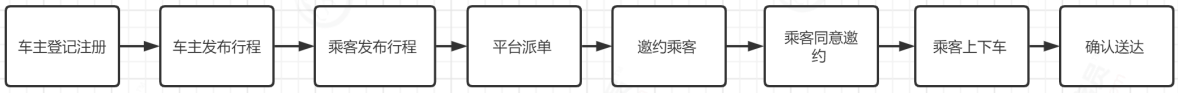
乘客智能打车

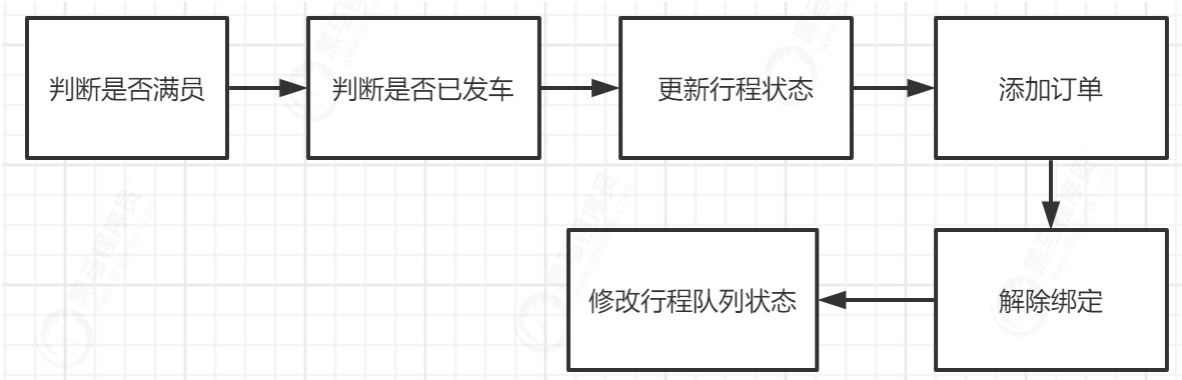


#

4.5 乘客确认/拒绝邀请

> 当司机发起邀约后乘客在手机端可以查看邀约通知,可以和司机进行沟通,然后同意或者拒绝邀约

4.5.1 具体流程



4.5.2 前端页面

4.5.3 确定邀约

4.5.3.1 判断是否发车

> 我们在确定邀约的时候判断车辆是否已发车,我们后期发车的时候会删除邀约列表,所以我们只要判断邀约列表的key是否存在即可,存在则说明邀约的车辆未发车,不存在则已发车

```
//判断司机是否已经发车
boolean isDepart = redisHelper.exists(HtichConstants.STROKE_INVITE_PREFIX,
inviterTripId);
if (!isDepart) {
    throw new BusinessRuntimeException(BusinessErrors.STOCK_ALREADY_DEPART);
}
```

4.5.3.2 更新行程状态

> 乘客确定邀约后,需要更新乘客的行程状态,将乘客的行程状态变为已接受邀请

```
/**
  * 处理行程状态变更
  *
  * @param tripid
  * @param orginStatus
  * @param targetStatus
  */

private StrokePO travelStatusChange(String tripid, int orginStatus, int
targetStatus) {
    StrokePO strokePO = new StrokePO();
    strokePO.setId(tripid);
    strokePO.setStatus(orginStatus);
```

```
    //如果查询行程状态错误则抛出异常
    List<StrokePO> strokePOList = strokeAPIService.selectlist(strokePO);
    if (strokePOList.isEmpty()) {
        throw new BusinessRuntimeException(BusinessErrors.DATA_STATUS_ERROR, "行
程状态错误");
    }
    strokePO.setStatus(targetStatus);
    strokeAPIService.update(strokePO);
    return strokePO;
}
```

4.5.3.3 添加订单

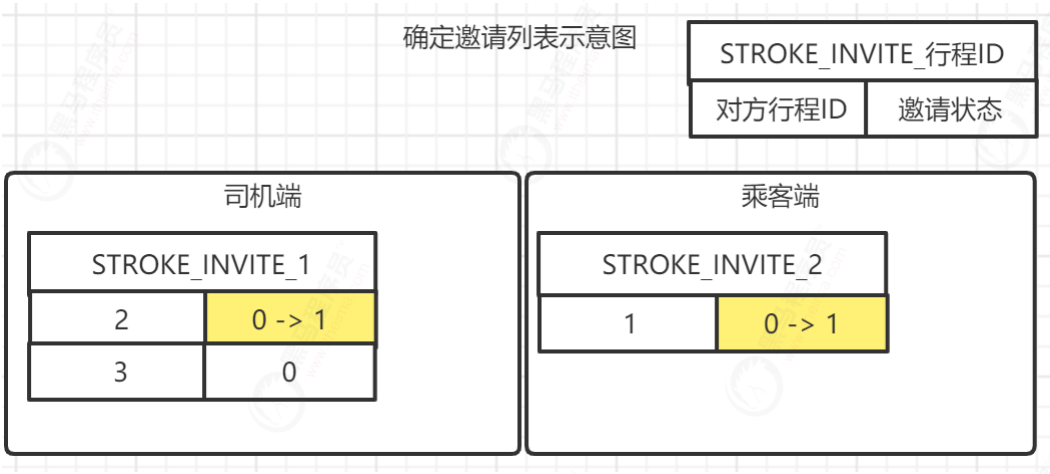
> 用户确定邀约后需要新增用户订单,用户订单涉及到计算距离金额等,这里我们使用百度地图进行金额计算,后面我们会进行介绍

```
/**
  * 添加订单
  *
  * @param inviter 司机行程对象
  * @param invitee 乘客行程对象
  */
private void addOrder(StrokePO inviter, StrokePO invitee) {
    OrderPO orderPO = new OrderPO();
    orderPO.setId(CommonsUtils.getWorkerID());
    orderPO.setDriverStrokeId(inviter.getId());
    orderPO.setDriverId(inviter.getPublisherId());
    orderPO.setPassengerStrokeId(invitee.getId());
    orderPO.setPassengerId(invitee.getPublisherId());
    orderPO.setCreatedBy(invitee.getCreatedBy());
    orderPO.setCreatedTime(new Date());
    orderPO.setUpdatedBy(invitee.getCreatedBy());
    orderPO.setUpdatedTime(new Date());
    orderPO.setStatus(0);

    RoutePlanResultBO resultBO = getRoutePlanResult(invitee);
    if (null != resultBO) {
        orderPO.setDistance(resultBO.getDistance().getValue());
        orderPO.setEstimatedTime(resultBO.getDuration().getValue());
        orderPO.setCost(CommonsUtils.valuationPrice(orderPO.getDistance()));
    }
    orderAPIService.add(orderPO);
}
```

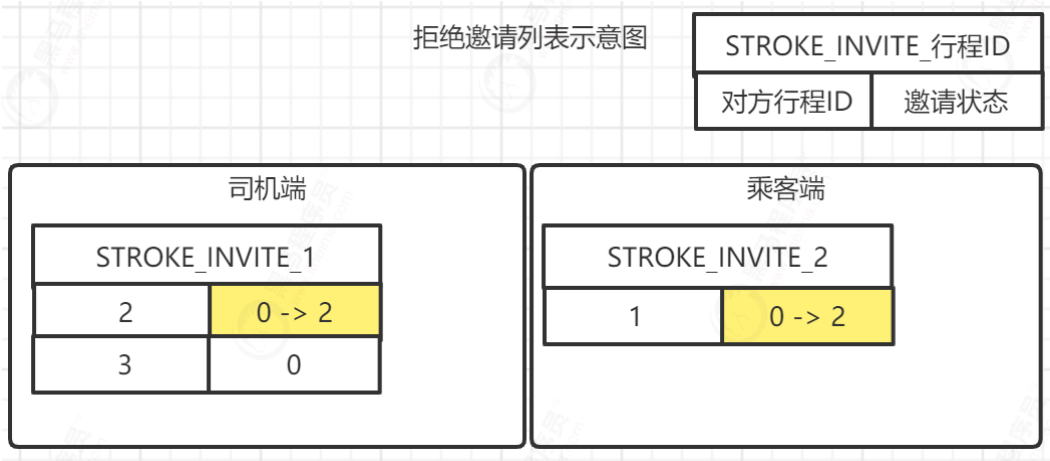
4.5.3.4 更新邀请状态

> 最后一步就是更新邀请列表,如果乘客将乘客的确认邀请状态或者拒绝邀请更新到邀请列表中

> 确定邀请的状态变化



拒绝邀请的状态变化



总体代码展示

```
/**
  * 接受以及拒绝邀请
  *
  * @param strokeVO
  * @return
  */
public ResponseVO<StrokeVO> inviteAccept(StrokeVO strokeVO) {
    isFullStarffed(strokeVO);
    //获取司机行程ID
    String inviterTripId = strokeVO.getInviterTripId();
    //判断司机是否已经发车
    boolean isDepart = redisHelper.exists(HtichConstants.STROKE_INVITE_PREFIX,
inviterTripId);
    if (!isDepart) {
        throw new BusinessRuntimeException(BusinessErrors.STOCK_ALREADY_DEPART);
    }
    //获取乘客行程ID
    String inviteeTripId = strokeVO.getInviteeTripId();
    //司机行程对象
    StrokePO inviter = strokeAPIService.selectByID(inviterTripId);
    //乘客行程对象
    StrokePO invitee = strokeAPIService.selectByID(inviteeTripId);
    //创建邀请状态
    int status = strokeVO.getStatus();
    InviteState state = InviteState.getState(status);
    if (null == state) {
        throw new BusinessRuntimeException(BusinessErrors.DATA_STATUS_ERROR);
```

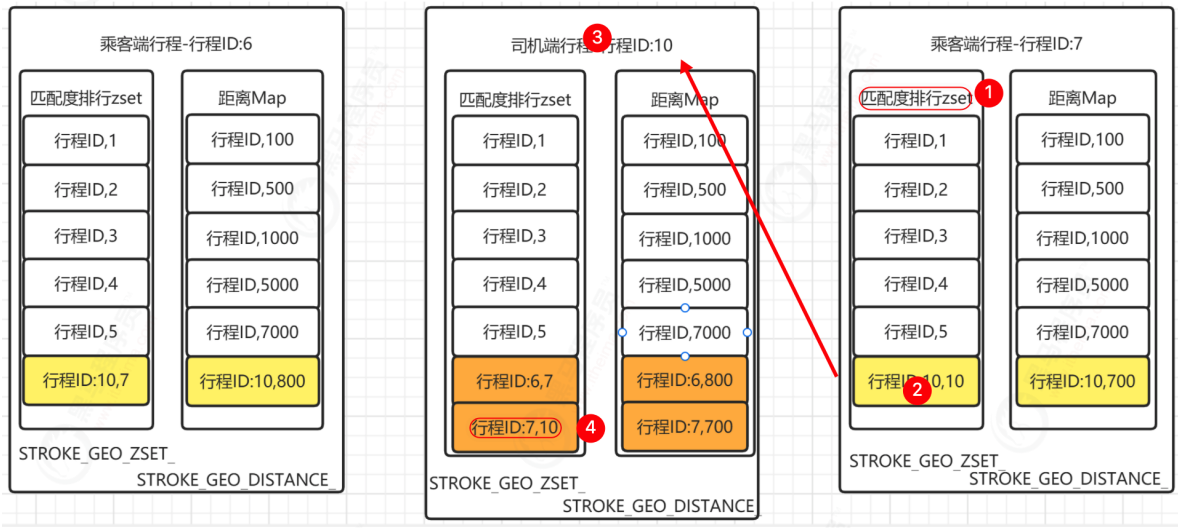
```
    }

    //已确认邀请 修改乘客状态为已邀请
    if (state == InviteState.CONFIRMED) {
        //行程ID更新，由 0 -> 1
        travelStatusChange(inviteeTripId, 0, 1);
        addOrder(inviter, invitee);
    }
    strokeVO.setRole(0);
    //删除用户zset用户信息
    unbindStroke(strokeVO);
    // 0 = 未确认， 1 = 已确认 ， 2= 已拒绝
    redisHelper.addHash(HtichConstants.STROKE_INVITE_PREFIX, inviteeTripId,
inviterTripId, String.valueOf(state.getCode()));
    redisHelper.addHash(HtichConstants.STROKE_INVITE_PREFIX, inviterTripId,
inviteeTripId, String.valueOf(state.getCode()));
    return ResponseVO.success(null);
}
```

4.5.3.5 清理缓存

> 当乘客确认了同行

- 删除本乘客的起点终点GEO坐标 - 防止别的司机新发行程时再看到
- 删除本乘客的zset，他不需要再看顺路的司机了
- 删除司机zset里的本乘客，他已经和你同行了不需要再展示

> 示意图如下,我们删除乘客行程的zset以及map后还需要将司机端的列表中将该乘客剔除出去




```
/**
  * 解除行程绑定
  *
  * @param strokeVO
  */
private void unbindStroke(StrokeVO strokeVO) {
    String unbindId = null;
    //乘客确认 解除绑定
    if (strokeVO.getRole().equals(0)) {
```

```
            unbindId = strokeVO.getInviteeTripId();

            //删除乘客端起点GEO信息
            redisHelper.delGEO(HtichConstants.STROKE_PASSENGER_GEO_START, "",
unbindId);
            //删除乘客端终点点GEO信息
            redisHelper.delGEO(HtichConstants.STROKE_PASSENGER_GEO_END, "",
unbindId);

            //司机确认 解除绑定
        } else if (strokeVO.getRole().equals(1)) {
            unbindId = strokeVO.getInviterTripId();
            //删除司机端起点GEO信息
            redisHelper.delGEO(HtichConstants.STROKE_DRIVER_GEO_START, "",
unbindId);
            //删除司机端终点点GEO信息
            redisHelper.delGEO(HtichConstants.STROKE_DRIVER_GEO_END, "", unbindId);
        }
        //解除zset绑定
        //通过乘客获取司机列表
        List<ZsetResultBO> zsetResultBOList =
redisHelper.getZsetSortVaues(HtichConstants.STROKE_GEO_ZSET_PREFIX, unbindId);
        for (ZsetResultBO zsetResultBO : zsetResultBOList) {
            //删除司机的zset关联关系
            redisHelper.delZsetByKey(HtichConstants.STROKE_GEO_ZSET_PREFIX,
zsetResultBO.getValue(), unbindId);
        }
        //删除乘客端zset
        redisHelper.delKey(HtichConstants.STROKE_GEO_ZSET_PREFIX, unbindId);
        //解除Hset绑定
        Map<String, String> driverMap =
redisHelper.getHashByMap(HtichConstants.STROKE_INVITE_PREFIX, unbindId);
        for (Map.Entry<String, String> entry : driverMap.entrySet()) {
            //已确认同行的不进行删除，用于座位数计算
            if
(entry.getValue().equals(String.valueOf(InviteState.CONFIRMED.getCode()))) {
                continue;
            }
            //删除司机的hash关联关系
            redisHelper.delHash(HtichConstants.STROKE_INVITE_PREFIX, entry.getKey(),
unbindId);
        }
        //删除乘客端hash
        redisHelper.delKey(HtichConstants.STROKE_INVITE_PREFIX, unbindId);

    }
```

4.5.4 闪电确认

> 在司机发起邀请的时候如果乘客开启了闪电确认就会不通过乘客确认自动进行确认流程,其实就是调用的我们的确定邀请的方法

```
/**
  * 闪电确认
  *
  * @param strokeVO
  */
```

```
private void quickConfirm(StrokeVO strokeVO) {
    StrokePO strokePO =
strokeAPIService.selectByID(strokeVO.getInviteeTripId());
    if (null == strokePO) {
        throw new BusinessRuntimeException(BusinessErrors.DATA_NOT_EXIST);
    }
    QuickConfirmState state =
QuickConfirmState.getState(strokePO.getQuickConfirm());
    if (state == QuickConfirmState.ENABLED) {
        strokeVO.setStatus(InviteState.CONFIRMED.getCode());
        inviteAccept(strokeVO);
    }
}
```

4.6 批量算路服务

> 我们添加订单的时候需要使用百度地图计算起始点的距离,通过距离来进行计费.

4.6.1 简介

批量算路服务（又名RouteMatrix API）是一套以HTTP/HTTPS形式提供的轻量级批量算路接口，用户可通过该服务，根据起点和终点坐标计算路线规划距离和行驶时间，RouteMatrix API V2.0支持中国大陆地区。

批量算路服务的配额和并发是按最终路线数来计算，而非RouteMatrix API请求数。如一次请求2个起点5个终点，则最终路线输出为2*5=10条，配额计为10次。

4.6.2 功能

- 批量算路目前支持驾车、摩托车、骑行（电动车/自行车）、步行。
- 根据起点和终点，批量计算路线的距离和耗时
- 融入出行策略（不走高速、常规路线、距离较短），路线和耗时计算考虑实时路况
- 驾车模式支持输入起点车头方向，提升准确性
- 步行时任意起终点之间的距离不得超过200KM，超过此限制会返回参数错误
- 一次最多计算50条路线，起终点个数之积不能超过50。 比如2个起点25个终点，50个起点1个终点等

4.6.3 适用场景

适用于高并发场景，如网约车派单、物流配送派单场景，同时发起多个起终点之间的算路，筛选所需要的订单起终点

4.6.4 相关API

```
开发api
https://lbsyun.baidu.com/index.php?title=webapi/route-matrix-api-v2

控制台：
https://lbsyun.baidu.com/apiconsole/key#/home
```

4.6.4.1 请求参数

参数名称	参数含义	类型	是否必填
ak	用户的AK	string	是
origins	纬度,经度。示例：40.056878,116.30815|40.063597,116.364973 【步骑行、摩托车】支持传入起点uid提升绑路准确性，格式为：纬度,经度;POI的uid|纬度,经度;POI的uid。示例：40.056878,116.30815;xxxxx|40.063597,116.364973;xxxxx	string	是
destinations	纬度,经度。示例：40.056878,116.30815|40.063597,116.364973 【步骑行、摩托车】支持传入终点uid提升绑路准确性，格式为：纬度,经度;POI的uid|纬度,经度;POI的uid。示例：40.056878,116.30815;xxxxx|40.063597,116.364973;xxxxx	string	是
tactics	驾车、摩托车可设置，其他无需设置。该服务为满足性能需求，不含道路阻断信息干预。驾车偏好选择，可选值如下： 10： 不走高速； 11：常规路线，即多数用户常走的一条经验路线，满足大多数场景需求，是较推荐的一个策略 12： 距离较短（考虑路况）：即距离相对较短的一条路线，但并不一定是一条优质路线。计算耗时时，考虑路况对耗时的影响； 13： 距离较短（不考虑路况）：路线同以上，但计算耗时时，不考虑路况对耗时的影响，可理解为在路况完全通畅时预计耗时。 注：除13外，其他偏好的耗时计算都考虑实时路况摩托车偏好选择，可选值如下：10： 不走高速； 11： 最短时间； 12： 距离较短。	string	否，默认为13：最短距离（不考虑路况）
riding_type	电动车、自行车骑行可设置，其他无需设置。骑行类型，筛选普通自行车、电动自行车骑行 可选值：0 普通自行车 1 电动自行车	string	否，默认为0
output	表示输出类型，可设置为xml或json。	string	否，默认为json
coord_type	坐标类型，可选值为：bd09ll（百度经纬度坐标）、bd09mc（百度墨卡托坐标）、gcj02（国测局加密坐标）、wgs84（gps设备获取的坐标）。	string	否，默认为bd09ll

4.6.4.2 返回参数

参数名	参数含义	类型	备注	
status	状态码	int	0：成功1：服务器内部错误2：参数错误	
message	返回信息	string	对status的中文描述	
result	返回的结果	array	数组形式。数组中的每个元素代表一个起点和一个终点的检索结果。顺序依次为（以2起点2终点为例）：origin1-destination1,origin1-destination2,origin2-destination1,origin2-destination2	
	distance	路线距离		
	text	线路距离的文本描述	string	文本描述的单位有米、公里两种
	value	线路距离的数值	double	数值的单位为米。若没有计算结果，值为0
duration	路线耗时			
	text	路线耗时的文本描述	string	文本描述的单位有分钟、小时两种

参数名	参数含义	类型	备注	
value	路线耗时的数值	double	数值的单位为秒。若没有计算结果，值为0	

4.6.5 路径计算工具类

> 该工具类可以通过两个坐标来进行路径规划以及计算距离以及耗时

```
public class BaiduMapClient {
    private static final String API_URL =
"https://api.map.baidu.com/routematrix/v2/driving";
    /*@Value(("${baidu.map.ak}"))*/
    private static final String ak = "xxxxxxxxxxxxxxxxxxx";

    /**
     * 路径规划
     *
     * @param origins
     * @param destinations
     * @return
     */
    public static List<RoutePlanResultBO> pathPlanning(String origins, String
destinations) {
        Map<String, String> reqMap = new HashMap<>();
        reqMap.put("ak", ak);
        reqMap.put("origins", origins);
        reqMap.put("destinations", destinations);
        String result = null;
        try {
            result = HttpClientUtils.doGet(API_URL, reqMap);
        } catch (Exception e) {
            e.printStackTrace();
        }
        return resultAssemble(result);
    }

    /**
     * 数据包装
     *
     * @param result
     * @return
     */
    public static List<RoutePlanResultBO> resultAssemble(String result) {
        List<RoutePlanResultBO> resultBOS = null;
        JSONObject jsonObject = (JSONObject) JSON.parse(result);
        if (null != jsonObject && jsonObject.getString("status").equals("0")) {
            JSONArray resultArray = jsonObject.getJSONArray("result");
            if (null != resultArray && !resultArray.isEmpty()) {
                resultBOS = resultArray.toJavaList(RoutePlanResultBO.class);
            }
        }
        return resultBOS;
    }
```

```
}
```

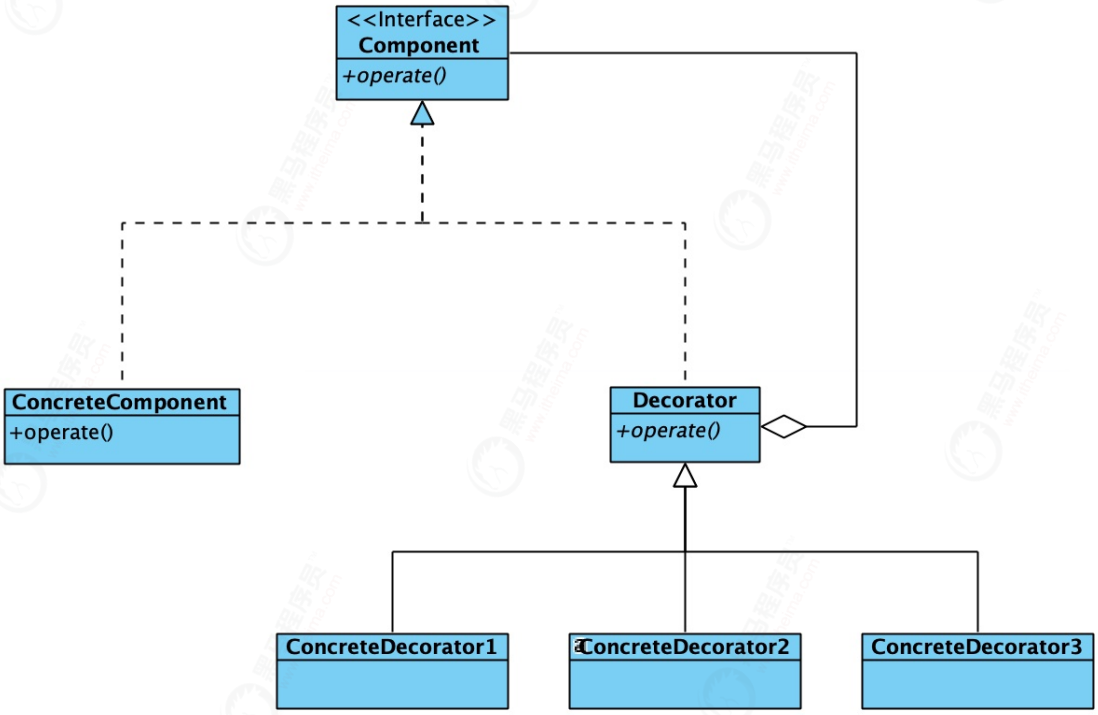
4.7 计算费用

> 这里我们使用北京的标准出租车的计费规则,3公里以内收费13元，基本单价2.3元/公里，燃油附加费1元/次

4.7.1 装饰者模式

> 这里计费规则我们使用了装饰着模式来来进行设计
>
> 设计模式可以参考下:https://baiyp.ren/tags/%E8%AE%BE%E8%AE%A1%E6%A8%A1%E5%BC%8F/

装饰器模式(Decorator Pattern)，动态地给一个对象添加一些额外的职责，就增加功能来说，装饰器模式比生成子类更灵活。



4.7.1.1 装饰器接口

> 提供装饰器的一个接口定义

```
/**
 * 计费接口
 */
public interface Valuation {
    /**
     * 计算方法
     * @param km
     * @return
     */
    float calculation(float km);
}
```

4.7.1.2 基础费用计算

```java
/**
 * 基础费率计算
 * 每公里2.3元 三公里以内不计费
 */
public class BasicValuation implements Valuation {
    /**
     * 每公里费用2.3元
     */
    private float basicPrice = 2.3F;


    @Override
    public float calculation(float km) {
        if (km <= 3) {
            return 0;
        }
        return (km - 3) * basicPrice;
    }
}
```

4.7.1.3 燃油费附加费

```java
/**
 * 燃油费附加费 一元
 */
public class FuelCostValuation implements Valuation {

    private Valuation valuation;
    private float fuelCosPrice = 1.0F;

    public FuelCostValuation(Valuation valuation) {
        this.valuation = valuation;
    }

    @Override
    public float calculation(float km) {
        return valuation.calculation(km) + fuelCosPrice;
    }
}
```

4.7.1.4 起步价

```java
/**
 * 起步价 13元
 */
public class StartPriceValuation implements Valuation {
    private Valuation valuation;

    private float startingPrice = 13.0F;

    public StartPriceValuation(Valuation valuation) {
        this.valuation = valuation;
    }

    @Override
    public float calculation(float km) {
```

```
        return valuation.calculation(km) + startingPrice;
    }
}
```

4.8 乘客上下车

4.8.1 乘客上车

> 当用户同意邀约后,司机就会发车去接该乘客,然后乘客上车

```
/**
  * 乘客上车
  *
  * @param tripid
  * @return
  */
public ResponseVO<StrokeVO> hitchhiker(String tripid) {
    //上车状态判断行程状态由 1->2
    StrokePO strokePO = travelStatusChange(tripid, 1, 2);
    return ResponseVO.success(strokePO);
}
```

> 乘客上车的主要流程是将用户的由邀请状态改为上车状态

```
/**
  * 处理行程状态变更
  *
  * @param tripid
  * @param orginStatus
  * @param targetStatus
  */
private StrokePO travelStatusChange(String tripid, int orginStatus, int
targetStatus) {
    StrokePO strokePO = new StrokePO();
    strokePO.setId(tripid);
    strokePO.setStatus(orginStatus);
    //如果查询行程状态错误则抛出异常
    List<StrokePO> strokePOList = strokeAPIService.selectlist(strokePO);
    if (strokePOList.isEmpty()) {
        throw new BusinessRuntimeException(BusinessErrors.DATA_STATUS_ERROR, "行
程状态错误");
    }
    strokePO.setStatus(targetStatus);
    strokeAPIService.update(strokePO);
    return strokePO;
}
```

4.8.2 乘客下车

> 当乘客到地方后就需要下车，下车后修改行程的状态

```
/**
  * 乘客下车
  *
  * @param tripid
  * @return
```

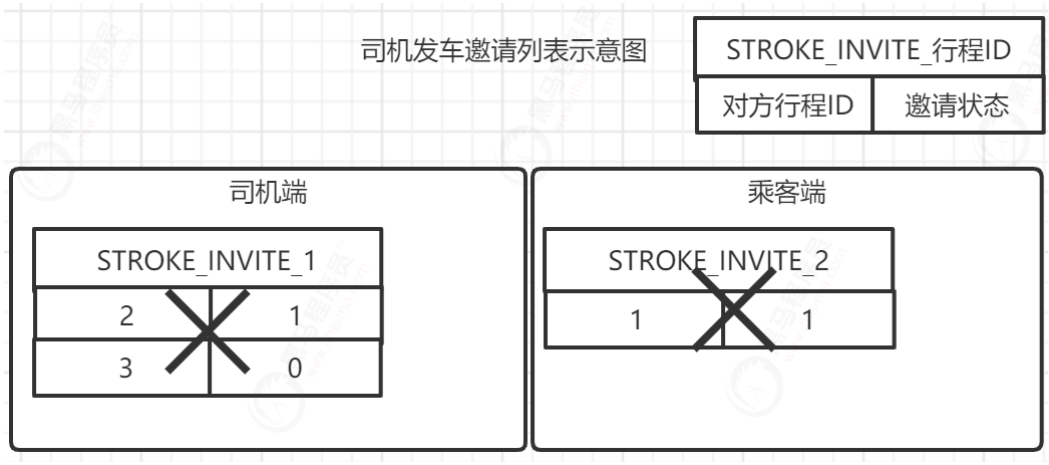
```
  */
@Transactional
public ResponseVO<StrokeVO> freeride(String tripid) {
    StrokePO strokePO = travelStatusChange(tripid, 2, 3);

    OrderPO orderPO = orderAPIService.selectByTripid(tripid);
    if (null == orderPO) {
        throw new BusinessRuntimeException(BusinessErrors.DATA_NOT_EXIST, "订单数
据不存在");
    }
    //下车车状态判断订单状态由 0->1
    orderStatusChange(orderPO.getId(), 0, 1);
    return ResponseVO.success(strokePO);
}
```

> 当乘客下车后需要修改订单的状态

```
/**
  * 订单状态变更
  *
  * @param tripid
  * @param orginStatus
  * @param targetStatus
  */

private OrderPO orderStatusChange(String tripid, int orginStatus, int
targetStatus) {
    OrderPO orderPO = new OrderPO();
    orderPO.setId(tripid);
    orderPO.setStatus(orginStatus);
    //如果查询行程状态错误则抛出异常
    List<OrderPO> orderPOList = orderAPIService.selectlist(orderPO);
    if (orderPOList.isEmpty()) {
        throw new BusinessRuntimeException(BusinessErrors.DATA_STATUS_ERROR, "订
单状态错误");
    }
    orderPO.setStatus(targetStatus);
    orderAPIService.update(orderPO);
    return orderPO;
}
```

4.9 司机发车

> 当乘客上车完成后，就需要司机发车，发车后需要删除乘客和司机的GEO数据以及改变行程状态
>
> - 删除司机的geo - 防止被新发的乘客看到
> - 删除司机的zset - 不需要看顺路的乘客了，已经发车了
> - 删除司机的邀请hset - 不需要看乘客同意还是拒绝了。已经发车
> - 删除司机的距离hset - 用不到了




```
/**
  * 司机发车
  *
  * @param tripid
  * @return
  */
public ResponseVO<StrokeVO> departure(String tripid) {
    //获取到乘客放的zset数据
    List<ZsetResultBO> zsetResultBOList =
redisHelper.getZsetSortVaues(HtichConstants.STROKE_GEO_ZSET_PREFIX, tripid);
    //在乘客列表中删除对应的司机信息
    for (ZsetResultBO zsetResultBO : zsetResultBOList) {
        //删除司机行程信息
        redisHelper.delZsetByKey(HtichConstants.STROKE_GEO_ZSET_PREFIX,
zsetResultBO.getValue(), tripid);
    }
    //删除司机的zset
    redisHelper.delKey(HtichConstants.STROKE_GEO_ZSET_PREFIX, tripid);
    //获取司机的邀请列表
    Map<String, String> inviteMap =
redisHelper.getHashByMap(HtichConstants.STROKE_INVITE_PREFIX, tripid);
    //删除乘客列表中的司机信息
    for (Map.Entry<String, String> entry : inviteMap.entrySet()) {
        redisHelper.delHash(HtichConstants.STROKE_INVITE_PREFIX, entry.getKey(),
tripid);
    }
    //删除司机的邀请 hset
    redisHelper.delKey(HtichConstants.STROKE_INVITE_PREFIX, tripid);
    //更新司机行程信息
    //行程ID更新，由 0 -> 1
    StrokePO strokePO = travelStatusChange(tripid, 0, 1);
    return ResponseVO.success(strokePO);
}
```

4.10 确认送达

> 当司机将乘客送达目的地后就需要进行点击送到后进行结算

4.10.1 更新状态

> 确认送达后还需要更新订单状态，将订单改为已送达，当前的订单是可见的，需要用户进行支付。

```
/**
  * 处理行程状态变更
  *
  * @param tripid
  * @param orginStatus
  * @param targetStatus
  */

private StrokePO travelStatusChange(String tripid, int orginStatus, int
targetStatus) {
    StrokePO strokePO = new StrokePO();
    strokePO.setId(tripid);
    strokePO.setStatus(orginStatus);
    //如果查询行程状态错误则抛出异常
    List<StrokePO> strokePOList = strokeAPIService.selectlist(strokePO);
    if (strokePOList.isEmpty()) {
        throw new BusinessRuntimeException(BusinessErrors.DATA_STATUS_ERROR, "行
程状态错误");
    }
    strokePO.setStatus(targetStatus);
    strokeAPIService.update(strokePO);
    return strokePO;
}
```

4.11 乘客支付

> 乘客送达目的地后就需要用户进行支付车费，具体支付后面单独课题讲解