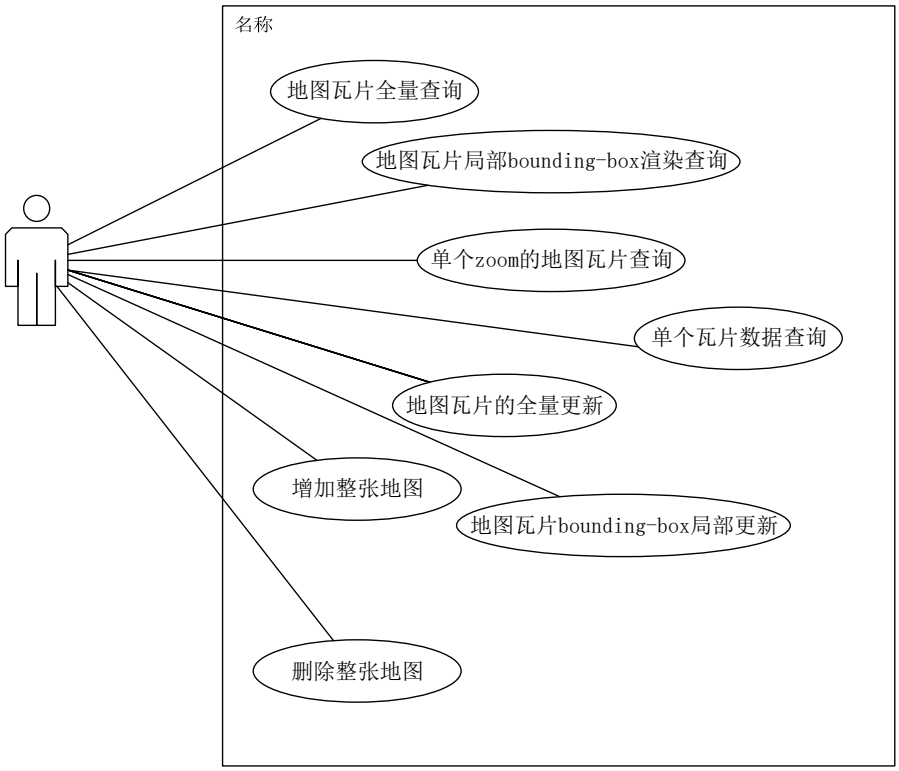

图 3.1: 瓦片数据服务的功能需求

注意：以上用例中的功能并不是逻辑完备的，某些逻辑功能，比如局部地图瓦片非渲染查询，这种需求在实际应用中没有使用场景，这里就没有列举。

3.3.5　瓦片数据服务用例描述

瓦片数据服务的部分功能操作简单明确，无需使用用例描述。本文只对操作比较复杂的“bounding-box更新地图瓦片数据”进行了具体描述。

表 3.5: bounding-box更新地图瓦片数据用例描述表

ID	**UC1**
名称	**bounding-box更新地图瓦片数据**
参与者	普通用户
目的	更新某张地图中一个矩形范围内的瓦片数据，以改变轨迹展示背景
描述	用户通过浏览器发送矩形参数和瓦片数据，以实现对瓦片数据库中数据的修改，进而改变业务展示的结果
优先级	高
触发条件	某一地图中某一部分数据发生变更，需要更改
前置条件	服务正常运行，用户进入服务界面
后置条件	地图瓦片数据完成更新，业务展示结果经过刷新可以看到地图变化
正常流程	1.进入瓦片数据更新界面 2.设置north,south,west,east,4个经纬度值 3.设置zoom区间 4.设置是否进行精度模糊 5.点击浏览本地文件并上传文件数据 6. 点击更新瓦片数据
异常流程	2a.经纬度大小值错误，前端应自动检测并警告 3a.指定矩形范围太小，在较小的zoom下无法更新瓦片,服务端应发现错误并返回告知用户 5a.用户选择的文件格式错误，则前端对用户发出警告 5b.用户选择的文件中并没有指定bounding-box中的数据，服务端应返回错误报告并挺行用户更改文件

3.4　GTDS系统概要设计

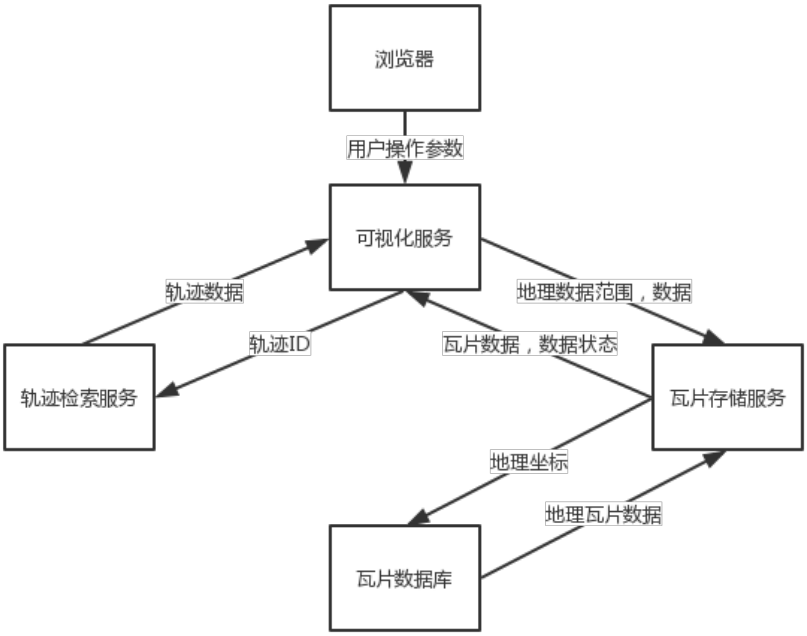

图 3.2: GTDS的总体架构

如图 3.2所示，可视化服务直接接收用户的界面操作，将其转化为轨迹ID和瓦片数据范围，作为参数，分别传递给轨迹搜索服务和瓦片存储服务。再使用前端库汇总这两部分的数据共同完成可视化功能。而瓦片存储服务本身不存储瓦片数据，只负责地图范围的解析，地理坐标的转换以及数据格式的转换，瓦片数据的存放位置是在具体的瓦片数据库中。

3.5　本章总结

本章先是通过概述说明了整个系统主要组件的职责和相互之间的关系。然后分别针对两个主要服务进行了需求分析。需求分析的过程中，使用了需求列表来展现主要的功能需求和非功能需求，还使用了用例图的形式对功能较多的地图瓦片服务进行了功能划分，并对其中操作较为复杂的功能使用用例描述进行了详细介绍。最后给出了整个系统的概要设计，明确了各个组件之间的依赖关系，为下一章按照模块进行详细设计和实现做准备。

第四章　系统详细设计与实现

4.1　轨迹检索检索服务详细设计

4.1.1　概述

轨迹检索服务是在全文检索引擎ElasticSearch的基础上，扩展实现了单独的优先点树索引结构来实现的。由于服务对外接口访问，分布式，故障检测等功能由ElasticSearch提供，因此本文只关注具体索引结构的设计和实现。轨迹检索服务的核心思想是，以JTS-Geometry作为路径的存储形式，也就是只考虑路径的几何展现，不去考虑路径的方向性。并以豪斯多夫距离衡量两个Geometry之间的距离。两个Geometry之间的豪斯多夫距离越短，就认为两个Geometry越相似，也就认为两个路径越相似。基于这样的前提，我们将Geometry作为度量空间中的数据点，实际上把相似路径检索问题转化为Geometry的KNN问题，然后通过建立vp-tree的数据结构，进行求解。**注意：由于算法实现流程的细节较多，本文采用流程+实现+设计要点的顺序进行阐述，其中设计要点是对主流程实现细节的单独阐述。**

4.1.2　优先点树索引类图

4.1.3　优先点树节点结构

本文所设计实现的优先点树的结构是对最简单vp-tree结构的改良，将原生vp-tree的两路结构改为多路结构。相应地，就需要把按中值进行二分修改为以边界值进行多分，并且为了提高检索的速度，减少检索时的距离运算量，改良后的vp-tree的非叶节点不仅存储了Vantage Point ID和每棵子树的指针，还存储了每个子树的距离值的上界和下界以及最大距离值。具体结构见图 4.1所示为一个4路vp-tree的内部结构示意图。

4.1.4　初始建树的流程

初始建树的输入数据是一系列的docId+Geometry，初始建树的输出是一颗完整的多路vp-tree。具体流程见 4.3。**注意：本文用DP来表示Data Point，即数据点，用VP来表示Vantage Point，即被选做优先点的数据点，用Node来表示vp-tree的节点。**

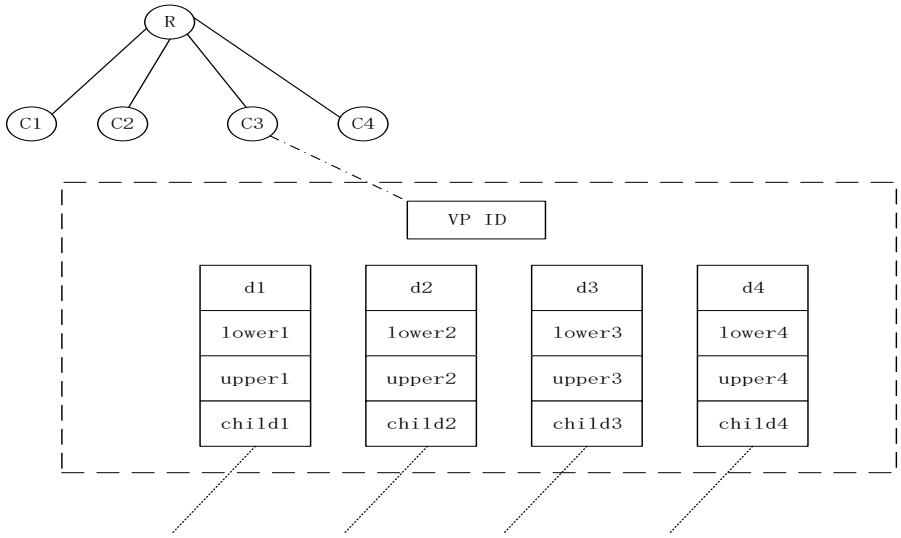

图 4.1: 4路vp-tree的内部结构示意图

如图所示初始建树实际上是一个递归的过程。但是为了避免出现内存溢出，本文选择了用循环+栈的模式。流程开始时，首先将根节点压栈，然后判断数据点的个数是不是小于叶节点的数据量标准。如果是，就意味着已经达到终止这一分支的条件，则直接为叶节点，然后通过空流程返回循环判断。如果数据量仍然大于叶节点的数据量，就使用选取函数选择优先点，计算其他各个数据点到优先点的距离，再根据距离进行升序排序。**注意：由于此时有一个点被选做优先点，所以数据点总量要减一**，然后判断剩余的数据量是否大于扇出数，如果数据量已经不能满足全部的扇出，那么就初始化为单个非叶节点入栈，从而进入下一次循环。如果数据量依然足够分割全部的扇出，就按照距离排序的结果进行多路切分，用新建的子节点代表新的子树，并入栈。将距离值，上下界值和子节点指针分别填入对应的数组中，再进入下一次循环。以此类推，循环往复，完成所有数据点的建树操作。

4.1.5　初始建树算法实现

见图**??注意，出于节省篇幅考虑，省略了部分简单实现**

```
private void createNode(Node root, BulkloadContext bulkloadContext,
SelectVpStrategy selectVpStrategy) {
    Deque<Node> nodeStack = new LinkedList<>();
    IntStack offsetStack = new IntStack(64), lengthStack = new IntStack(64);
        //迭代使用的偏移量栈和长度
    nodeStack.push(root);
    offsetStack.push(0);
        lengthStack.push(positions.length);
        //初始状态
    float[] distanceBuffer = new float[bulkloadContext.total];
    Node currentNode = null;
    int currentOffset, currentLength;
    int fanout = configuration.getFanout();
    SelectVpResult selectVpResult = new SelectVpResult();
    while (!nodeStack.isEmpty()) {
        currentNode = nodeStack.pop();
        currentOffset = offsetStack.pop();
        currentLength = lengthStack.pop();
        if (currentLength > configuration.getEntrySize()) {
            currentNode.initAsNonLeaf(configuration);
            selectVpStrategy.selectVp();//选取优先点
            ……//计算各个数据点与优先点的距离并
            --currentLength;//数据点总量减 1
            if (currentLength <= fanout) {
               currentNode.childrenBounds.add(distanceBuffer[1]);
                ……..//初始化为单个非叶节点并且入栈
            } else {
               int childSize = (int) Math.ceil(currentLength * 1.0 / fanout);
                //计算每个子树应有的数据点量
                for (int i = 0, start = 1, end; i < fanout; i++) {
                    end = Math.min(start + childSize - 1, currentLength);
                    if (end < start) {
                        break;
                    }
                    currentNode.childrenBounds.add(distanceBuffer[start]);
                    currentNode.childrenBounds.add(distanceBuffer[end]);
                    currentNode.distances[i]= distanceBuffer[end];
                    currentNode.childrenNodes[i]=new Node(nextNodeId(), false)
                    nodeStack.push(currentNode.childrenNodes[i]);
                    offsetStack.push(currentOffset + start);
                    lengthStack.push(end - start + 1);
                    start = end + 1;
                    //设置子树指针并分别为每路子树，设置最大距离值，距离上下界值
                }
            }
        } else {
            currentNode.initAsLeaf(currentLength);
            for (int i = 0; i < currentLength; i++) {
               currentNode.children.add(bulkloadContext.ids.get(currentOffset+
i]);
            }//初始化叶节点，结束一个分支
        }
    }
```

图 4.3: 初始建树代码

4.1.6　设计要点1：使用长度栈和偏移量栈记录内存状态，避免冗余内存

初始建树的输入数据是两端段很长的数组，分别保存了docID和对应的Geometry。为了减少内存使用，本文采用偏移量+长度这样的组合量来记录每个节点所涉及的数据状态，从而避免了输入数据的内存复制。另外，由于使用了栈实现，在多路划分的过程中，优先级是从右向左，而且是深度优先的。也就是说优先点树的最右边一个分支会最早完成创建。**由于vp-tree的多路切分是均衡的，所以vp-tree自然是一个平衡树，分支初始化的顺序与最终结果没有关系。**

4.1.7　设计要点1：第二，优先点的选择算法

在初始建树的过程中，优先点选取是非常关键的一步。选取的好坏取决于一个优先点能否让切分出来的各路子树的边界值相差足够大，因为各路子树的边界值相差越大，在检索的时候，距离值落入某一子树的边界内的可能性越大，剪枝成功的可能性越高，性能就越好。反之，如果优先点选择很差，导致多个子树的上下界非常接近，就很容易出现一个距离值可能在多个子树中搜索的情况，造成性能下降。因此，如何选择尽可能优的优先点，是算法实现的重点。

本文针对优先点选择的实现是基于随记取样和标准差结果的。本文认为，一个点与其他点距离的标准差越大，作为优先点的性能越好，而由于数据全量很大，不可能都计算，就采用随机抽样的方式进行。其设计思路是，在数据点全集中随机取样K 个点，作为候选的优先点，针对这K 个候选的优先点进行循环遍历，每个候选优先点再随机取K 个点作为参照点，然后计算每个候选优先点和参考点之间距离的标准差，最后取标准差最大的那个候选点作为真正的优先点。具体实现如图 4.4所示。

4.1.8　KNN问题的定义和解决思路

在我们的轨迹检索服务的作用域内，K Nearest Neighbour的含义是，找到与目标Geometry距离最近的K个Geometry。

原生vp-tree的搜索算法是面对NN问题，也就是single nearest Neighbour问题，只招一个距离最近的点。那么面对KNN问题，显然不能通过简单地运用K次原生搜索算法来解决，那样毫无疑问会造成重大的性能损耗。

一个比较直观的想法是，使用一个大小为K的最小堆，在搜索过程中实时更新这个最小堆的状态，那么在搜索算法走完的时候，这个最小堆中的结果就是K个距离最近的Geomtry。这是[**？**]中所阐述的思路，本文实现的算法的借鉴

了这一思路，同样是用堆来动态维护状态，但采取了一些措施应对这一思路的明显短板以获取更好的检索性能。详见下节。

4.1.9　设计焦点：容忍距离的收敛起点和收敛速度

在KNN问题的搜索过程中，对于每一个多路节点，除了考虑与优先点的距离之外，还要考虑一个容忍距离T,也就是超出这个容忍距离T的数据点不再予以考虑，这是距离范围剪枝的基本依据。显而易见的是，这个容忍距离越小，成功剪枝的概率越高，检索性能越好。但是如果容忍距离太小，又可能过度剪枝造成检索不到结果。所以容忍距离的收敛速度是直接影响检索性能的关键。

在上文所述的单纯用最小堆动态更新检索结果的算法中，其最大问题在于，容忍距离是从正无穷开始更新的。这使得检索从一开始完全不可能做到剪枝，容忍距离的收敛速度会非常慢，导致剪枝的效率很低，性能也比较差。

4.1.10　本文的算法实现思路概述

基于上面所提到的容忍距离收敛较慢问题，本文采用了结果堆预填+回溯的方式进行优化处理。即通过原生vp-tree搜索算法，先找到single nearest neighbour，并保存从root到single nearest neighbour的整条路径。然后再通过整条路径的回溯先预填结果堆，这样以结果堆中最大距离作为容忍距离，再回溯整条路径中的节点，完成检索。这样通过预先填结果堆，使得容忍距离以更低的起点收敛，收敛的速度更快，性能更好。

4.1.11　本文的算法实现的流程图

4.1.12　本文的算法实现的代码

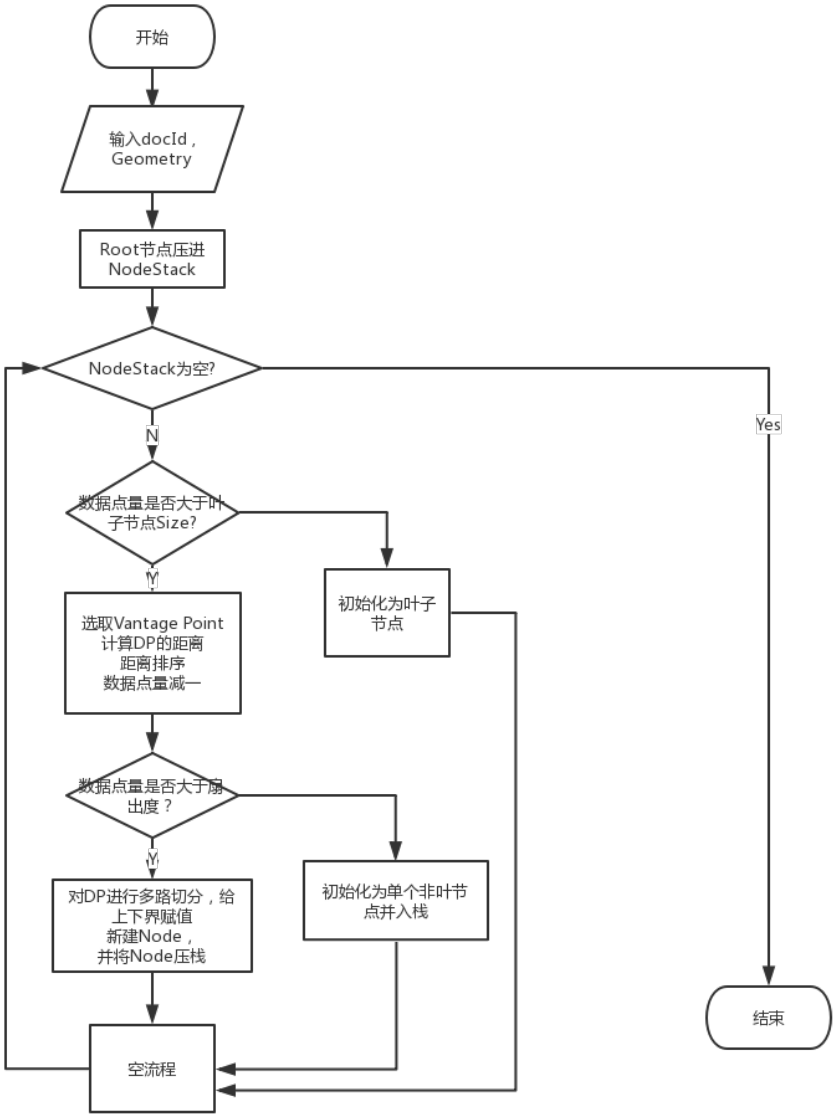
开始
输入docId，
Geometry
Root节点压进
NodeStack
NodeStack为空?
Yes
N
数据点量是否大于叶
子节点Size?
Y
初始化为叶子
节点
选取Vantage Point
计算DP的距离
距离排序
数据点量减一
数据点量是否大于扇
出度？
Y
对DP进行多路切分，给
上下界赋值
新建Node，
并将Node压栈
初始化为单个非叶节
点并入栈
空流程
结束

图 4.2: 初始建树流程图

```
public void selectVp(BulkloadContext bldCtx, int curOff, int
curLen, int[] values,float[] disBuf, SelectVpResult result) {
   int spSize = Math.max((int) (curLen * conf.ratio), 1);
   SampleResult sampleResult = new SampleResult(bldCtx.spBuf,
spSize),
   SampleResult sampleResultInner = new
SampleResult(bldCtx.spBufInner, spSize);
   //随机抽取候选优先点
   sampler.sample(values, curOff, curLen, sampleResult);

   float maxStdev = -1;
   for (int i = 0; i < spSize; i++) {
      Geometry candidate = bldCtx.geometries[bldCtx.spBuf[i]];
      sampler.sample(values, curOff, curLen, sampleResultInner);
      //随机抽取参考点
      for (int j = 0; j < spSize; j++) {
         ……//计算候选点与对应参考点的距离
      }
      float current = computeStdev(disBuf, 0, spSize);
      //计算当前候选点的标准差
      if (current > maxStdev) {
         maxStdev = current;
         result.vpIndex = bldCtx.spBuf[i];
         result.vpGeometry = candidate;
      }
   }
}
```

图 4.4: 优先点选取代码

第五章　总结和展望

5.1　这是节标题

这是地理信息系统的总结和展望

5.1.1　这是小节标题

5.1.2　这是小小节标题

参考文献

[1] Gao Q, Zhang FL, Wang RJ, and Zhou F. Trajectory big data:a review of key technologies in data processing. *Ruan Jian Xue Bao*, 28(4):28–34, 2016.

[2] BaiduYingyan System. http://lbsyun.baidu.com/trace.

[3] tileserver System. http://tileserver.org.

[4] Peter N. Yianilos. Data structures and algorithms for nearest neighbor search in general metric spaces. pages 311–321, 1993.

[5] Keinosuke Fukunaga and Patrenahalli M. Narendra. A branch and bound algorithms for computing k-nearest neighbors. *IEEE Trans. Computers*, 24(7):750–753, 1975.

[6] Elasticsearch.org. http://www.elasticsearch.org/.

[7] How to monitor Elasticsearch performance. https://www.datadoghq.com/blog/monitor-elasticsearch-performance-metrics/#what-is-elasticsearch.

简历与科研成果

基本情况 韩淳，男，汉族，1994 年 8 月出生，吉林省松原市人。

教育背景

2017.9～2019.6 南京大学软件学院 硕士

2013.9～2017.6 南京大学软件学院 本科

这里是读研期间的成果（实例为受理的专利）

1. 刘海涛，**韩淳**，“基于矢量瓦片和优先点树的相似路径检索和可视化服务的设计和实现”，申请号：20xx1018xywz.a，已受理。

致　　谢

这里是致谢。一般的感谢顺序：导师，其他指导老师，师兄弟姐妹、同学，父母和伴侣。