北京邮电大学 数据结构课程设计	产品名称	产品版本	密级
	Covid-19 旅行模拟系统	V1.0	公开
	定稿日期： 2020 年 7 月 9 日		**共 11 页**

Covid-19 旅行模拟查询系统 V1.0

各模块设计说明

文 档 作 者： 史嘉程

学　　院： 计算机学院

专　　业： 计算机类

班　　级： 2018211318

指 导 教 师： 杨　俊老师

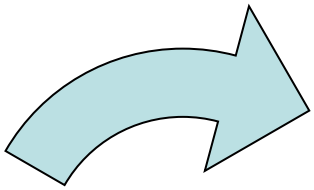
1 总体概览

本系统采用分层思想进行模块化设计，整个系统分为以下两个层次：

a） **用户交互层**

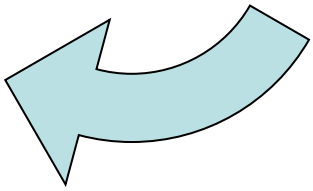
b） **应用服务层**

这两个层次之间相互提供服务。其中应用服务层实现实时响应用户所提出的需求并负责主要的运算工作；用户交互层用于展示用户界面、接收用户提出的需求、实时输出计算结果，展示模拟旅行信息等功能，以图形化用户界面的方式实现。



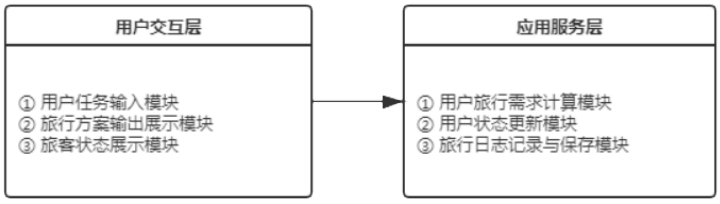
用户交互层

应用服务层

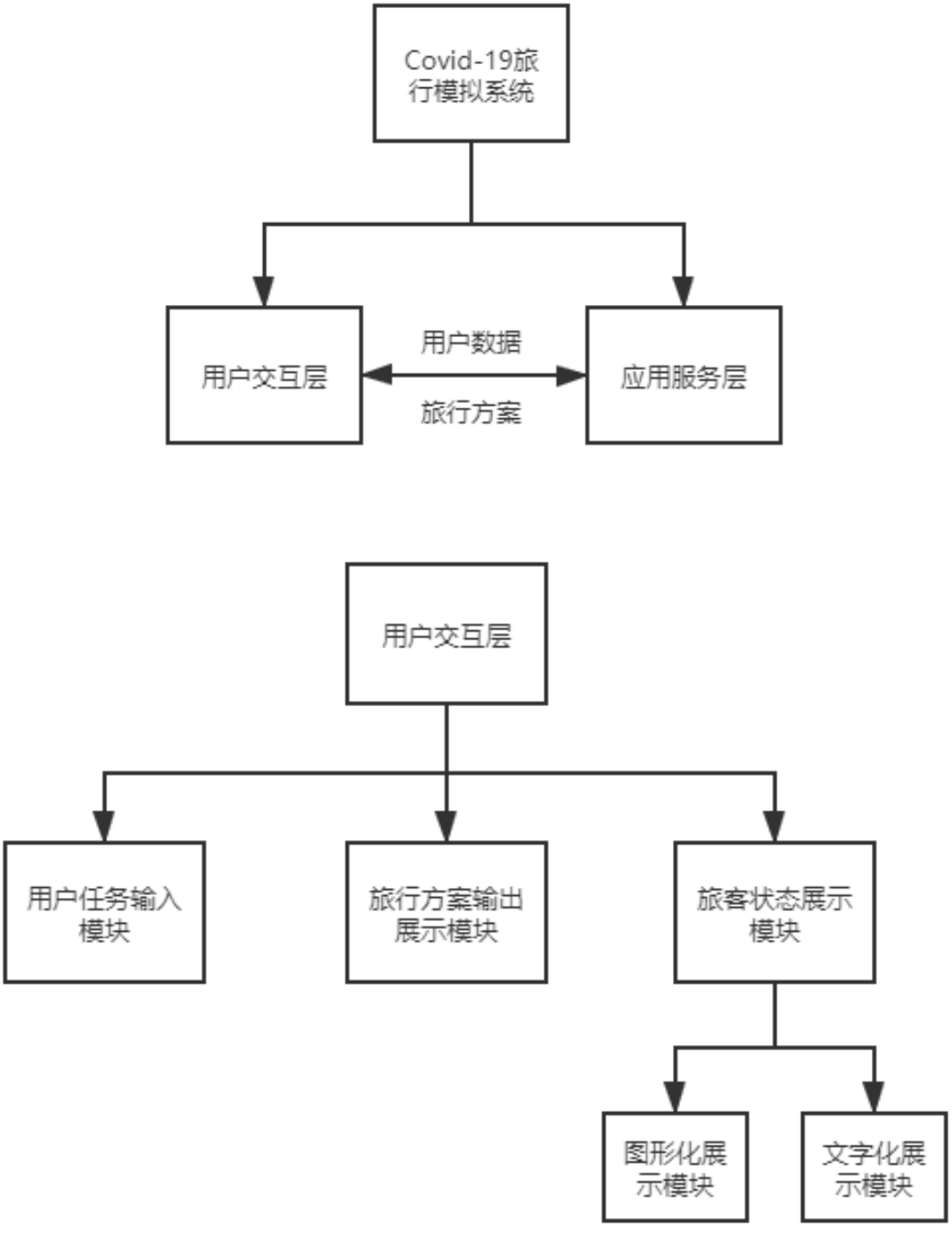


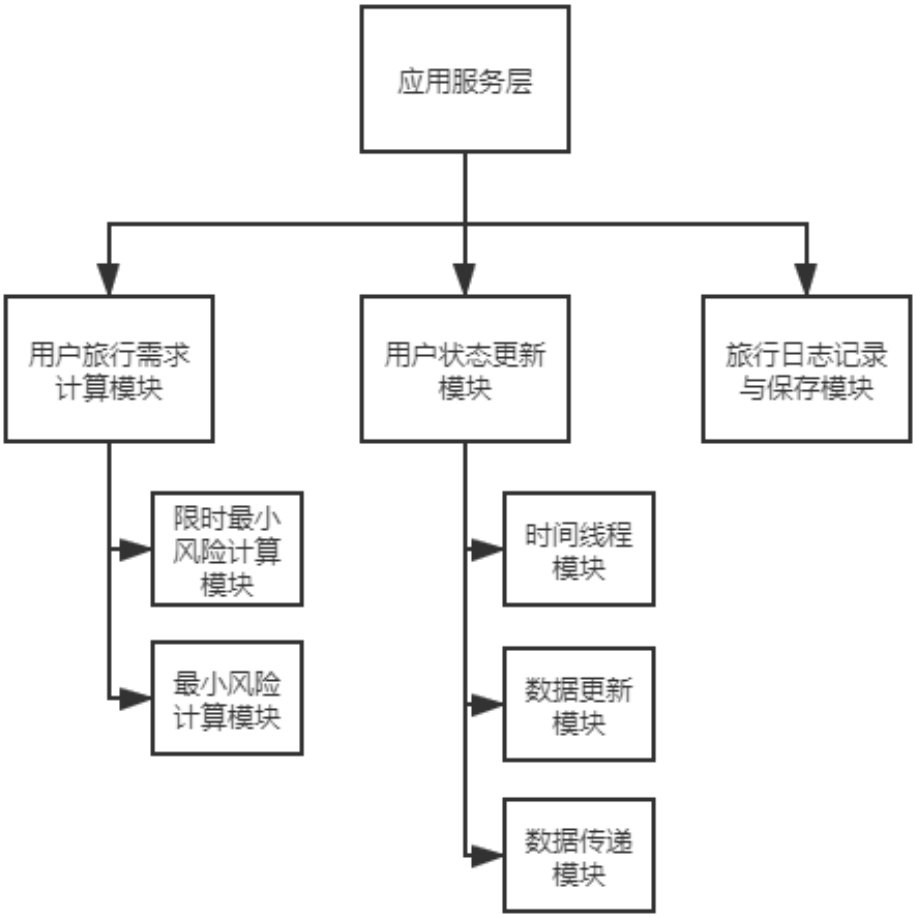
软件体系结构：系统采用双层结构，用户从交互层输入任务，交互层将任务数据传递给应用服务层，应用服务层计算结果并且更新旅客信息，并将数据传递回用户交互层进行输出。数据在两个层次之间形成循环流动。

用户交互层和应用服务层的程序不同，其中面向用户的程序为用户交互层。用户交互层主要拥有用户界面、旅行方案输出与展示、当前状态查询等模块；而应用服务层主要提供用户旅行需求计算、用户数据管理、用户状态更新与查询、系统维护及用户日志记录与保存等功能模块。



上图中的模块为 Covid-19 旅行模拟查询系统所具有的功能模块。这些模块是系统基本模块，也是满足用户旅行需求、为用户提供良好交互的必要支撑。上述的是个模块中有部分模块由子模块构成，系统的完整模块划分如下图所示：



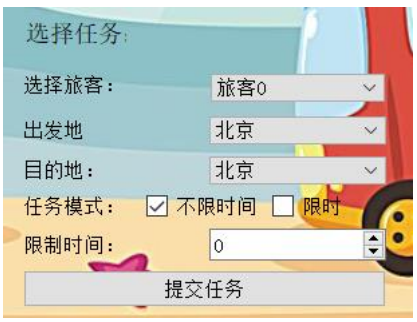


2 用户交互层模块设计说明

2.1 用户任务输入模块

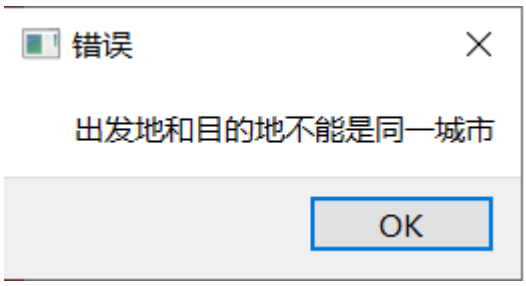
本模块被融合入整个用户交互界面之中，目的是为了帮助用户方便快捷地输入任务，并且给予相应的反馈。

用户可以直观地选择旅行任务的所有信息，如出发地、目的地、旅行方案、限制时间等，提交任务后，数据将传递给应用服务层进行处理，并生成最合理的旅行方案。

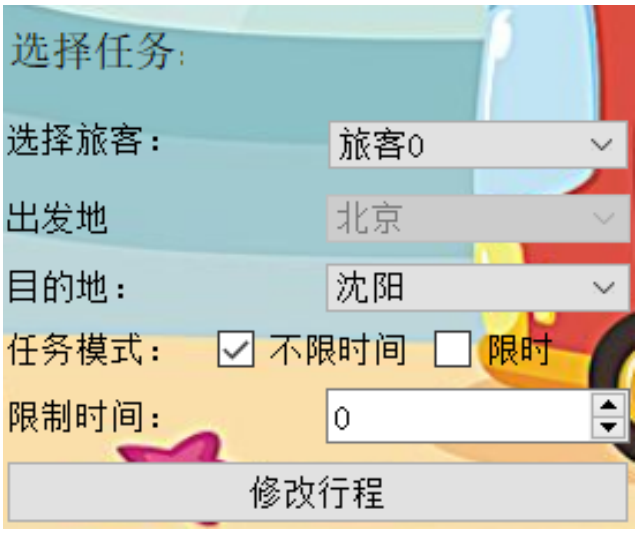


此外，输入模块还对一些常见的不合理的输入情况进行了反馈和规范，如出发地与目的地相同、限制时间不合理、该旅客仍在交通工具上，无法更改目的地等情况。

（1）出发地和目的地相同，弹窗提示



（2）旅客已在旅行途中，只能以当前城市为出发地

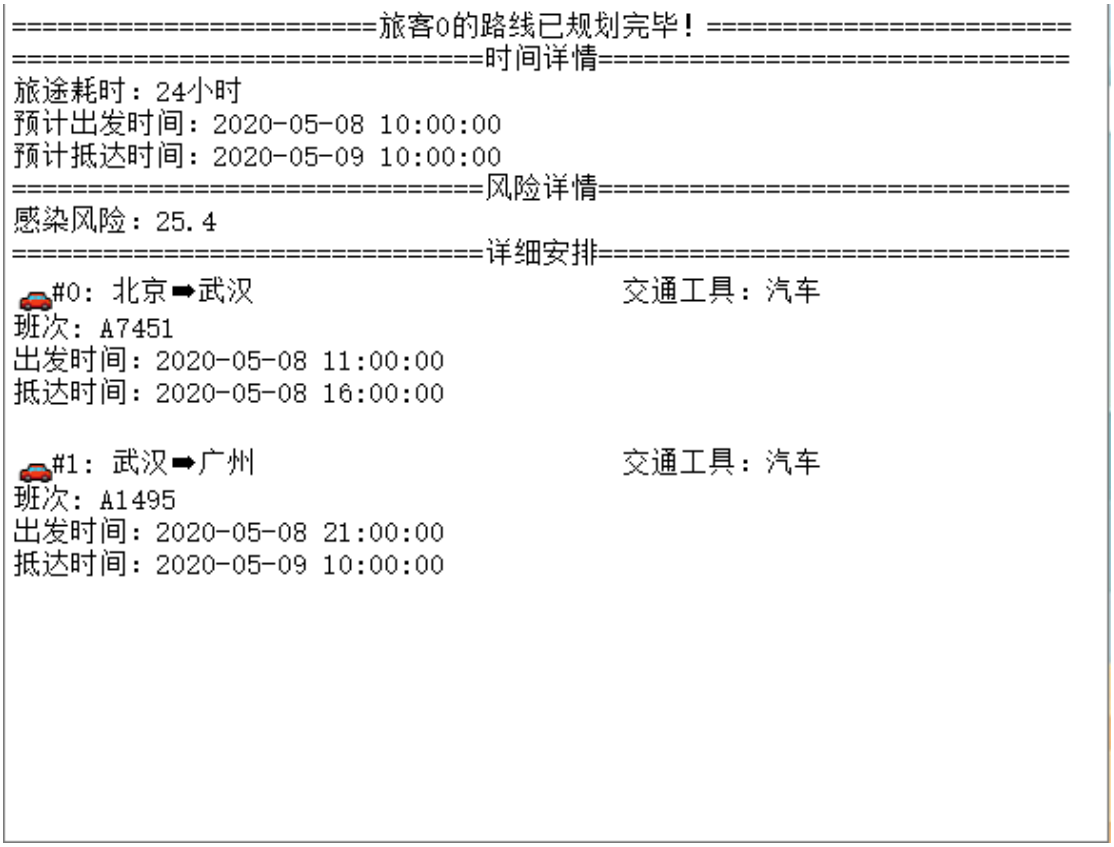


（3）限制时间不合理，弹窗提示

2.2 旅行方案展示模块

本模块是直观地向用户反馈旅行方案的模块，通过 pyqt5 的文本框实现，展示了包括旅途耗时、预计出发时间、预计抵达时间、感染风险、旅途详细安排等关键信息。只有在提交新的任务时该模块才会更新，应用服务层将计算好的方案

信息传递给用户交互层，由本模块提取方案中的关键信息，转换为字符串的形式，并通过本模块输出。为了使输出结果更为直观，我在输出字符串中添加了一定的符号和图标。



2.3 用户状态展示模块

为了更加直观、合理地展示用户当前状态，我采用了地图加文字两种展示方式相结合的形式，并得到了非常好的展示效果。我通过 Pyecharts 图形库，通过使用现有的中国地图的 HTML 文件，结合了前端知识，达到了不错的可视化效果。为了将其与文字展示模块合并，我使用了 pyqt5 库，并通过多线程解决了卡顿问题。

2.4.1 文字化展示模块

该模块主要用于展示旅客当前位置、行为以及接下来的行为安排。在旅客旅行方案的数据结构中，我采用了队列的思想，每条行程信息即为队列中的一项。当用户完成一段行程，则该行程“出队”，并更新相关信息。这种数据结构保证了每一时间点都能输出最准确的状态信息。

2.4.2 图形化展示模块

在图形化展示模块，我主要使用了 Pyecharts 这个库达到更加完美的地图显示效果。Pyecharts 是由百度开发的，可视化功能非常强大的 JS 图形库。用户交互层通过应用服务层获取用户当前状态，筛选关键信息，利用该 Pyecharts 类库生成 HTML 文件，通过时刻表刷新 HTML 显示文件，达到图形化显示当前位置的效果。另外，PyQt5 提供了一个简单的浏览器，可以在窗口中显示 HTML 文件。由于 HTML 文件的其动态的展示效果和多样的图形元素，图形化展示模块给用户带来良好的视觉体验。

3 应用服务层模块设计说明

3.1 用户旅行需求计算模块

本模块是该系统的核心模块，通过从用户交互层获取用户的旅行需求，提取相应的需求信息，调用正确的方案模块，产出用户的旅行方案。该模块为旅行查询系统提供了算法层面的支持，为实现旅行模拟和查询打下基础。

本模块由两个子模块构成，分别对应两个策略的算法，因为两个策略采取的算法思想一致，仅在算法的最后一步（修枝）上有区别，因此放在一起进行介绍。

3.1.1 风险最少算法模块

本模块用于寻找风险最少的旅行策略，是本系统的核心算法。本模块主要由函数 get_best_route 和 minimize_risk_algorithm 组成，所对应的算法为深度优先搜索算法。

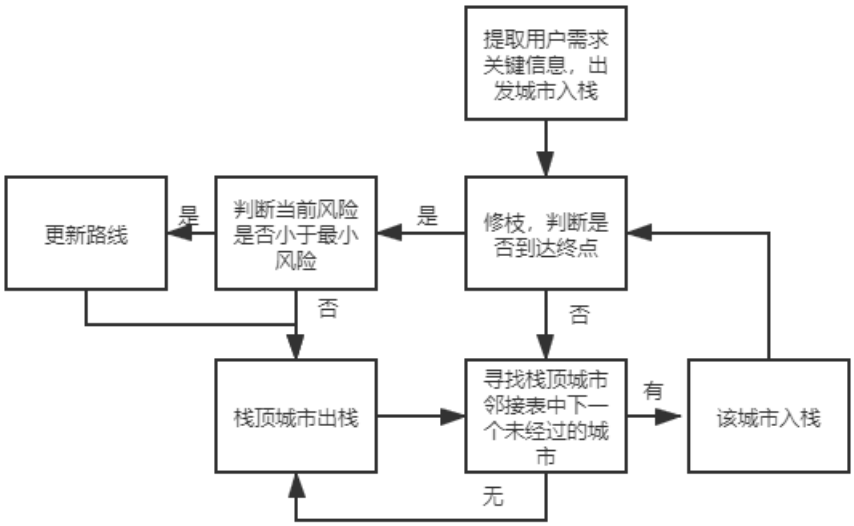
【调用者】用户交互层；

【模块输入】用户需求（包括出发城市、目的城市、出发时间限制、以及旅行策略）；

【模块输出】运算结果（包括详细的班次顺序、总风险、总时间等）；

【模块算法】深度优先搜索（DFS）算法；

【算法步骤】



① 根据时刻表对图的点和边数据进行提取和抽象，得到城市之间的邻接表；

② 提取用户需求，获取算法所需要的关键信息，出发城市入栈

③ 修枝，判断栈顶城市是否是目的城市，如果是，跳转到④；如果不是，跳转到 ⑤

④ 如果当前方案风险小于全局最小风险，则更新。栈顶城市出栈。转到⑤

⑤ 在当前栈顶城市的邻接城市中寻找下一个未经过城市，如果有，该城市入栈，转到③；如果没有，栈顶城市出栈。转到⑤

算法伪代码：

```
def minimize_risk_dfs(depart_time, dep, final_dest, city_graph, time_
limit, total_time, total_risk, route,
                      arrived_city):
    nonlocal min_risk, min_risk_route, min_risk_time
    trans_weight = {'飞机': 9, '火车': 5, '汽车': 2}
    # 修枝，加快运行速度
    if total_risk > min_risk:
        return
    # 限时模式修枝
    if total_time > time_limit:
        return

    # 到达终点
    if dep == final_dest:
        if total_risk < min_risk:
            更新最小风险的方案信息;
        return

    # 深度搜索每一个没去过的城市
    # 获取每一个当前城市可以到达的城市
    trans_list = city_graph[dep]["transportation"]
    for trans in trans_list:
        # 判断该城市有没有去过
        if trans['destination'] not in arrived_city:
            判断是当天出发还是到达城市的第二天出发，更新信息的计算方法有不同;

            # 在到达城市的同一天出发
            # 等待时间不会超过24h，不然永远无法到达
            if 同一天出发:
                更新当前方案的各种信息;
                 继续搜索没去过的，该城市可达的城市;
                 arrived_city.remove(trans['destination'])
            # 等到第二天再出发
            else:
                更新当前方案的各种信息;
```

```
                继续搜索没去过的，该城市可达的城市;
                arrived_city.remove(trans['destination'])
```

【算法分析】

假设城市的总数为n，每个城市的邻接城市的数量为m，则 DFS 算法的时间复杂度为$O(n+m)$；另外，本算法的空间复杂度为$O(n^2)$。对于用户限制旅行时间的情况，本算法也同样适用，但需要在搜索到可达到目的地的方案时进行判断，如果当前方案的总时间大于限制时间，则舍去该方案，剩余部分则完全一致，故不再进行详细说明。

本算法模块的实现经过大量测试，因为数据量的限制，深度搜索产生的时间延迟非常小，并且通过该算法得到的解一定是最优解。经过了不断的优化与改进（修枝），其延迟时间相对于系统的推进时间可忽略不计。

3.2 用户状态更新模块

该模块用于推进全体旅客的旅行模拟过程，是用于处理旅客的旅行逻辑，如基本的系统时间、旅客到站处理、旅客换乘处理、旅客结束行程处理等。当处理完这些逻辑后，本模块将会将更新后的旅客信息传递给用户交互层，以便实时输出旅客信息。其主要分为三个子模块：时间模块、数据更新模块、数据传递模块。

3.2.1 时间模块

该模块复杂掌控整个系统的时间推进。我调用了 Thread 线程库的功能，将时间模块设定为一个单独的线程，每次经过一个特定的时间，则通过数据传送子模块向用户交互层提交用户更新信息。

3.2.2 数据更新模块

为了能够实时得到旅客当前状态，每次系统时间向前推进的同时，数据更新模块将根据所有旅客的旅行方案队列来更新旅客信息。比如更新旅客所在城市、是否在交通工具上、是否抵达目的地等。

3.2.3 数据传递模块

本模块是在双层结构中起到桥梁作用的关键模块。我采用了 pyqt5 中的信号与槽的思想，将时间模块的更新作为信号，向用户交互层传递用于展示的相关信息，如当前时间点下的旅客信息、时间信息、旅客剩余旅行计划等用于用户交互层展示界面的信息。

3.3 旅行日志输出与保存模块

该模块用于更新旅行日志。需要保存用户日志的主要分为两类信息。其中一类信息是用户操作，它保存了用户选定的旅行方案。另一类信息是实际旅行信息，它表示了用户在某一时刻旅行中所处的关键状态。这部分主要功能是记录与保存，便于用户追溯曾经发生过的关键事件。