

室内地图路径导航引擎的设计与实现

罗少华, 罗红

(北京邮电大学计算机学院物联网中心, 北京 100876)

5 **摘要:** 在室内地图的背景下, 分析设计一个路径导航引擎的系统框架、对外接口和应采用的适当算法及其优化。引擎对外提供两点或多点最短路径查询服务, 并支持跨楼层查询。引擎通过 HTTP 协议接收用户路径查询请求, 并将返回结果封装在 XML 中。本论文详细介绍了上述功能在此引擎中的实现方法。

关键词: 室内地图; 最短路径; 导航引擎

10 **中图分类号:** TP311

Design and Implementation of a Route Engine for Indoor Map

Luo Shaohua, Luo Hong

15 (School of computer, Beijing University of Posts and Telecommunications, Beijing 100876)

Abstract: In the background of indoor map, this paper analyzes and designs a route engine's system framework, external interfaces, appropriate algorithms and the optimization of this system. Engine provides the service of shortest path inquires between two or more points which can on different floors. Engine receives user's path queries through HTTP protocol, and returns the response in the format of XML. This paper introduces the realization method of the above function of the engine.

20 **Key words:** indoor map; shortest path; route engine

0 引言

25 如今, 很多公司特别是互联网公司都对外提供地图服务, 地图已经成为人们生活中不可缺少的一部分。但分析发现这些地图都没有涉及到室内, 于是我们决定开发一个室内地图系统, 就是开发一个方便用户查看建筑物内地理信息的系统。这个系统将作为对室外地图一个很好的补充。比如你想逛某个商场, 你只能在百度地图上查到商场所在位置, 对于商场内的情形却不知晓。如果添加上该室内地图系统, 就可方便用户在未到商场之前, 先对商场内有一个了解。

30 本文要设计和实现的是这个室内地图系统的一个子系统: 室内地图路径导航引擎。对任何一个地图系统, 路径查询都应该是其提供的一个基础服务, 我们的系统也不例外。室内导航将有许多情形不同于室外, 首先一个建筑物会分多层, 室内道路一般都没有名称, 室内跨楼层搜索道路时肯定会经过楼梯等。本文将介绍, 针对室内地图特殊情况, 如何设计和实现一个快速有效的路径导航引擎。

1 路径导航引擎的设计

路径导航引擎的设计总体来说要先设计系统框架, 确定系统与数据库的接口以及与服务访问者的接口, 其次要为实现引擎选择和设计合适的算法, 以使引擎有高的服务效率。

作者简介: 罗少华(1985-), 男, 硕士研究生, 主要研究方向: 计算机网络, 室内地图等。E-mail: luoshaohua.cn@gmail.com

通信联系人: 罗红(1968-), 女, 教授, 现代通信网络技术及物联网, 通信软件。E-mail: luoh@bupt.edu.cn

1.1 系统框架设计

40 室内地图系统的路径导航子系统如图 1，路径引擎运行于中间的服务器上，使用地图数据库提供的楼层地理数据，为它的服务访问者（如：室内地图网站服务器等）提供路径查询服务。引擎与数据库间的接口使用 ODBC，这样可以方便适应以后数据库软件的变化^[1]。引擎与服务访问者之间的接口设计为 HTTP+XML 方式。HTTP 能方便服务访问者构造请求，也能方便开发者测试引擎提供的服务。HTTP 的正文部分是路径查询的结果，用 XML 描述，
45 这样可以提高接口的可扩展性^[2]。

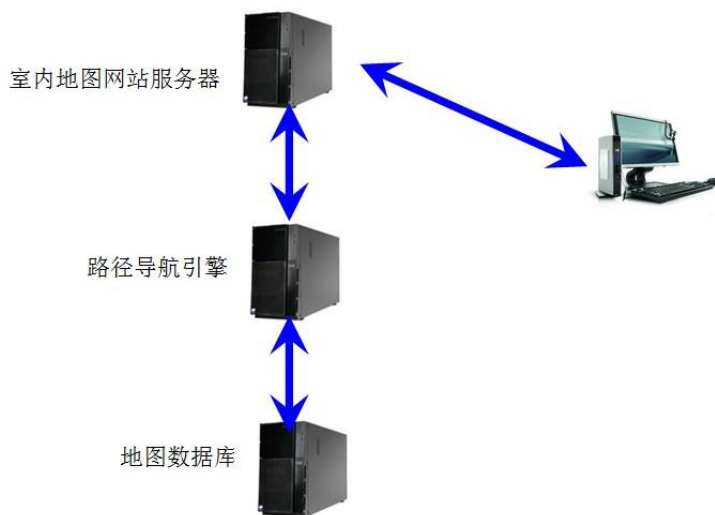


图 1 路径引擎系统结构图
Fig. 1 The structure of route engine

1.2 引擎算法设计

50 在引擎的实现中，涉及三个主要问题需要选择和设计适当算法。1、如何确定两点间最短路径；2、如何确定多个点的拜访顺序，以使总路径最短；3、如何提供跨楼层路径搜索。

1.2.1 两点最短路径

对比各种两点最短路径算法，最后选择 Dijkstra 算法。虽然在计算两点最短路径上，A* 算法最快，但其不好之处在于只计算了这两点的最短路径，而 Dijkstra 可以一次计算一点到所有点的最短路径，这些额外的结果都可以保存在一个表里（这个表的 i 行 j 列记录 i 点到 j 点的最短路径）。开始表为空，一次查询，就可以填入一行。引擎面对大量查询，如果查询结果已在表中，直接取出；如果没有，利用 Dijkstra 可以更新表中的一行。这样算法计算结果可以复用，提高了效率。Bellman-Ford 算法虽然也是一次计算一点到其他所有点最短路径，但效率低。两点最短路径另一常用的算法 Floyd 是一次计算任意两点最短路径^[3]，这会
60 使引擎运行初期路径查询响应缓慢。

1.2.2 多点最短路径

引擎提供遍历多点的最短路径查询，这是一个 TSP 问题，不存在多项式复杂度的最优解，常用的近似解法有模拟退火算法、遗传算法等^[4]，但这些算法效率低，不满足系统快速响应需求。最后选择简化这个 TSP 问题，要求游客开始一直往一个方向走，到头后，再返回，即遍历路线为一个 U 型。这样，这个问题就有了动态规划的多项式复杂度的最优解。
65

1.2.3 跨楼层路径搜索

如果用户的最短路径搜索是跨楼层的，那么我们可以根据如下常识，即路径跨楼层必然要经过楼梯，将搜索分成两步进行。第一步只考虑楼梯点，而忽略其他所有道路节点，这样图就大大简化。在此仅由楼梯点构成的图中，先确定从起点到终点要经过哪几个楼梯。第二步再确定楼梯与楼梯之间如何走，这时只有两种情况：一、两楼梯点不在同一层，那它们必然直接相通，上下楼梯即可；二、两楼梯点在同层，这就转化成非跨楼层路径搜索问题。

2 路径导航引擎的实现

引擎总体按功能划分成四个模块：外部请求接收模块、地图数据加载/释放模块，路径生成模块和路径数据输出模块。其中地图数据加载模块/释放模块分为楼层道路数据加载/释放子模块和建筑物楼梯数据加载/释放子模块。最为重要的模块是路径生成模块，它分如下四个阶段：确定游客拜访顺序、确定经过的楼梯顺序、确定经过的 node 点顺序，最后确定经过的 point 点顺序。

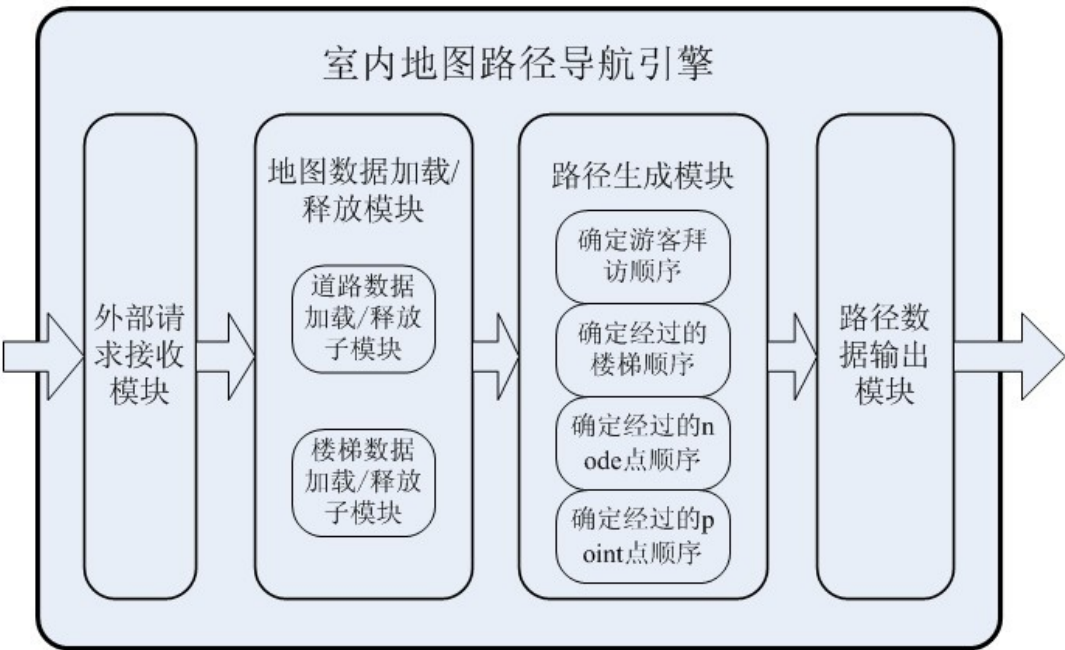


图 2 路径引擎系统流程图
Fig. 2 The flow chart of route engine

2.1 外部请求接收模块

外部请求接收模块是用来接收并解析用户的路径查询请求。它定义了引擎对外提供的接口。接口采用 HTTP+XML，用户的查询请求被封装在一个 HTTP GET 请求报文中。例如用户请求查询：在建筑物 ID 为 8 的楼内，从一层（Floor1）坐标点（10 米，20 米）如何到达二层（Floor2）坐标点（80 米，90 米）。URL 请求为：

```
http://59.64.159.93:5555/xml?2&10,20,Floor1&80,90,Floor2&8
这时类似如下的 HTTP GET 报文将被发往路径导航引擎：
GET /xml?2&10,20,Floor1&80,90,Floor2&8 HTTP/1.1
Host: 59.64.159.93:5555
Connection: keep-alive
User-Agent: Mozilla/5.0 (Windows NT 6.1) AppleWebKit/534.30 (KHTML, like Gecko)
```

Chrome/12.0.742.122 Safari/534.30

Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8

Accept-Encoding: gzip,deflate,sdch

95 Accept-Language: zh-CN,zh;q=0.8

Accept-Charset: GBK,utf-8;q=0.7,*;q=0.3

本模块在收到该请求后，就开始解析请求字符串，在本例中就是：
2&10,20,Floor1&80,90,Floor2&8，解析结果存入相应结构体，以备后续模块使用。

2.2 地图数据加载/释放模块

100 当外部请求接收模块解析好用户请求的数据后，下一步就是分析为了得出用户查询的路径，有哪些地图数据需要载入内存。比如用户要查询某楼一层某点到二层某点的最短路径。我们就需要将此楼一层二层的道路信息从数据库中载入内存，以供后续计算。

由于内存的容量是有限的，而数据库中的存储了许多建筑物的楼层，所以不可能将所有楼层都加入内存。地图加载是一个耗时的过程，所以一旦某层地图加载到内存中，即使其暂时不使用时，也不应立即释放，因为之后的路径查询可能会再用到此楼层数据。为了缓存这些加载过的楼层数据，我们在引擎中设计了一个地图池。地图池的大小固定，这样就限制了引擎消耗的最大内存。

105 当某个楼层需要被使用时，先查询地图池中是否已经有了该楼层。如果有，就可直接使用，无需加载；如果没有，准备加载地图。加载时，先看地图池有没有空闲位置，如果有，
110 将需加载楼层载入此空闲位置；如果没有，用一定的机制淘汰掉地图池中一个楼层（淘汰机制见 3.2 节），然后将需加载楼层载入此淘汰后留下的空位置。

2.3 路径生成模块

整个路径生成模块分四个阶段：确定游客拜访顺序、确定经过的楼梯顺序、确定经过的 node 点顺序，最后确定经过的 point 点顺序。

115 2.3.1 确定游客拜访顺序

首先确定游客拜访顺序。就是如果路径请求中，游客要拜访几个拜访点，那么我们首先要确定的是这个拜访顺序，以何种次序拜访这几个点，总路径能最短。作为一个特例，拜访点的个数是 2，那么拜访顺序就是唯一的，这种情形下就可以跳过这个阶段。如果拜访点数大于 2，拜访顺序使用 1.2.2 节讨论的简化的 TSP 问题的解决算法来确定。

120 2.3.2 确定经过的楼梯顺序

确定了拜访顺序后，我们开始依次确定连续的两个拜访点之间的最短路径问题。不失一般性，如果这两个拜访点在不同楼层，那么我们需要确定这两拜访点间的最短路径要通过哪几个楼梯。确定的算法参见 1.2.3 节。作为一个特例，此二拜访点在同一个楼层，这种情形下就可以跳过这个确定经过的楼梯顺序的阶段。

125 一旦确定了楼梯的通过顺序，那么我们就开始确定在这个楼梯序列中，连续出现的两个楼梯之间的路径如何走。这时只会有两种情况：1、这两楼梯不在同一层，这时情况最简单，两楼梯必然之间相通，通过这两个楼梯只需上下楼梯即可；2、这两楼梯在同一层，这时问题变成在同一层中确定两点最短路径问题。这个问题交给下一阶段处理。

2.3.3 确定经过的 node 点顺序

确定经过的 node 点顺序的阶段用来确定同一层中两点最短路径。我们抛去道路的具体形状，一条没有分叉的道路（如果分叉，每个分叉又算是一条道路）可以被抽象成两个端点和一个道路长度。我们把这个端点称为 node 点。node 点和 node 点之间的道路长度就构成了一个有权无向图。我们使用 1.2.1 节讨论的方法在这个有权无向图上得出指定两点之间的最短路径。

2.3.4 确定经过的 point 点顺序

确定经过的 node 点顺序后，为了输出最后的路径，我们就需要考虑每条道路的形状，依次输出每两个 node 点之间的 point 点坐标，这样就得出了完整的导航路径。

2.4 路径数据输出模块

路径数据输出模块又一次涉及到引擎的对外接口。它采用 HTTP+XML 方式，即先输出一个 HTTP 200 OK 头，再输出一个 XML 序言<?xml version="1.0" encoding="gb2312"?>。接着用如下 XML 结构输出路径上的各点。

```
<POINT unit="米">
  <X>10.00</X>
  <Y>20.00</Y>
  <NAME>起点</NAME>
  <FLOOR>Floor1</FLOOR>
</POINT>
```

3 路径导航引擎的优化

3.1 道路数据预处理

路径导航引擎是根据数据库中的有关楼层的地理信息数据来进行计算，以生成路径导航结果反馈给有路径查询需求的客户。为了加快路径导航引擎对用户的响应速度，我们可以预先对这些数据进行相应的预处理，预处理的结果也保存在数据库中。这样，当某楼层的地理数据需要加载到内存中供路径导航引擎处理时，可以减少某些数据处理过程，从而节省时间。

引入预处理的优点是：一、避免引擎运行时计算这一过程，提高了引擎响应速度。二、由于预处理结果是一静态数据，只和数据库中有关楼层地理信息数据有关，预处理一次后，可重复利用。然而、引入预处理的缺点是：系统的复杂度和出问题概率提高了。预处理结果虽是一静态数据，但这是相对的。如果楼层地理信息数据发生了变化，那么针对这些变更数据的预处理就失效了，这些预处理应该被重做。可见，预处理增加了数据库中数据不一致性的风险。

本引擎对楼层地理数据的预处理是统计出每层道路条数，道路节点数等信息，以加快楼层数据载入内存。如果不进行预处理，这些信息也需要在加载时计算出来，已决定申请分配内存的大小。

3.2 引擎内存管理

本论文 2.2 节中讨论的地图数据加载/释放模块中，存在一个内存的分配和释放问题。地图数据在数据库中数量很大，并且由于程序的时间代价和空间代价是一对共轭量，我们需要某种策略在引擎的内存消耗上和引擎的响应速度上有一个平衡。

引擎设计了一个地图池，目前池大小设定为 100。也就是说最多同时有一百层楼的道路信息被载入内存。由于每层楼中道路条数不尽相同，所以地图池中的每一项不应该设置为固定大小，而应该动态分配内存。

如上介绍的是楼层信息载入策略，下面介绍释放策略。当地图池满后，有新楼层需要载入地图池，我们就需要淘汰一个近期内使用频率最低的楼层。为了进行这种淘汰，我们为每个载入地图池的楼层设置了一个标记其使用频率的计数器。此计数器在对应楼层载入地图池时设初值为 64，以后这个楼层每次被使用到时，计数器加一。在每次需要淘汰掉一个楼层时，淘汰计数器值最小的楼层，并把没有淘汰的所有楼层对应的计数器减半。淘汰逻辑见如下函数：

```
int get_free_floor ( MAP *floor , int l_max )
```

```
{
```

```
    int i , ret , min_access = INF ;
```

```
    //floor 表示地图池， l_max 是地图池的大小
```

```
    //如下代码表示，如果地图池有空闲项，就直接返回该空闲项下标
```

```
    for ( i = 0 ; i < l_max ; i++ )
```

```
        if ( NULL == floor[i].node )
```

```
            return i ;
```

如果没有空闲项，如下代码选择 access_frequency 值最低的项淘汰。并将未淘汰的使用计数器减半，以防止计数器不断增大。这相当于隔一段时间让计数器进行衰老，以降低过去的时间中该楼层访问次数对该计数器值的贡献。这样能让下次淘汰时，淘汰掉最近不常使用的楼层。

```
    for ( i = 0 ; i < l_max ; i++ )
```

```
    {
```

```
        if ( min_access > floor[i].access_frequency )
```

```
        {
```

```
            min_access = floor[i].access_frequency ;
```

```
            ret = i ;
```

```
        }
```

```
        floor[i].access_frequency /= 2 ;
```

```
    }
```

```
    return free_floor ( floor , ret ) ;
```

```
}
```

4 结束语

本文介绍了如何设计和实现一个实用的室内地图路径导航引擎。引擎对外接口设计成 HTTP+XML 方式，这样能方便访问和测试又能为将来提供很好的扩展性。在引擎所使用的算法上考虑了响应速度和计算结果复用程度，并使用了预处理和缓冲地图池等优化策略。引擎对外提供的服务效果展示见图 3。



图 3 路径引擎响应例图

Fig. 3 An example of route engine's response

210 只要使用 HTTP 协议就能访问该引擎,该引擎除了服务室内地图系统,还能在其上面进行二次开发。目前这个导航引擎子系统所在的室内地图系统部分功能已经在百度地图上上线。

215

[参考文献] (References)

- 220
- [1] 沈岚, 范丰龙, 李晓红. 数据库技术及应用教程[M]. 北京: 清华大学出版社, 2011.
 - [2] 粟松涛. XML 程序设计[M]. 北京: 清华大学出版社, 2001.
 - [3] Thomas H C, Charles E L, Ronald L R, Clifford Stein. 算法导论[M]. 潘金贵, 顾铁成, 李成法, 叶懋. 北京: 机械工业出版社, 2006.
 - [4] 张焱, 华斌. 求旅行商 (TSP) 问题的几种改进遗传算法的比较分析[D]. 天津: 天津财经大学, 2007.