啄木鸟后台围栏轨迹里程计算性能及问题汇总

目录

[1. 知识点](#_Toc5704_WPSOffice_Level1) [1](#_Toc5704_WPSOffice_Level1)

[1.1. 围栏与应用](#_Toc12499_WPSOffice_Level2) [1](#_Toc12499_WPSOffice_Level2)

[1.2. 距离&相似度算法](#_Toc2368_WPSOffice_Level2) [2](#_Toc2368_WPSOffice_Level2)

[2. 功能点](#_Toc12499_WPSOffice_Level1) [4](#_Toc12499_WPSOffice_Level1)

[2.1. 围栏设置与管理](#_Toc20078_WPSOffice_Level2) [4](#_Toc20078_WPSOffice_Level2)

[2.2. 围栏轨迹里程计算](#_Toc17518_WPSOffice_Level2) [5](#_Toc17518_WPSOffice_Level2)

[3. 数据存储参考方向](#_Toc2368_WPSOffice_Level1) [5](#_Toc2368_WPSOffice_Level1)

[3.1. TableStore实战：轻松实现轨迹管理与地理围栏](#_Toc16130_WPSOffice_Level2) [5](#_Toc16130_WPSOffice_Level2)

[3.2. Hadoop+HDFS+HIVE：存储经纬度点聚合查询](#_Toc29181_WPSOffice_Level2) [5](#_Toc29181_WPSOffice_Level2)

[4. 数据库统计和增量备份](#_Toc20078_WPSOffice_Level1) [6](#_Toc20078_WPSOffice_Level1)

[4.1. Canal增量日志服务](#_Toc1945_WPSOffice_Level2) [6](#_Toc1945_WPSOffice_Level2)

[4.2. Otter分部式数据库同步](#_Toc14091_WPSOffice_Level2) [7](#_Toc14091_WPSOffice_Level2)

[5. Geometry计算方法](#_Toc17518_WPSOffice_Level1) [9](#_Toc17518_WPSOffice_Level1)

[5.1. 轨迹点与围栏关系计算逻辑](#_Toc9238_WPSOffice_Level2) [9](#_Toc9238_WPSOffice_Level2)

[5.1.1. 轨迹在围栏内部](#_Toc12499_WPSOffice_Level3) [9](#_Toc12499_WPSOffice_Level3)

[5.1.2. 轨迹在围栏外部](#_Toc2368_WPSOffice_Level3) [10](#_Toc2368_WPSOffice_Level3)

[5.2. 轨迹里程计算效率测试](#_Toc17989_WPSOffice_Level2) [10](#_Toc17989_WPSOffice_Level2)

[5.2.1. Geometry轨迹围栏交集计算效率](#_Toc20078_WPSOffice_Level3) [10](#_Toc20078_WPSOffice_Level3)

[5.2.2. Geometry两个之间距离计算效率](#_Toc17518_WPSOffice_Level3) [11](#_Toc17518_WPSOffice_Level3)

[5.3. Geometry使用注意事项](#_Toc20856_WPSOffice_Level2) [12](#_Toc20856_WPSOffice_Level2)

[5.3.1. 调用方式](#_Toc16130_WPSOffice_Level3) [12](#_Toc16130_WPSOffice_Level3)

[5.3.2. 正常情况](#_Toc29181_WPSOffice_Level3) [12](#_Toc29181_WPSOffice_Level3)

[5.3.3. 异常情况](#_Toc1945_WPSOffice_Level3) [15](#_Toc1945_WPSOffice_Level3)

[5.3.4. 异常分析](#_Toc14091_WPSOffice_Level3) [17](#_Toc14091_WPSOffice_Level3)

[5.3.5. 避免异常](#_Toc9238_WPSOffice_Level3) [18](#_Toc9238_WPSOffice_Level3)

# 知识点

## 围栏与应用

[GPS/轨迹追踪、轨迹回放、围栏控制](https://www.cnblogs.com/visec479/p/4599551.html)

**[地理围栏算法解析（Geo-fencing）](https://www.cnblogs.com/LBSer/p/4471742.html)**

 地理围栏（Geo-fencing）是LBS的一种应用，就是用一个虚拟的栅栏围出一个虚拟地理边界，当手机进入、离开某个特定地理区域，或在该区域内活动时，手机可以接收自动通知和警告。如下图所示，假设地图上有三个商场，当用户进入某个商场的时候，手机自动收到相应商场发送的优惠券push消息。地理围栏应用非常广泛，当今移动互联网主要app如美团、大众点评、手淘等都可看到其应用身影。



## 距离&相似度算法

<https://blog.csdn.net/GoodShot/article/details/78302357>

在计算机人工智能领域，距离(distance)、相似度(similarity)是经常出现的基本概念，它们在自然语言处理、计算机视觉等子领域有重要的应用，而这些概念又大多源于数学领域的度量(metric)、测度(measure)等概念。   
这里拮取其中18种做下小结备忘，也借机熟悉markdown的数学公式语法。

| **英文名** | **中文名** | **算式** | **说明** |
| --- | --- | --- | --- |
| Euclidean Distance | 欧式距离 |  | 以古希腊数学家欧几里得命名的距离；也就是我们直观的两点之间直线最短的直线距离 |
| Manhattan Distance | 曼哈顿距离 |  | 是由十九世纪的赫尔曼·闵可夫斯基所创词汇；是种使用在几何度量空间的几何学用语，用以标明两个点在标准坐标系上的绝对轴距总和；也就是和象棋中的“車”一样横平竖直的走过的距离；曼哈顿距离是超凸度量 |
| Minkowski Distance | 闵氏距离 |  | 以俄罗斯数学家闵可夫斯基命名的距离；是欧式距离的推广，p=2时等价于欧氏距离，和p-范数等值 |
| Hamming Distance | 海明距离 | 逐个字符(或逐位)对比，统计不一样的位数的个数总和 | 所得值越小，参与对比的两个元素约相似；下面是从wikipedia借的4bit的海明距离示意图IMG_256 |
| Jaccard Coefficient | 杰卡德距离 |  | 越大越相似；分子是A和B的交集大小，分母是A和B的并集大小 |
| Ochiai Coefficient | ? |  |  |
| Pearson Correlation | 皮尔森相关系数 |  | 分子是两个集合的交集大小，分母是两个集合大小的几何平均值。是余弦相似性的一种形式 |
| Cosine Similarity | 余弦相似度 |  |  |
| Mahalanobis Distance | 马氏距离 | 其中S是x和y的协方差矩阵 | 印度统计学家马哈拉诺比斯(P. C. Mahalanobis)提出的，表示数据的协方差距离。它是一种有效的计算两个未知样本集的相似度的方法；若协方差矩阵是对角阵(diagonal)，则该距离退化为欧式距离 |
| Kullback-Leibler Divergence | K-L散度 |  | 即相对熵；是衡量两个分布(P、Q)之间的距离；越小越相似 |
| PMI(Pointwise Mutual Information) | 点对互信息 |  | 利用co-occurance来衡量x和y的相似度；越大越相关；可以看做局部点的互信息(mutual information) |
| NGD(Normalized Google Distance) | ? |  | 这是google用来衡量两个不同的关键字(keyword)的检索结果之间的相关程度；其中f(x)代表包含了关键字x的页面数量，f(x,y)代表同时包含了关键字x和关键字y的页面的数量，M代表google所搜索的总页数；若两个关键字总是成对出现在页面上，那么NGD值为0，相反的，如果两个关键字在所有页面上都没有同时出现过，那么NGD值为无穷；该量是从normalized compression distance (Cilibrasi & Vitanyi 2003)衍生而来的 |
| Levenshtein Distance(Edit Distance) | Levenshtein距离(编辑距离) | f(n)= | 是指两个字串之间，由一个转成另一个所需的最少编辑操作次数；俄罗斯科学家Vladimir Levenshtein在1965年提出这个概念；编辑距离越小的两个字符串越相似，当编辑距离为0时，两字符串相等 |
| Jaro-Winkler Distance | ? |  |  |
| Lee Distance | 李氏距离 |  | 在编码理论(coding theory)中两个字符串间距离的一种度量方法 |
| Hellinger Distance | ? | 当为概率密度函数时，进一步有 | 注意在作为概率意义的计算时需在测度空间进行；通常被用来度量两个概率分布的相似度，它是f散度的一种；由Ernst Helligner在1909年引进 |
| Canberra Distance | 坎贝拉距离 | whereand |  |
| Chebyshev Distance | 切比雪夫距离 |  | 切比雪夫距离是由一致范数(uniform norm)(或称为上确界范数)所衍生的度量，也是超凸度量 |

# 功能点

## 围栏设置与管理

此功能已有：实时监控>地图>显示网格。



注：基于此功能可以给用户进、出区域提醒规范控制工作范围，可以完善这部分需求。

## 围栏轨迹里程计算

  
围栏与轨迹的关系有以下几种：包含、相交、相离。围栏内轨迹是包含和相交两种关系，围栏外轨迹是相交和相离两种关系。

注：Geometry提供了可以计算几何关系的方法，只需要判断点与围栏的关系或者求交集运算即可（需要注意Geometry的方法调用）。

# 数据存储参考方向

## TableStore实战：轻松实现轨迹管理与地理围栏

<https://yq.aliyun.com/articles/668085?utm_content=m_1000026277>

<https://yqfile.alicdn.com/d61994c78686ae667deffa8a58bf9f0777f8faac.gif>

## Hadoop+HDFS+HIVE：存储经纬度点聚合查询

<https://github.com/Esri/gis-tools-for-hadoop>

1. Means：

[https://nbviewer.jupyter.org/github/nborwankar/LearnDataScience/blob/master/notebooks/D3.%20K-Means%20Clustering%20Analysis.ipynb](https://nbviewer.jupyter.org/github/nborwankar/LearnDataScience/blob/master/notebooks/D3. K-Means Clustering Analysis.ipynb)

# 数据库统计和增量备份

## Canal增量日志服务

**业务范围**

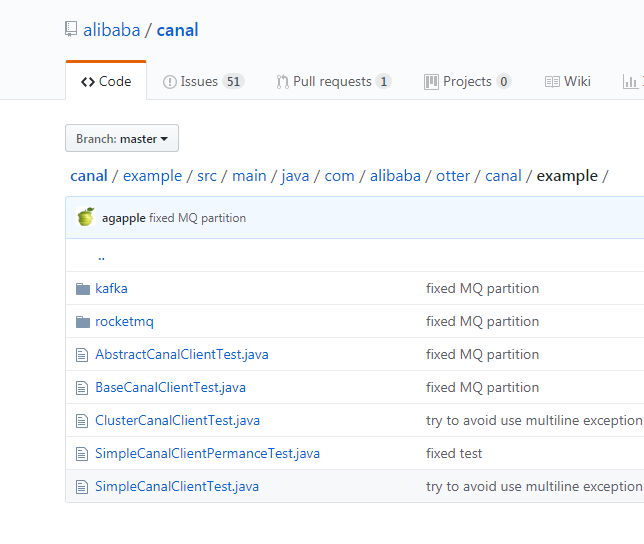
<https://github.com/alibaba/canal>

基于日志增量订阅&消费支持的业务：

* 数据库镜像
* 数据库实时备份
* 多级索引 (卖家和买家各自分库索引)
* search build
* 业务cache刷新
* 价格变化等重要业务消息

**服务支撑**

支持消息队列如kafka、rocketmq和数据库集群模式



可以明确到数据库实例、表、列、日志文件的操作。



## Otter分部式数据库同步

  阿里巴巴B2B公司，因为业务的特性，卖家主要集中在国内，买家主要集中在国外，所以衍生出了杭州和美国异地机房的需求，同时为了提升用户体验，整个机房的架构为双A，两边均可写，由此诞生了otter这样一个产品。

otter第一版本可追溯到04~05年，此次外部开源的版本为第4版，开发时间从2011年7月份一直持续到现在，目前阿里巴巴B2B内部的本地/异地机房的同步需求基本全上了otte4。

**目前同步规模**

同步数据量6亿

文件同步1.5TB(2000w张图片)

涉及200+个数据库实例之间的同步

80+台机器的集群规模

**项目介绍**

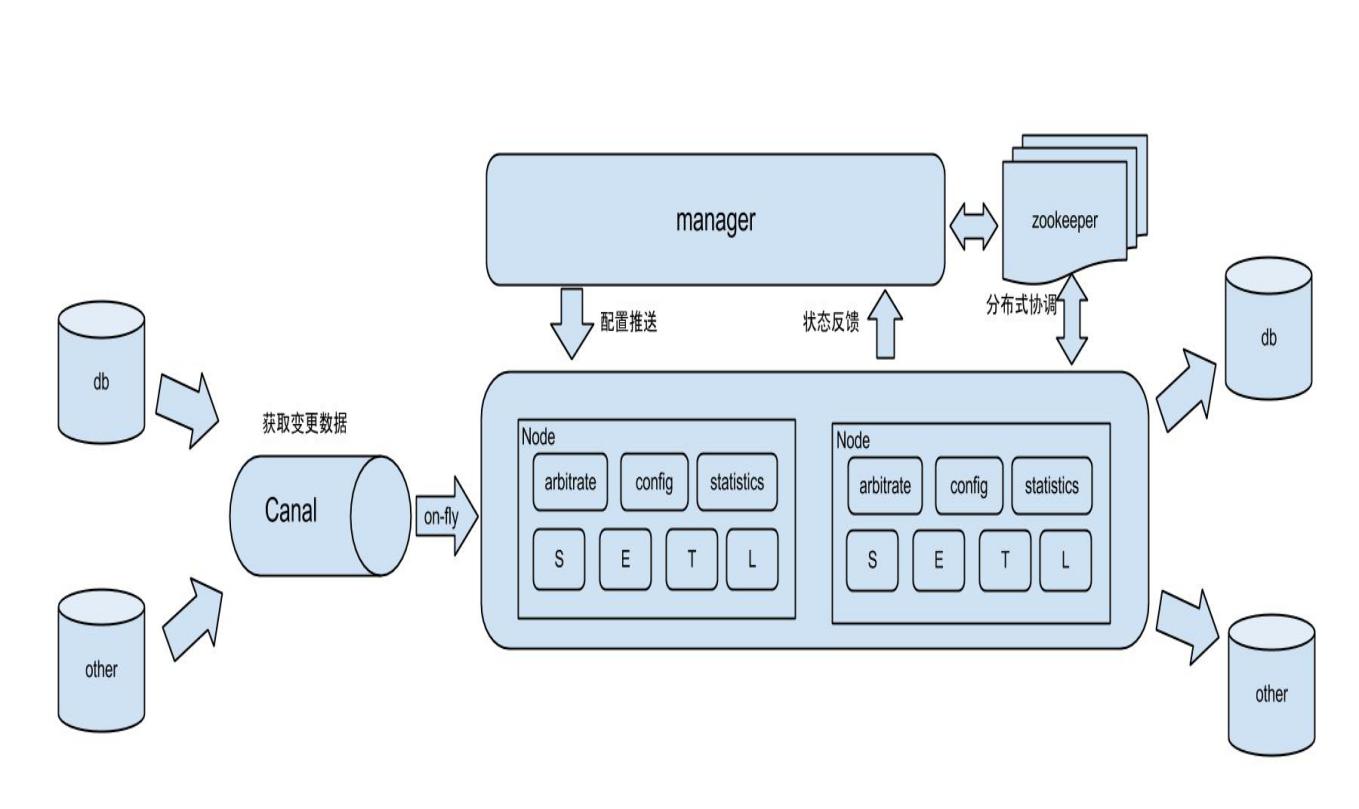
名称：otter ['ɒtə(r)]

译意： 水獭，数据搬运工

语言： 纯java开发

定位： 基于数据库增量日志解析，准实时同步到本机房或异地机房的mysql/oracle数据库. 一个分布式数据库同步系统

**工作原理**



原理描述：

1. 基于Canal开源产品，获取数据库增量日志数据。 什么是Canal, 请[点击](https://github.com/alibaba/canal)

2. 典型管理系统架构，manager(web管理)+node(工作节点)

    a. manager运行时推送同步配置到node节点

    b. node节点将同步状态反馈到manager上

3. 基于zookeeper，解决分布式状态调度的，允许多node节点之间协同工作.

# Geometry计算方法

## 轨迹点与围栏关系计算逻辑

将轨迹分成两部分进行运算，即在围栏的内部和外部，笼统地都可以用geometry求交集。

### 轨迹在围栏内部



几何关系：包含&相交。

**计算步骤：**

1. 判断是否包含
2. 判断是否相交
3. 前两步都为真，则计算轨迹里程等于围栏内运动里程，围栏外里程为0。

### 轨迹在围栏外部



几何关系：非包含&不相交。

**计算步骤：**

1. 判断是否包含
2. 判断是否相交
3. 前两步都为真，则计算轨迹里程等于围栏外运动里程，围栏内里程为0。

## 轨迹里程计算效率测试

### Geometry轨迹围栏交集计算效率

此种计算方式需要做两个维度的计算：

* 轨迹本身的里程
* 交叉里程（围栏内里程）

**轨迹点的计算时间复杂度： T(n)=O(f(n-1));**

**轨迹与围栏交集里程时间复杂度： T(n)=O(f(n));**

|  |  |  |
| --- | --- | --- |
| **轨迹点个数** | **耗时(ms)** | **平均耗时(ms)** |
| 100 | 485,24,28,33,43,26,70,34,52,39, | 83 |
| 300 | 545,38,34,30,33,27,25,42,32,33, | 83 |
| 500 | 570,40,34,43,32,29,49,42,28,27, | 89 |
| 1000 | 580,41,33,31,30,37,30,28,27,28, | 86 |
| 2000 | 641,38,41,52,34,51,42,40,29,28, | 99 |
| 3000 | 633,46,53,38,29,31,32,58,25,23, | 96 |
| 5000 | 619,53,57,33,39,76,32,38,25,30, | 100 |
| 8000 | 660,46,45,64,33,35,35,49,40,32, | 103 |
| 10000 | 695,62,60,60,33,47,43,50,31,28, | 110 |

### Geometry两个之间距离计算效率

此种方式只需要计算轨迹的里程，计算过程中区分围栏内里程和围栏外里程。

**轨迹点的计算时间复杂度： T(n)=O(f(n-1));**

|  |  |  |
| --- | --- | --- |
| **轨迹点个数** | **耗时(ms)** | **平均耗时(ms)** |
| 100 | 363,24,33,14,16,17,14,14,11,10, | 51 |
| 300 | 449,55,41,33,26,20,20,13,30,17, | 70 |
| 500 | 451,52,31,38,36,34,27,28,25,32, | 75 |
| 1000 | 687,72,65,62,53,75,48,47,62,464, | 163 |
| 2000 | 636,100,95,105,81,97,101,244,81,111, | 165 |
| 3000 | 695,128,136,116,190,145,154,95,170,156, | 198 |
| 5000 | 802,301,326,222,270,210,251,497,206,268, | 335 |
| 8000 | 1075,464,437,497,487,489,527,494,490,362, | 532 |
| 10000 | 1107,1148,587,683,534,632,639,580,644,741, | 729 |

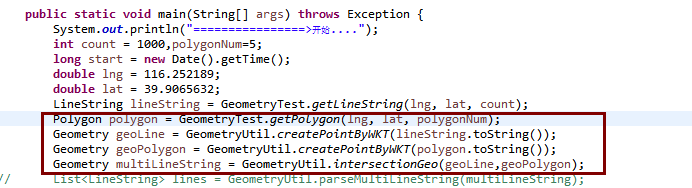
注：这里耗时体现在每次都对点是否在围栏里面进行判断。

## Geometry使用注意事项

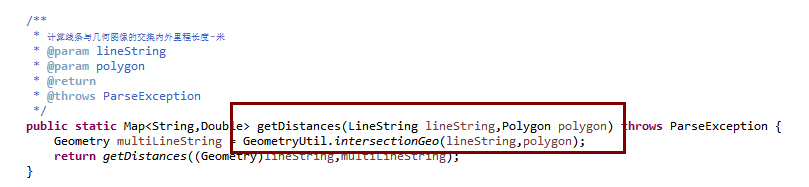
使用Geometry求轨迹与围栏交集运算需要注意方法调用。

### 调用方式

错误的方式：



正确使用方式：



注：下面产生异常的方式主要是方法调用产生的异常，导致geometry与geometry之间判断相交产生冲突。

### 正常情况

**正常有无交集**

LINESTRING (116.25219844906536 39.906786278366184, 116.25244821464976 39.90685660540906,.......)

POLYGON ((116.25535777758334 39.915633025102814, 116.2574767832826 39.92280683703119, 116.26090814827204 39.931584334357126, 116.26145755480688 39.94115959034832, 116.26261596265554 39.94975783123877, 116.25535777758334 39.915633025102814))

lineStringContainsLineString=true

**polygonContainsPolygon=true**

================>开始....

线的长度:0.7536681662645706

线轨迹的实际里程(米):63327.0

面的面积(平方米):493080.36165995616

两个空间对象最近的距离:0.003595942642908063

两个几何对象的交集:LINESTRING EMPTY

围栏内里程(米):0.0

围栏外里程(米):63327.0

================>耗时641ms,points=1000

================>开始....

线的长度:0.7536681662645706

线轨迹的实际里程(米):63327.0

面的面积(平方米):493080.36165995616

两个空间对象最近的距离:0.003595942642908063

两个几何对象的交集:LINESTRING EMPTY

围栏内里程(米):0.0

围栏外里程(米):63327.0

================>耗时102ms,points=1000

================>开始....

线的长度:0.7536681662645706

线轨迹的实际里程(米):63327.0

面的面积(平方米):493080.36165995616

两个空间对象最近的距离:0.003595942642908063

两个几何对象的交集:LINESTRING EMPTY

围栏内里程(米):0.0

围栏外里程(米):63327.0

================>耗时111ms,points=1000

**正常有交集**

**lineStringContainsLineString=true**

**polygonContainsPolygon=true**

================>开始....

线的长度:0.7621779140342029

线轨迹的实际里程(米):62687.0

面的面积(平方米):808894.4476752883

两个空间对象最近的距离:0.0

两个几何对象的交集:MULTILINESTRING ((116.25688221662699 39.914395421882276, 116.25758584647397 39.91484391938795, 116.25822427104575 39.91560473167323, 116.25860369811619 39.91625944161212, 116.2587542555323 39.917155419248814, 116.25934281853155 39.91740573376146, 116.26017238150571 39.91837305892461, 116.26047989801104 39.918917913968706), (116.26066125935785 39.91908979976905, 116.26139721203687 39.91909890945397, 116.261493614772 39.91969117292497, 116.26209510309556 39.919864426634085, 116.26267162465133 39.920768643283466, 116.26283447817538 39.92087285535724, 116.26320152504029 39.92117941999886, 116.26371154759461 39.92198072026618))

围栏内里程(米):860.0

围栏外里程(米):61827.0

================>耗时529ms,points=1000

================>开始....

线的长度:0.7621779140342029

线轨迹的实际里程(米):62687.0

面的面积(平方米):808894.4476752883

两个空间对象最近的距离:0.0

两个几何对象的交集:MULTILINESTRING ((116.25688221662699 39.914395421882276, 116.25758584647397 39.91484391938795, 116.25822427104575 39.91560473167323, 116.25860369811619 39.91625944161212, 116.2587542555323 39.917155419248814, 116.25934281853155 39.91740573376146, 116.26017238150571 39.91837305892461, 116.26047989801104 39.918917913968706), (116.26066125935785 39.91908979976905, 116.26139721203687 39.91909890945397, 116.261493614772 39.91969117292497, 116.26209510309556 39.919864426634085, 116.26267162465133 39.920768643283466, 116.26283447817538 39.92087285535724, 116.26320152504029 39.92117941999886, 116.26371154759461 39.92198072026618))

围栏内里程(米):860.0

围栏外里程(米):61827.0

================>耗时97ms,points=1000

================>开始....

线的长度:0.7621779140342029

线轨迹的实际里程(米):62687.0

面的面积(平方米):808894.4476752883

两个空间对象最近的距离:0.0

两个几何对象的交集:MULTILINESTRING ((116.25688221662699 39.914395421882276, 116.25758584647397 39.91484391938795, 116.25822427104575 39.91560473167323, 116.25860369811619 39.91625944161212, 116.2587542555323 39.917155419248814, 116.25934281853155 39.91740573376146, 116.26017238150571 39.91837305892461, 116.26047989801104 39.918917913968706), (116.26066125935785 39.91908979976905, 116.26139721203687 39.91909890945397, 116.261493614772 39.91969117292497, 116.26209510309556 39.919864426634085, 116.26267162465133 39.920768643283466, 116.26283447817538 39.92087285535724, 116.26320152504029 39.92117941999886, 116.26371154759461 39.92198072026618))

围栏内里程(米):860.0

围栏外里程(米):61827.0

================>耗时85ms,points=1000

### 异常情况

由于模拟数据生成的随机性（不排除用户设置的围栏有交叉情况），会发生如下异常：

LINESTRING (116.25233147795146 39.90671657422486, 116.25329975573385 ..........................................

, 116.75523501649404 40.38680842299548, 116.75542104290876 40.38752711749191, 116.75557794481281 40.3880272337106, 116.7563739582496 40.38839729795829)

POLYGON ((116.25705727212875 39.907554807329966, 116.26239238833828 39.9171876545588, 116.2723211320915 39.92447819980439, 116.27886972069327 39.924660627590114, 116.27984327323371 39.93156232014855, 116.25705727212875 39.907554807329966))

lineStringContainsLineString=true

**polygonContainsPolygon=false**

================>开始....

com.vividsolutions.jts.geom.TopologyException: found non-noded intersection between LINESTRING ( 116.2723211320915 39.92447819980439, 116.27886972069327 39.924660627590114 ) and LINESTRING ( 116.27984327323371 39.93156232014855, 116.25705727212875 39.907554807329966 ) [ (116.27314128058926, 39.924501047151125, NaN) ]

at com.vividsolutions.jts.noding.FastNodingValidator.checkValid(FastNodingValidator.java:130)

at com.vividsolutions.jts.geomgraph.EdgeNodingValidator.checkValid(EdgeNodingValidator.java:94)

at com.vividsolutions.jts.geomgraph.EdgeNodingValidator.checkValid(EdgeNodingValidator.java:59)

at com.vividsolutions.jts.operation.overlay.OverlayOp.computeOverlay(OverlayOp.java:170)

at com.vividsolutions.jts.operation.overlay.OverlayOp.getResultGeometry(OverlayOp.java:127)

at com.vividsolutions.jts.operation.overlay.OverlayOp.overlayOp(OverlayOp.java:66)

at com.vividsolutions.jts.operation.overlay.snap.SnapIfNeededOverlayOp.getResultGeometry(SnapIfNeededOverlayOp.java:96)

at com.vividsolutions.jts.operation.overlay.snap.SnapIfNeededOverlayOp.overlayOp(SnapIfNeededOverlayOp.java:58)

at com.vividsolutions.jts.geom.Geometry.intersection(Geometry.java:1342)

at com.forestar.geometry.utils.GeometryUtil.intersectionGeo(GeometryUtil.java:122)

at test.GeometryTest.run(GeometryTest.java:108)

================>开始....

at test.GeometryTest.runLoop(GeometryTest.java:144)

at test.GeometryTest.main(GeometryTest.java:155)

com.vividsolutions.jts.geom.TopologyException: found non-noded intersection between LINESTRING ( 116.2723211320915 39.92447819980439, 116.27886972069327 39.924660627590114 ) and LINESTRING ( 116.27984327323371 39.93156232014855, 116.25705727212875 39.907554807329966 ) [ (116.27314128058926, 39.924501047151125, NaN) ]

at com.vividsolutions.jts.noding.FastNodingValidator.checkValid(FastNodingValidator.java:130)

at com.vividsolutions.jts.geomgraph.EdgeNodingValidator.checkValid(EdgeNodingValidator.java:94)

at com.vividsolutions.jts.geomgraph.EdgeNodingValidator.checkValid(EdgeNodingValidator.java:59)

at com.vividsolutions.jts.operation.overlay.OverlayOp.computeOverlay(OverlayOp.java:170)

at com.vividsolutions.jts.operation.overlay.OverlayOp.getResultGeometry(OverlayOp.java:127)

at com.vividsolutions.jts.operation.overlay.OverlayOp.overlayOp(OverlayOp.java:66)

at com.vividsolutions.jts.operation.overlay.snap.SnapIfNeededOverlayOp.getResultGeometry(SnapIfNeededOverlayOp.java:96)

at com.vividsolutions.jts.operation.overlay.snap.SnapIfNeededOverlayOp.overlayOp(SnapIfNeededOverlayOp.java:58)

at com.vividsolutions.jts.geom.Geometry.intersection(Geometry.java:1342)

at com.forestar.geometry.utils.GeometryUtil.intersectionGeo(GeometryUtil.java:122)

at test.GeometryTest.run(GeometryTest.java:108)

at test.GeometryTest.runLoop(GeometryTest.java:144)

at test.GeometryTest.main(GeometryTest.java:155)

================>开始....

com.vividsolutions.jts.geom.TopologyException: found non-noded intersection between LINESTRING ( 116.2723211320915 39.92447819980439, 116.27886972069327 39.924660627590114 ) and LINESTRING ( 116.27984327323371 39.93156232014855, 116.25705727212875 39.907554807329966 ) [ (116.27314128058926, 39.924501047151125, NaN) ]

at com.vividsolutions.jts.noding.FastNodingValidator.checkValid(FastNodingValidator.java:130)

at com.vividsolutions.jts.geomgraph.EdgeNodingValidator.checkValid(EdgeNodingValidator.java:94)

at com.vividsolutions.jts.geomgraph.EdgeNodingValidator.checkValid(EdgeNodingValidator.java:59)

at com.vividsolutions.jts.operation.overlay.OverlayOp.computeOverlay(OverlayOp.java:170)

at com.vividsolutions.jts.operation.overlay.OverlayOp.getResultGeometry(OverlayOp.java:127)

at com.vividsolutions.jts.operation.overlay.OverlayOp.overlayOp(OverlayOp.java:66)

at com.vividsolutions.jts.operation.overlay.snap.SnapIfNeededOverlayOp.getResultGeometry(SnapIfNeededOverlayOp.java:96)

at com.vividsolutions.jts.operation.overlay.snap.SnapIfNeededOverlayOp.overlayOp(SnapIfNeededOverlayOp.java:58)

at com.vividsolutions.jts.geom.Geometry.intersection(Geometry.java:1342)

at com.forestar.geometry.utils.GeometryUtil.intersectionGeo(GeometryUtil.java:122)

at test.GeometryTest.run(GeometryTest.java:108)

at test.GeometryTest.runLoop(GeometryTest.java:144)

at test.GeometryTest.main(GeometryTest.java:155)

### 异常分析

在上面的测试中，异常情况下多边形不能被其本身所包含。即：

正常情况:

lineStringContainsLineString=true

**polygonContainsPolygon=true**

异常情况:

lineStringContainsLineString=true

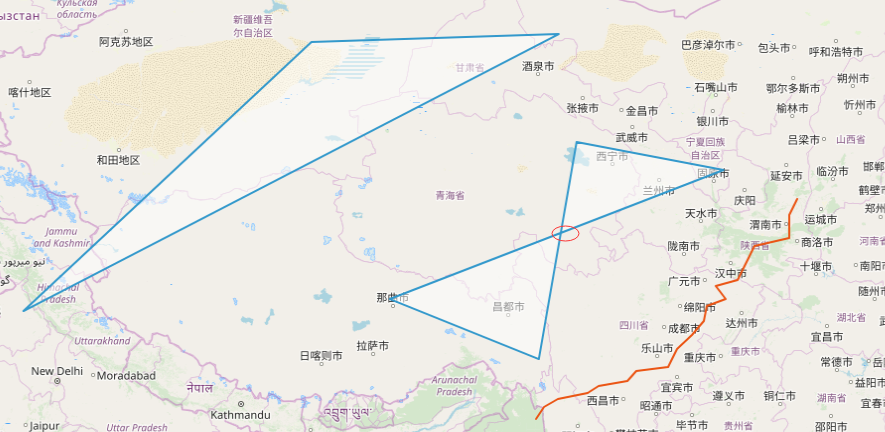
**polygonContainsPolygon=false**

可以在求交集之前做一次自身的包含计算，以避免异常的产生。

异常产生的提示是：发现LINESTRING之间的非节点交叉点



可以看到类似8字形的多边形，其交叉点并没有实际的点存在，这种图像就会产生异常。



### 避免异常

最简单的方式就是合理的使用Geometry的方法，也就不存在异常问题了。

**计算步骤：**

1. 判定多边形（围栏）是否对本身包含
2. 如果为假，则直接通过判断轨迹点是否在围栏内部进行分段计算。

注：分段计算是将轨迹区别运算，分别统计围栏内外的轨迹，效率不高，Geometry求交集方式的效率最高。