目标客户:

1. 本身有送餐服务，但是忙的时候送不过来的餐馆
2. 本身没有送餐服务，但是每天可以产生外面的餐馆

业务场景：

商家在他们的客户端下单-> 会出现等待时间，价格等信息->商家确认，正式下单

订单信息自动经过我们的计算逻辑，推送到送餐员的终端里

送餐员需要严格按照终端的指示信息进行工作，主要工作为取餐和配送

在商家和送餐员这两端都会出现一些类似以下的意外，

送餐员到达餐厅的时候餐馆没有准备好餐

送餐员在路上出现意外（抛锚或者有事故）

送餐员配送失败等

以上情况，送餐员只需要在自己的终端上更新状态（包括成功配送和出现意外），然后不需要对意外情况作处理，就继续执行客户端分配的下一个任务。

调度人员会专门对所有的意外情况进行处理，尤其是电话联系

重点：

以下任何事件都会重新计算并更新该区域内所有送餐员的任务（他们的任务是实时更新的）：

商家下单；商家要求重新配送

送餐员取到餐；送餐员取餐失败；送餐员路上出状况；送餐员配送失败（客人不在家）；配送成功

调度人员重新分配任务给别的送餐员

每次重新计算需要考量的信息：

当前任务能否插入到某个送餐员的任务列表里面，给谁的效果最理想，又如何和既有的任务分配先后顺序（因为每一个任务分为取和拿两个步骤）；要计算最佳路径；（未来可能要考虑路况，暂时不用考虑）

已经开始配送的任务不应受到影响

新加入任务不能导致之前任务的配送时间超过限制（设定的一个参数）

司机终端

调度

后台处理

商家终端

以上是数据流的一个简单说明；Firebase是一个实时数据库，可以理解为只存储着数据，使用它的原因是因为一旦Firebase的数据有变化，可以实时更新到司机和商家的移动终端上。

以下是每一个数据流向的解释（目前仅限涉及算法处理的部分，未来会有更细化的解释）：

司机终端 -> 后台

以下情况司机会向后台发送更新请求，后台会做相应的处理：

取完单，取单失败，配送完成，配送失败，出现意外

发送的信息包含司机的坐标（Lat,Lng）时间

餐馆 -> 后台

以下情况餐馆会向后台发送更新请求，后台会做相应的处理：

获取等待时间，下单，（意外后）再次配送, (意外后)取消 发送的信息包含餐厅的坐标（Lat,Lng），客人的坐标，时间

调度 <->后台

出现意外的情况下，调度可以自动也可以手动把一个任务安排给另外一个司机

自动的情况下，发送的信息有旧司机的坐标，时间

手动的情况下，发送的信息有旧司机的坐标，新司机的坐标，时间

FireBase <-> 后台，纯粹的数据读写交互

FireBase –> 司机 以及 Firebase–> 餐馆，单向的数据传递，换句话说，一切计算和逻辑处理都在后台部分，Firebase只存储处理好的数据

其实任何状态的更新都会导致所有的司机任务的更新甚至取消！

在Firebase中，一直存储着每个司机的任务列表，类似于以下格式

Current Task (pick up / delivery)；Next Target address; deadline time

Coming Task 1 (pick up / delivery); Target address; expired time

Coming Task 2 (pick up / delivery); Target address; expired time

注意，每个司机都会有一个类似上面的任务列表

对于运算部分，最常见的场景应该是 新来一个任务 （包含取餐地址，送达地址），怎样获得分配最优解，一般来说，我们可以通过 Google Maps获得多个地点的Best Route以及到每个节点的时间，那就需要考虑下面很多问题：

Google Mpas的API的调用频率有限制；

当然司机如果正在开始配送（已经取到餐）则不应打乱当前未配送的任务，最早只能插入到下一个还没有开始配送的任务（可以是已经取到的）；

某司机已开始的任务不能再调整之后超过一个最长等待时间；每一个任务都包含取和送两部分，这两部分可以分别整合到已有的司机那里；

如果某个任务delay的时间太久，需要考虑取消他未开始的任务（未取）；

某些场景配送失败，但是餐还在这个司机身上，则重新分配的时候只能选择这个司机（除非是有事故）；

很多条件的判断标准可能是一些我们设定好的参数，并且参数应该便于调整；需要考虑当前司机是否在工作时间，

* 以上的信息仅仅是关于分配计算的，不包含所有细节，稍后会介绍所有接口细节