乘客智能打车

1. 智能打车流程

2.1 车主登记注册

> 顺风车主要接单之前需要先进行司机注册以及车主认证，车主必须实名认证以及车辆注册认证后才能进行发布行程

2.2 司机发布行程

> 当司机顺利注册后就可以发布行程了，司机先发送一个行程，等下系统分配给顺路乘客的行程

2.3 乘客发布行程

> 当乘客想要打车的时候需要发布行程，发布完成行程后就需要等待司机的邀约了

2.4 平台派单

> 由平台进行数据运算，并将乘客需求发布并推送给顺路车主，

2.5 邀约乘客

> 司机可以看到乘客的行程列表以及行程的匹配度，司机可以和乘客聊天然后邀约乘客

2.5 乘客同意邀约

> 当顺路乘客同意邀约后车主就和乘客缔结了乘车协议，会创建订单，等待乘客上车

2.6 乘客上下车

> 同意邀约后，司机就需要去接该乘客，然后乘客上车，等到送达乘客完成后就乘客下车

2.7 确认送达

> 车主送达该乘客后就需要确认送达，修改订单状态，并且让用户进行支付

2. 智能打车难点

匹配相关用户行程

行程匹配度排行

车主发起邀请，乘客接受 / 拒绝邀请

剩余座位判断

邀请超时取消

路径计算

计费 - 装饰者模式

3. 涉及到的技术点

3.1 Redis相关操作

3.1.1 zset操作回顾

> Redis 有序集合和集合一样也是 string 类型元素的集合,且不允许重复的成员。

不同的是每个元素都会关联一个 double 类型的分数。redis 正是通过分数来为集合中的成员进行从小到大的排序,有序集合的成员是唯一的,但分数(score)却可以重复。

3.1.1.1 zadd

> 向有序集合添加一个或多个成员，或者更新已存在成员的分数

格式是：zadd zset的key score值 项的值， Score和项可以是多对，score可以是整数，也可以是浮点数，还可以是+inf表示无穷大，-inf表示负无穷大

3.1.1.2 zrange

> 通过索引区间返回有序集合指定区间内的成员

格式是：zrange zset的key 起始索引 终止索引（ withscore ）

3.1.1.3 zrangebyscore

> 通过分数返回有序集合指定区间内的成员

格式是：zrangebyscore zset的key 起始score 终止score (withscore),默认是包含端点值的，如果加上"("表示不包含，后面还可以加上limit来限制。

3.1.1.4 zrevrangebyscore

> 返回有序集中指定分数区间内的成员，分数从高到低排序

格式是：zrevrangebyscore zset的key 终止score 起始score(withscores)

3.1.1.5 zrem

> 移除有序集合中的一个或多个成员

格式是：zrem zset的key 项的值，项的值可以是多个

3.1.2 hash操作回顾

> Redis hash 是一个 string 类型的 field（字段）和 value（值）的映射表，hash 特别适合用于存储对象。

Redis 中每个 hash 可以存储 2的32次方 - 1 键值对（40多亿），但是不推荐。

3.1.2.1 hset

> 新增或更新一个配置项

格式是：hset hash的key 项的key 项的值

3.1.2.2 hmset

> 同时将多个 field-value (域-值)对设置到哈希表 key 中。

格式是：hmset hash的key 项的key 项的值。（项的key和项的值可以多对）

3.1.2.3 hget

> 获取存储在哈希表中指定字段的值。

格式是：hget hash的key 项的key

3.1.2.4 hgetall

> 获取在哈希表中指定 key 的所有字段和值

格式是：hgetall hash的key

4.业务分析

4.1 前期条件

> 车主注册主要涉及到用户注册，然后进行车主认证以及车辆认证

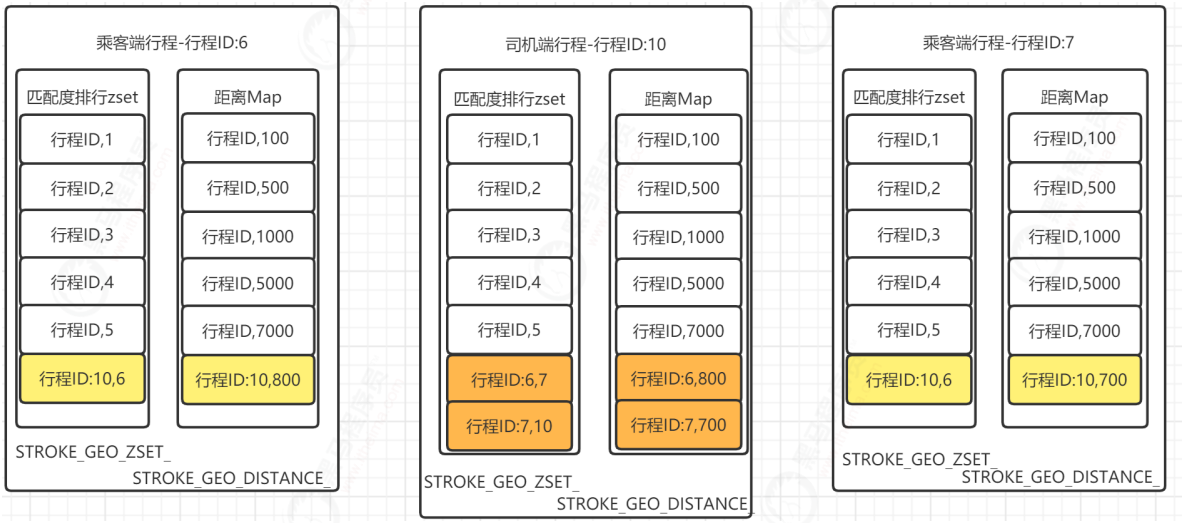
4.2 司机乘客发布行程

> 回顾行程缓存体系（重要）

4.3 查看顺路行程

> 当司机或者乘客发布行程后，司机或者乘客能够在手机端能够看到顺路的乘客

这里用到了我们上一节发布行程中，将行程发布到到redis后，并会通过RedisGEO计算出来相应的匹配度，并添加到，对应的redis的zset的结构中，乘客和司机互相保存对应行程列表



4.3.1 前端页面

ajax提交行程坐标

4.3.2 顺路行程分解

> 当手机端查看行程列表的时候会先从我们的redis的zset结构中获取到对应的行程id，按照分值的匹配度从高到地进行排序，并在手机端显示出来

4.3.2.1 获取列表数据

> 这里我们需要先根据乘客或者司机获取到当前的行程id再来从zset中获取对应的行程列表

```
/**
  * 根据行程ID查看行程列表
  *
  * @param strokeVO
  * @return
  */
public ResponseVO<StrokeVO> itineraryList(StrokeVO strokeVO) {
```

```
    StrokePO strokePO = CommonsUtils.toPO(strokeVO);
    //获取当前行程信息
    List<StrokePO> resultList = strokeAPIService.selectlist(strokePO);
    StrokePO result = LocalCollectionUtils.getOne(resultList);
    if (null == result) {
        throw new BusinessRuntimeException(BusinessErrors.DATA_NOT_EXIST);
    }
    //从zset中获取行程列表
    List<StrokeVO> strokeVOS = getZsetResulets(result);
    return ResponseVO.success(strokeVOS);
}
```

4.3.2.2 从zset获取行程

> 这里我们是要从zset中排序后的行程列表

```
/**
  * 获取排序后的结果
  *
  * @param strokePO
  * @return
  */
public List<StrokeVO> getZsetResulets(StrokePO strokePO) {
    //从zset中获取行程数据
    List<ZsetResultBO> zsetResultBOList =
redisHelper.getZsetSortVaues(HtichConstants.STROKE_GEO_ZSET_PREFIX,
strokePO.getId());
    List<StrokeVO> strokeVOS = new ArrayList<>();
    for (ZsetResultBO zsetResultBO : zsetResultBOList) {
        //获取乘客详细
        StrokePO tmp = strokeAPIService.selectByID(zsetResultBO.getValue());
        if (null == tmp) {
            continue;
        }
        //跳过同一个用户是司机以及乘客的情况
        if (tmp.getPublisherId().equals(strokePO.getPublisherId())) {
            continue;
        }
        StrokeVO strokeVO = (StrokeVO) CommonsUtils.toVO(tmp);
        strokeVO.setInviterTripId(strokePO.getId());
        strokeVO.setInviteeTripId(tmp.getId());
        //乘客查看司机的空余座位数
        if (tmp.getRole() == 1) {
            strokeVO.setQuantity(getSurplusSeats(tmp.getId()));
        }
        renderStrokeVO(strokeVO);
        //设置匹配度
        strokeVO.setSuitability(zsetResultBO.getScore().toString());
        strokeVOS.add(strokeVO);
    }
    return strokeVOS;
}
```

4.3.2.3 行程数据渲染

> 因为我们的行程列表数据显示了匹配度起点终点距离等信息，我们需要将我们匹配的行程数据补全

```
/**
  * 行程数据渲染
  *
  * @param strokeVO
  */
private StrokeVO renderStrokeVO(StrokeVO strokeVO) {
    //渲染距离数据
    String distanceStr =
redisHelper.getHash(HtichConstants.STROKE_GEO_DISTANCE_PREFIX,
strokeVO.getInviterTripId(), strokeVO.getInviteeTripId());
    //设置起点距离
    if (StringUtils.isNotEmpty(distanceStr) && distanceStr.contains(":")) {
        String[] distances = distanceStr.split(":");
        strokeVO.setStartDistance(Float.parseFloat(distances[0]));
        strokeVO.setEndDistance(Float.parseFloat(distances[1]));
    }

    //获取用户对象
    AccountPO accountPO =
accountAPIService.getAccountByID(strokeVO.getPublisherId());
    if (null != accountPO) {
        strokeVO.setUseralias(accountPO.getUseralias());
        strokeVO.setAvatar(accountPO.getAvatar());
    }
    return strokeVO;
}
```

4.3.2.4 司机空座数

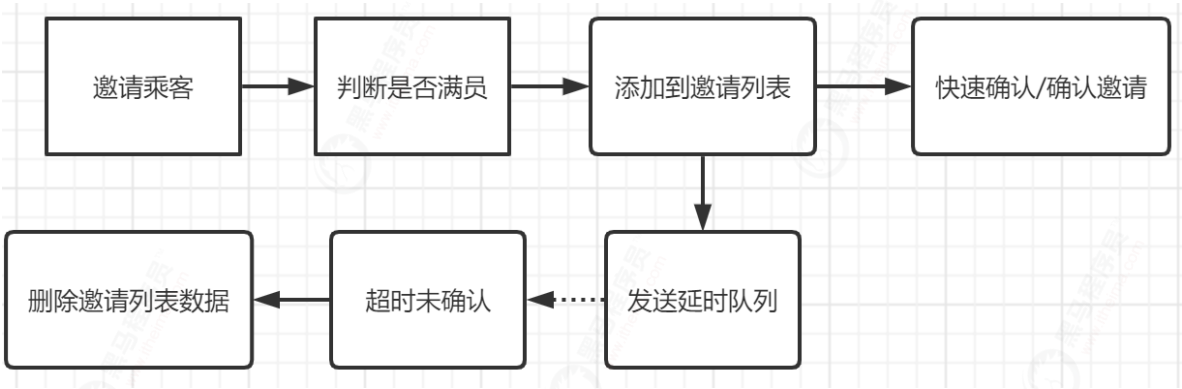
> 一个顺风车司机的座位数是固定的，当邀约到一个乘客后，空座数要减一，我们的这个结构是通过一个redis的hash来保存的，具体结构我们在下面的邀约乘客来说说明

```
/**
  * 获取剩余座位数
  *
  * @return
  */
private int getSurplusSeats(String inviterTripId) {
    StrokePO strokePO = strokeAPIService.selectByID(inviterTripId);
    if (null == strokePO) {
        throw new BusinessRuntimeException(BusinessErrors.DATA_NOT_EXIST);
    }
    Map<String, String> driverMap =
redisHelper.getHashByMap(HtichConstants.STROKE_INVITE_PREFIX, inviterTripId);
    //获取已确认的成员数量
    int member = 0;
    for (Map.Entry<String, String> entry : driverMap.entrySet()) {
        //已确认行程的数据进行统计
        if
(entry.getValue().equals(String.valueOf(InviteState.CONFIRMED.getCode()))) {
            member++;
        }
    }
    return strokePO.getQuantity() - member;
}
```

4.4 邀请乘客

> 当司机看到乘客的匹配列表后，司机就可以对乘客发起邀请，邀请后乘客需要进行同意或者拒绝邀请

4.4.1 具体流程



4.4.2 前端页面

前端页面！！！

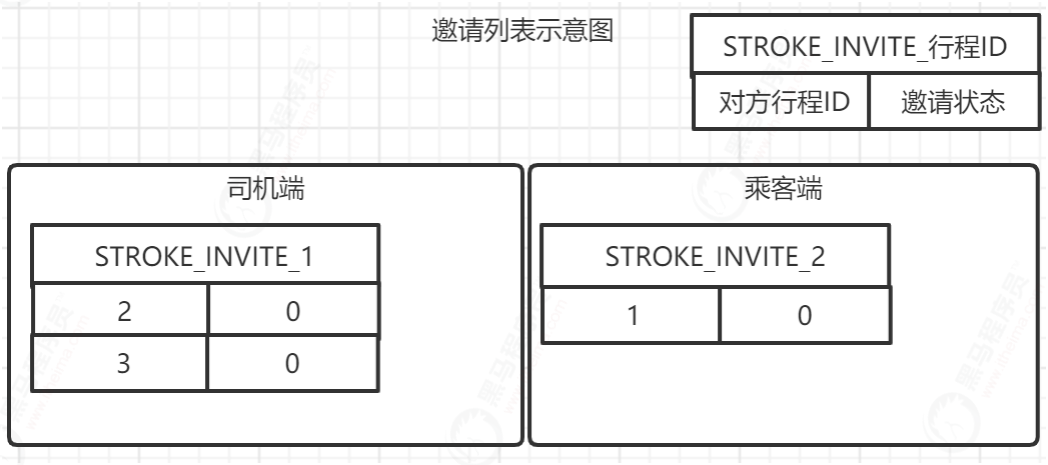
4.4.3 邀请乘客分解

创建一个司机行程，两个乘客行程。

让司机邀请这两个乘客

4.4.3.1 邀请列表数据结构

> 邀请列表是用互相保存对方的行程id以及邀请状态,我们用redis的hash结构来保存邀请状态



4.4.3.2 邀请状态说明

- 0:未确认
- 1:已确认
- 2:已拒绝
- 3:已超时

4.4.3.3 邀请乘客

> 司机邀请这两个乘客行程
>
> 邀请乘客后我们的主要功能，主要操作是在我们的redis中在乘客端和司机端的hash结构中互相保存对方的id以及邀约状态

```
/**
  * 顺风车邀请
  *
  * @param strokeVO
  * @return
  */
public ResponseVO<StrokeVO> invite(StrokeVO strokeVO) {
    isFullStarffed(strokeVO);
    //获取司机行程ID
    String inviterTripId = strokeVO.getInviterTripId();
    //获取乘客行程ID
    String inviteeTripId = strokeVO.getInviteeTripId();

    //创建邀请状态
    // 0 = 未确认， 1 = 已确认 ， 2= 已拒绝
    redisHelper.addHash(HtichConstants.STROKE_INVITE_PREFIX, inviteeTripId,
inviterTripId, String.valueOf(InviteState.UNCONFIRMED.getCode()));
    redisHelper.addHash(HtichConstants.STROKE_INVITE_PREFIX, inviterTripId,
inviteeTripId, String.valueOf(InviteState.UNCONFIRMED.getCode()));
    //发送延时消息
    mqProducer.sendOver(JSON.toJSONString(strokeVO));
    quickConfirm(strokeVO);
    return ResponseVO.success(null);
}
```

4.4.3.4 判断是否满员

> 邀请乘客的时候需要判断是否已经满员，我们是通过上面的hash来判断的，司机的车辆有座位数的限制，邀请一个乘客座位数-1，我们只需要判断邀请`hash`的受邀的数量然后用司机的座位数减去就可以

```
/**
  * 检查是否已满员
  *
  * @param strokeVO
  */
private void isFullStarffed(StrokeVO strokeVO) {
    //获取司机行程ID
    int surplusSeats = getSurplusSeats(strokeVO.getInviterTripId());
    //座位数大于等于已确认人数 抛出异常
    if (surplusSeats <= 0) {
        throw new BusinessRuntimeException(BusinessErrors.STOCK_FULL_STARFFED);
    }
}
```

```
/**
  * 获取剩余座位数
  *
  * @return
  */
private int getSurplusSeats(String inviterTripId) {
    StrokePO strokePO = strokeAPIService.selectByID(inviterTripId);
    if (null == strokePO) {
        throw new BusinessRuntimeException(BusinessErrors.DATA_NOT_EXIST);
    }
    //获取邀请列表
```

```
    Map<String, String> driverMap =
redisHelper.getHashByMap(HtichConstants.STROKE_INVITE_PREFIX, inviterTripId);
    //获取已确认的成员数量
    int member = 0;
    for (Map.Entry<String, String> entry : driverMap.entrySet()) {
        //已确认行程的数据进行统计
        if
(entry.getValue().equals(String.valueOf(InviteState.CONFIRMED.getCode()))) {
            member++;
        }
    }
    return strokePO.getQuantity() - member;
}
```

4.4.4 邀请超时取消

> 我们在这里使用了延时队列来实现超时自动取消邀请，我们下面来回顾下RabbitMQ的延时队列

4.4.4.1 延时队列回顾

延时队列顾名思义，即放置在该队列里面的消息是不需要立即消费的，而是等待一段时间之后取出消费。

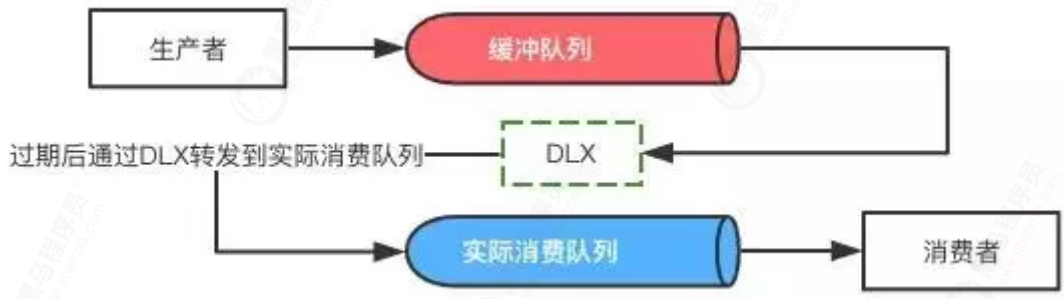
4.4.4.2 适用场景

在订单系统中，一个用户下单之后通常有30分钟的时间进行支付，如果30分钟之内没有支付成功，那么这个订单将进行一场处理。这是就可以使用延时队列将订单信息发送到延时队列。

用户希望通过手机远程遥控家里的智能设备在指定的时间进行工作。这时候就可以将用户指令发送到延时队列，当指令设定的时间到了再将指令推送到智能设备。

4.4.4.3 利用死信队列来实现

> AMQP协议和RabbitMQ队列本身没有直接支持延迟队列功能，但是可以通过以下特性模拟出延迟队列的功能。



RabbitMQ的Queue可以配置x-dead-letter-exchange和x-dead-letter-routing-key（可选）两个参数，如果队列内出现了dead letter，则按照这两个参数重新路由转发到指定的队列。

- x-dead-letter-exchange：出现dead letter之后将dead letter重新发送到指定exchange
- x-dead-letter-routing-key：出现dead letter之后将dead letter重新按照指定的routing-key发送

4.4.4.4 Rabbitmq配置

```
stroke项目下的 RabbitConfig
```

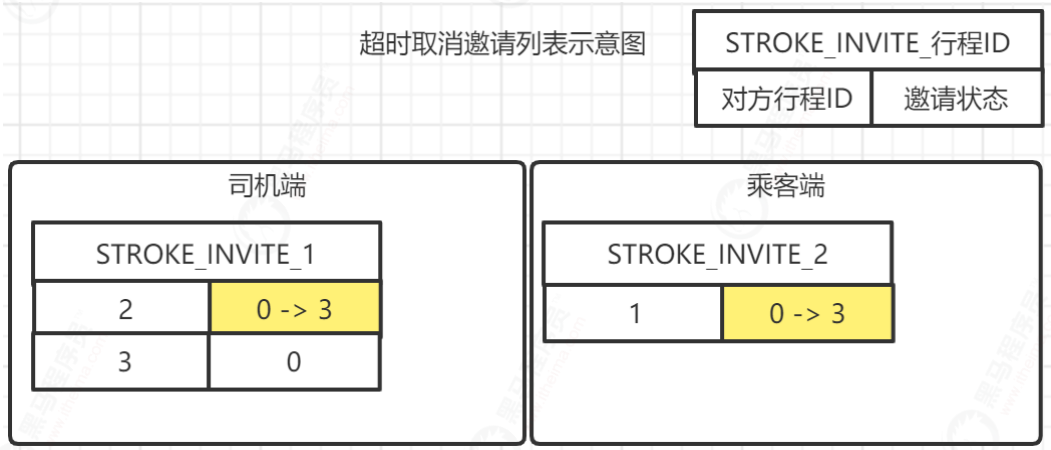
4.4.4.5 发送延时队列

> 在发起邀请后为了防止没有乘客没有接受一直存在,我们使用延时队列在一定时间内没有接受邀请就自动取消邀请

```
/**
  * 发送延时订单MQ
  *
  * @param mqMessage
  */
public void sendOver(String mqMessage) {
    //发送消息
    rabbitTemplate.convertAndSend(RabbitConfig.STROKE_OVER_QUEUE_EXCHANGE,
RabbitConfig.STROKE_OVER_KEY, mqMessage);
}
```

4.4.4.6 超时取消邀请

> 如果在规定时间内没有接受邀请,我们就需要将邀请状态改为超时状态,让前端进行展示




```
/**
  * 行程超时监听
  *
  * @param massage
  * @param channel
  * @param tag
  */
@RabbitListener(
    bindings =
    {
        @QueueBinding(value = @Queue(value = RabbitConfig.STROKE_DEAD_QUEUE,
durable = "true"),
                      exchange = @Exchange(value =
RabbitConfig.STROKE_DEAD_QUEUE_EXCHANGE), key = RabbitConfig.STROKE_DEAD_KEY)
    })
@RabbitHandler
public void processStroke(Message massage, Channel channel,
@Header(AmqpHeaders.DELIVERY_TAG) long tag) {
    StrokeVO strokeVO = JSON.parseObject(massage.getBody(), StrokeVO.class);
    if (null == strokeVO) {
        return;
    }
    try {
        strokeHandler.timeoutHandel(strokeVO);
        //手动确认机制
        channel.basicAck(tag, false);
```

```
    } catch (Exception e) {
        e.printStackTrace();
    }
}
```

```
/**
  * 处理邀请超时
  *
  * @param strokeVO
  */
public void timeoutHandel(StrokeVO strokeVO) {
    //获取司机行程ID
    String inviterTripId = strokeVO.getInviterTripId();
    //获取乘客行程ID
    String inviteeTripId = strokeVO.getInviteeTripId();
    String tripeeStatus =
redisHelper.getHash(HtichConstants.STROKE_INVITE_PREFIX, inviteeTripId,
inviterTripId);
    String triperStatus =
redisHelper.getHash(HtichConstants.STROKE_INVITE_PREFIX, inviterTripId,
inviteeTripId);
    //如果是未邀请状态
    if (tripeeStatus.equals(String.valueOf(InviteState.UNCONFIRMED.getCode()))
&& triperStatus.equals(String.valueOf(InviteState.UNCONFIRMED.getCode()))) {
        //设置为超时状态
        redisHelper.addHash(HtichConstants.STROKE_INVITE_PREFIX, inviteeTripId,
inviterTripId, String.valueOf(InviteState.TIMEOUT.getCode()));
        redisHelper.addHash(HtichConstants.STROKE_INVITE_PREFIX, inviterTripId,
inviteeTripId, String.valueOf(InviteState.TIMEOUT.getCode()));
    }
}
```