# **Railroad Center 设计文档**

## **设计背景**

当地的通勤铁路为新西兰的许多城镇提供服务。出于货币方面的考虑，所有这些投资都是单向的。也就是说，从开泰到因弗嘉吉的路线并不意味着存在从因弗嘉吉到开泰的路线。事实上，即使这两条路线碰巧都存在，它们也是不同的，而且不一定是相同的距离。

## **整体设计**

整个应用分为两个部分，Client模块和Core模块，当项目变得复杂或者需求更多时，可以扩展为微服务形式，这里仅仅做一个演示作用；表现类似Spring Cloud 这种微服务设计。

1. Core模块作为整个应用的核心，需要做到的是完全独立，因此，我从中间件和开源的角度来做设计，它的实现不依赖与Client模块，能够单独放到任何环境种单独执行。
2. Core模块中，我还模拟了具体的插件实现，把算法实现和模块又做了分离，能够实现热插拔的机制根据场景需求进行算法替换，这里紧紧做了、Java SPI机制的简单实现。
3. Client模块作为交互的角色，设计成用户可输入的形式，通过一个外部文本配置，可自定义构建一个完整的图，这个图需要保证正常的逻辑，否则如果都是不可达点则会让整个应用变得没有意义。
4. Core模块设计上，使用了策略模式，这是一种面向接口的实现方式，上层调用完全不关心具体实现的差异。通过策略模式很好的将Client和Core的调用解耦。

## **核心实现**

算法部分，根据常见的路径算法，我又做了详细的划分，总共5种，分别为迪杰斯特拉算法，广度优先算法，深度优先算法，洪泛算法，贝尔曼福特算法，贝尔曼福特优化算法。这里仅作热插拔的演示，后三种，没有做过深入研究，暂不做讨论；广度优先算法和深度优先算法相对于迪杰斯特拉算法来的简单，由于时间关系，也不做实现。我们仅作迪杰斯特拉算法的实现。算法的基本思想就是每一步都需要找与当前结点最近的点，不断迭代，最终获取最优路径解。

在代码设计种，我总结抽象了两个对象Edge和Node。一个Edge，里面包含一个Node起点和一个Node终点，还有一个这两个点的距离，即图算法中的权重，权重越小表示两点之间越近，反之越远。逻辑实现上，可能存在两点之间并不相邻的情况，这种情况下，并没有权重，默认为0，并且如果是可达路径，则表现为Edge的一个数组集合，一个Edge代表一个步子step。

抽象的Edge对象与实际用户认识的转化是通过一个简单的三元组来实现的，如AB5，表示A点到B的路径长度为5个单位。而A-B-C这种，表示成一个route路径，在具体逻辑中，表现为两个Edge，AB0，BC0（之所以为0是应该在该场景下，用户并不知道当前路径的权重，我们需要路径探索完成相应的路径计算）。这在使用文档中有具体距离。

在接口设计上，总共实现了如下接口：

1. init初始化接口设计，完整图的构建，和参数的初始化。

2. toMatrix图形转化接口，将实体世界的边和权重转化成一个二维矩阵，这个矩阵可能是稀疏矩阵等，在迪杰斯特拉算法实现中，这是一个可达矩阵。

3. getShortestDistance2AllNodes获取最短路径接口，这里为了效率，我们直接获取一个点到所有点的最短路径结果集，共上层调用，实现具体的业务逻辑。

4. getSpecialRouteDistance 获取指定路径的长度。

5. getRoutesBetweentTwoNodes 获取两个点之间的所有可达路径。