大作业–Leo

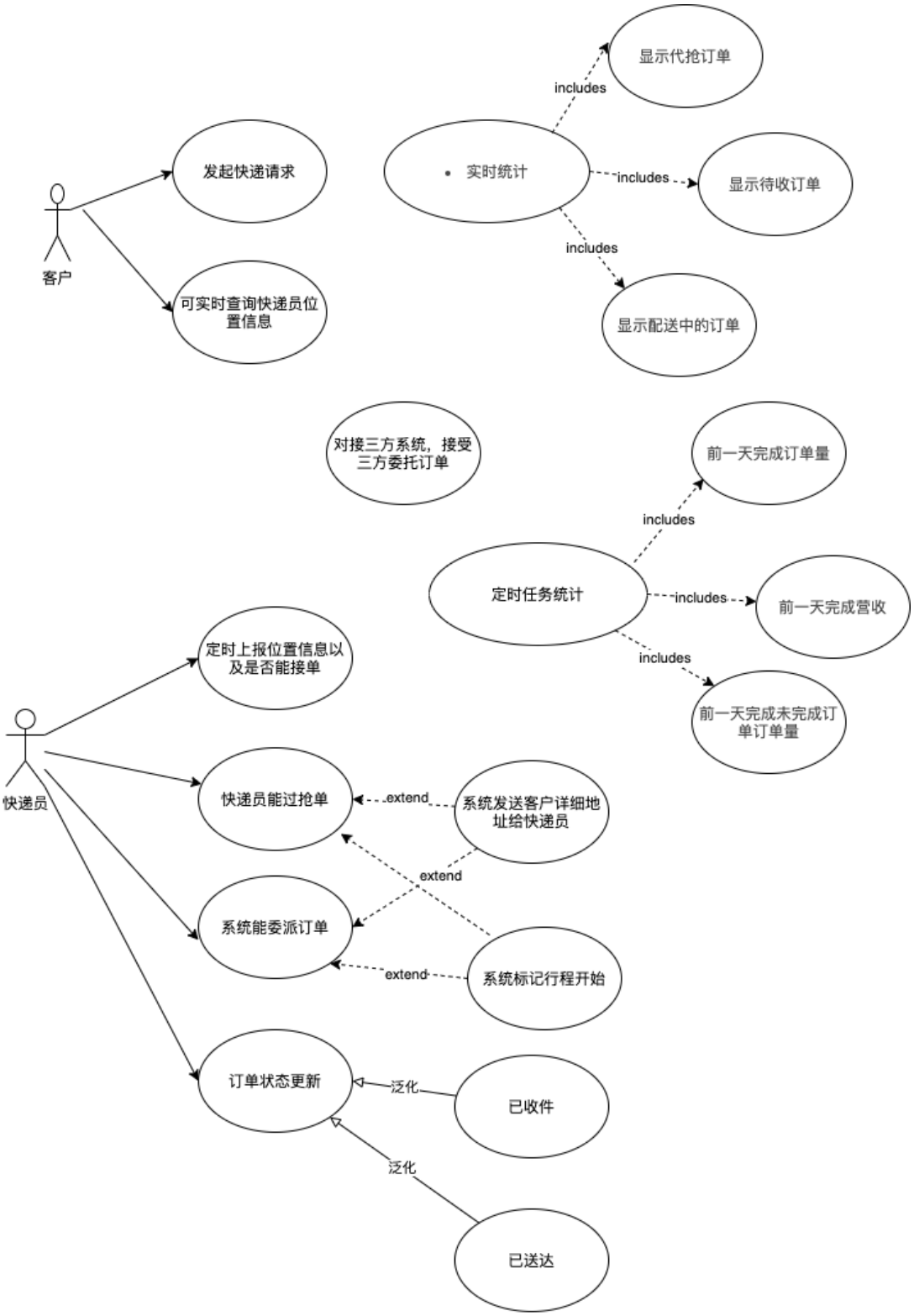
1. 需求分析与系统目标

我们的系统中有两种类型的用户：1）快递员， 2) 客户

- 快递员需要定期向服务器上报其位置和其当前是否能够接单；
- 客户发起快递请求，附近的快递员能接到通知并进行抢单；
- 当附近没有快递员进行抢单时，系统能够直接派单给附近的某个空闲的快递员；
- 当有快递员接单后，快递员标记行程开始，系统发送客户的详细地址到快递员手机上；
- 当快递员收取快递后，系统订单状态变为：已收件；
- 在订单完成之前，用户能够看到快递员当前位置；
- 当快递员将快递送到目的地后，快递员标记行程完成；
- 当快递员将快递送到目的地后，系统订单状态变成已送达；
- 系统能够按区域实时统计显示代抢订单，待收订单，配送中的订单；
- 每天上午8点前按区域统计显示:前一天完成订单量、营收、未完成订单订单量
- 系统能够获取饿了么外卖订单并派发给附近快递员

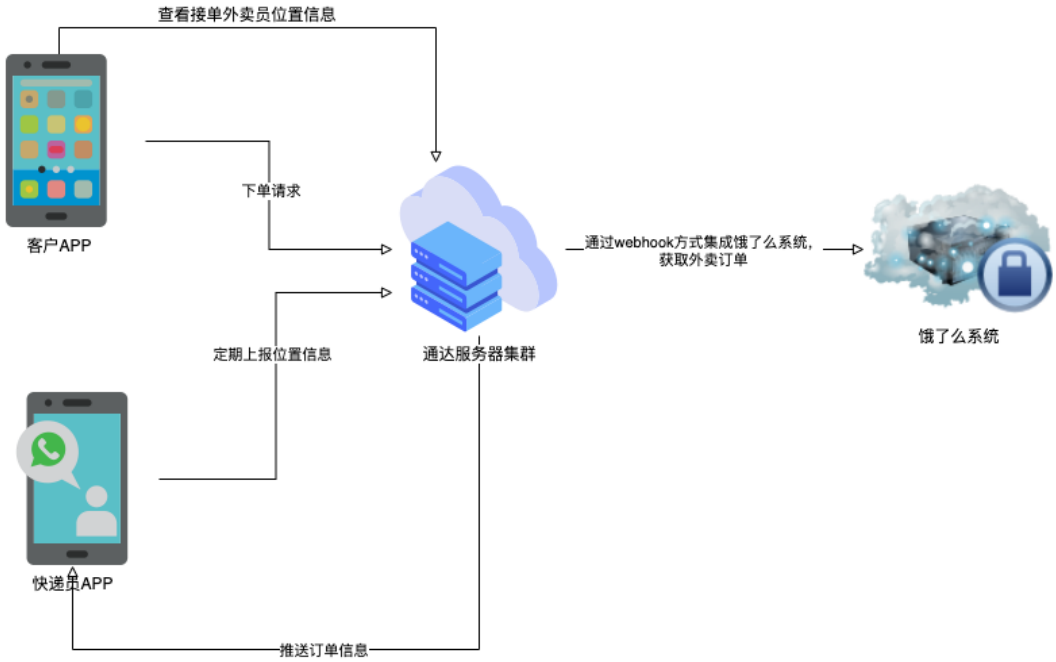
1.1 系统用例图

根据上述需求，系统用例图如下：



1.2 系统整体部署模型

根据上述讨论的系统需求和功能，下图是系统整体模型，描述了用户APP，快递员APP，三方系统（比如饿了么）之间的交互关系，



2. 系统容量估算与约束

预计三个月后日订单超过1万，一年日单超过50万，考虑到满足业务快速增长的需求，我们系统设计目标定为日单100万，并假设日活50万用户，2万快递员，并且假设所有的快递员每3秒上报他们的当前位置，一旦有客户发起下单请求，系统能够实时通知到快递员。

3. 系统基本设计与实现

我们先看下如何存储快递员当前位置并且找到附近的快递员，一个简单的方案是使用数据库存储快递员位置信息，我们可以使用RDBMS数据库，比如使用mysql之类的RDBMS数据库存储所有快递员的数据，使得每个快递员的位置信息存储一行，每行中有经度和纬度两个列的信息，并在经度和纬度上加索引来加快查询速度。

为了找到用户位置（X，Y）附近所有距离小于D的快递员，我们可以这样做：

```
Select * from Places where Latitude between X-D and X+D and Longitude
between Y-D and Y+D
```

*注意这里使用了简化算法，所以上述查询并不是精确的，精确的两点之间距离计算应该采用距离公式。

下面我们看下这个查询的效率如何，前面我们假设有2万个快递员，所以数据库需要两万条记录，因为两个位置上都建立了索引，所以查询的效率应该是可以的。

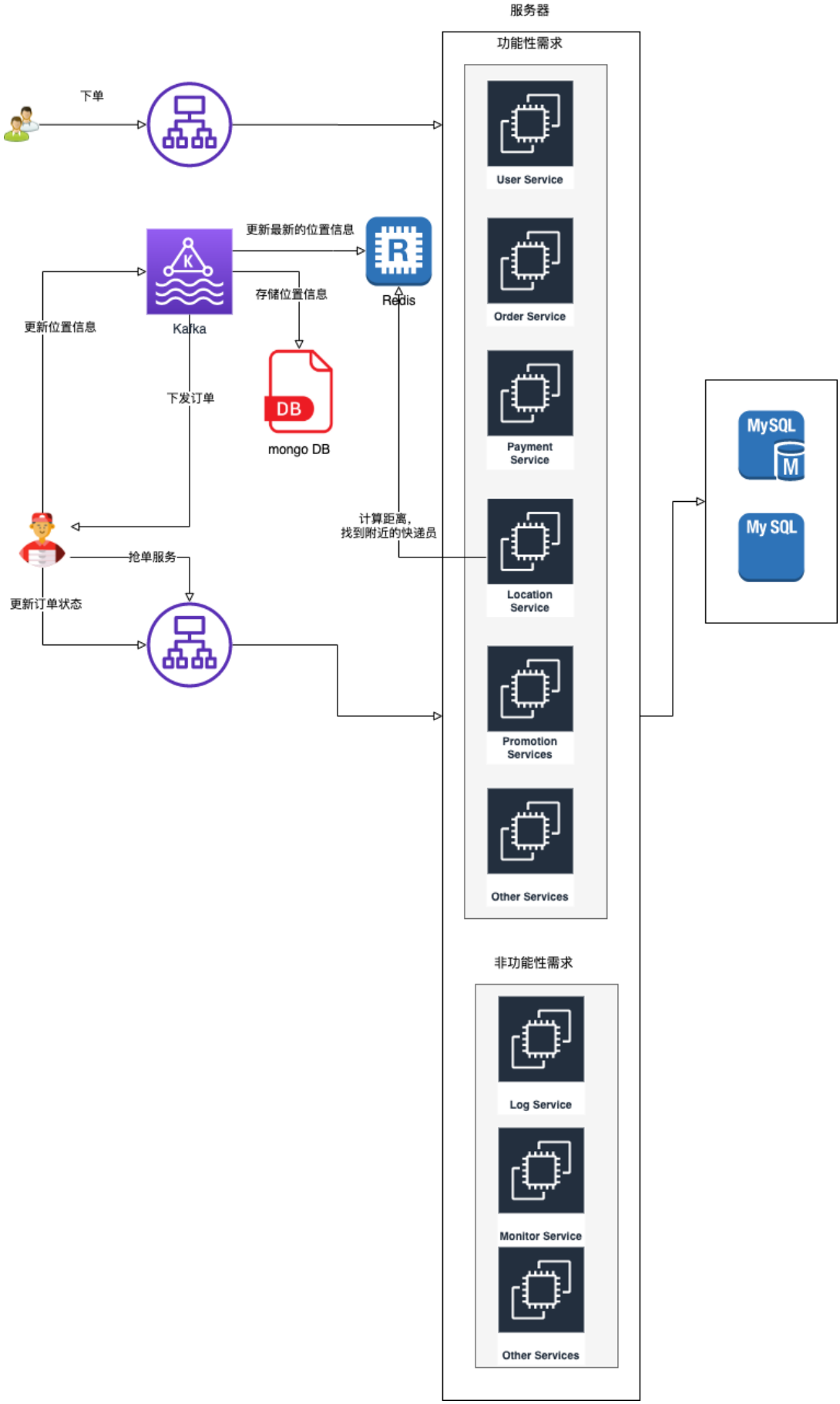
在让我们看下更新效率，因为有两万个快递员每30s就要上报一次当前位置，之后数据库要更新位置信息，TPS约为667, 根据来自这个文档中的信息(https://mysqlserverteam.com/mysql-connection-handling-and-scaling/)，MySQL数据库是能够满足这个TPS要求的，所以我们可以在我们的方案中使用mysql或其他的RDBMS来存储快递员的最新的位置更新信息。不过为了进一步提高性能，可以考虑使用Redis来保存快递员实时位置。

再看下如何保存快递员的运动轨迹，因为快递员没3秒钟上报一次位置更新，为了保存轨迹，我们需要记录每一次快递员的位置更新，假设一个快递员一天工作8小时，那么一天中快递员的数据量是：

> 8小时*3600s/30s*20000快递员 = 19200000

可以看到2万个快递员1天会产生近2千万条记录，RDBMS基本不能满足需求，而且由于系统对于快递员的位置信息并不需要非常准确，即使丢失信息也可以接受，所以我们可以使用Nosql(比如MongoDB)来保存用户的历史位置信息数据。这样系统可以可以从Nosql数据库中获取某个时间段的位置信息来实时展示快递员的位置轨迹。

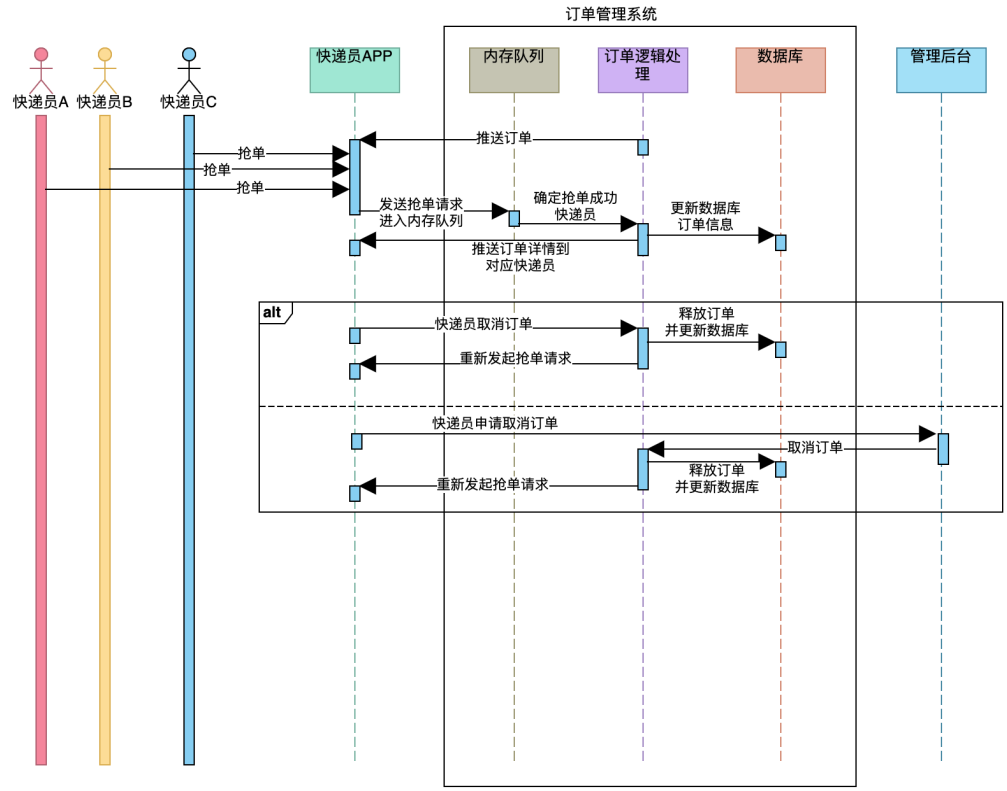
那么如何把快递员的信息发送到系统上呢？对于这么大的数据流我们可以使用Kafka消息流来推送位置信息到服务器，并将位置信息保存在Nosql中。
再来考虑下如何推送订单信息给快递员，简单的讲，有两种方式，一种是polling，即快递员app不停的去服务器端拉取最新订单信息，这种方式对服务器压力较大，每次拉取都要进行一次查询，而且polling的时间间隔也不好决定。另一种方式是服务器推送消息到快递员APP，这种方式节省了服务器的计算资源，考虑到推送使用的具体技术，可以采用wesocket或消息队列的方式，因为前面使用了Kafka来推送位置信息，这里我们也同样可以使用Kafka消息队列来推送订单信息给APP。
综上所述，系统的基本部署模型如下图所示：



4. 抢单系统设计

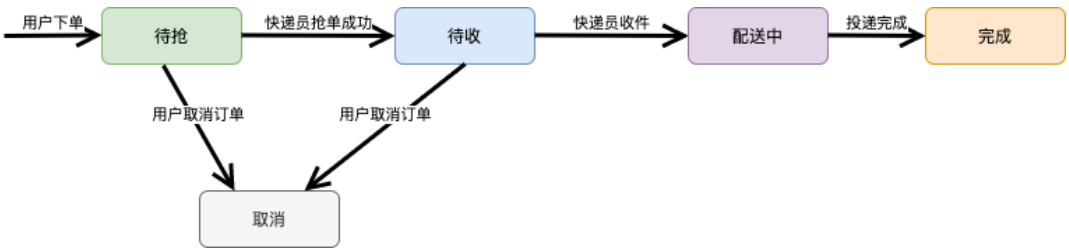
下面我们来看下如何设计抢单系统，该抢单的场景和互联网秒杀促销的场景并不相同，并没有那么大的并发量，因为每个区域的快递员数量有限，一个大点区域几十个快递员的规模就已经很大了，假设该区域有50个快递员，那么也就是有50个并发请求去抢同一个订单，所以在这个抢单系统设计中我们并不需要使用redis，MQ等第三方组件来提升并发性，不过我们可以通过自己实现内存队列的方式来对抢单请求进行排序，然后可以按照先来先得的方式将订单派发

给第一个进到内存队列中的快递员，将数据库中订单的信息和该快递员关联，同时内存队列中的其他快递员直接返回，显示抢单失败。
另外我们也要考虑当抢单成功后的快递员由于某些原因需要取消订单的场景，针对该场景，我们提供两种方式释放订单，一是该快递员自己手动释放订单，二是该快递员上报给运营组织，然后运营组织通过管理后台释放订单，释放订单后系统重新发起抢单流程。
系统的时序模型如下图示



5. 订单状态流转

订单的状态有五种，分别是：待抢，待收，配送中，完成，取消。分别通过用户下单，抢单成功，收件，投递完成，用户取消的事件来触发，具体的订单状态流转如下图所示，



6. 高可用设计

为了保障系统的高可用，避免出现单点故障，我们使用负载均衡技术和部署冗余服务器来进行故障转移，对于数据库系统，采用主备分离的方式，主库负责写入，从库负责读取数据。

7. 系统未来演进

- 系统中调度系统中使用的算法还比较粗超，可以使用更先进的算法来提升性能，比如基于地理分片的方式来划分不同的区域，比如借鉴Uber的实时调度平台的算法：https://www.infoq.com/news/2015/03/uber-realtime-market-platform/， 也可以基于未来的北斗卫星定位提供更精确的地理位置信息。
- 系统可以提前分组不同区域的快递员，比如从大到小划分不同的区域，并将快递员指定分配到不同区域，例如有上海大区和北京大区，然后上海大区细分为静安中区，徐汇中区等，静安中区可以划分为更小的区域，将快递员分配到这些更小的区域中，这样在查询某个区域快递员时可以更快的搜索数据库。

- 增加排名系统，将快递员进行排名，排名规则可以按照用户评分，成单量等综合考虑，对于排名高的有抢单权重增加，更容易抢单等。
- 增加权限系统，给不同部门的人分配不同的权限，以便进行更好的管理。