## Pipeline模式

类图：

总接口：

public interface IFlightPipeLine<R,S> {  
 AirResponse<S> pipeline(R requestObject, FlightContext.AirHeader airHeader, HttpServletRequest request, HttpServletResponse response);  
}

FlightPipeline：

public class FlightPipeLine implements IFlightPipeLine<BaseRequest, BaseResponse> {  
 private static final Logger *LOGGER* = LoggerFactory.*getLogger*(FlightPipeLine.class);  
 private List<IFlightHandler<BaseRequest, BaseResponse>> handlers;  
 public AirResponse<BaseResponse> pipeline(BaseRequest requestObject, FlightContext.AirHeader airHeader, HttpServletRequest request, HttpServletResponse response) {  
 if (CollectionUtils.*isEmpty*(this.handlers)) {  
 *LOGGER*.error("handlers is empty");  
 return AirResponse.*failure*("handlers is empty");  
 }  
 FlightContext<BaseRequest, AirResponse<BaseResponse>> context = new FlightContext<BaseRequest, AirResponse<BaseResponse>>();  
 context.setRequestBody(requestObject);  
 context.setRequest(request);  
 context.setResponse(response);  
 FlightStateEvent flightStateEvent = new FlightStateEvent();  
 flightStateEvent.setFlightStateEnum(FlightStateEnum.*WORKING*);  
 try {  
 for (IFlightHandler<BaseRequest, BaseResponse> handler : handlers) {  
 handler.process(context, flightStateEvent);  
 if (FlightStateEnum.*WORKING* != flightStateEvent.getFlightStateEnum()) {  
 break;  
 }  
 }  
 } catch (Exception e) {  
 if (context.getResult() != null) {  
 context.getResult().setStatus(-1);  
 context.getResult().setErrorMessage("失败");  
 }  
 *LOGGER*.error("handler process is error .", e);  
 }  
 return context.getResult();  
 }  
  
 public void setHandlers(List<IFlightHandler<BaseRequest, BaseResponse>> handlers) {  
 this.handlers = handlers;  
 }  
}

FlightContext：

public class FlightContext<R,S> {  
 */\*\*  
 \* 输入  
 \*/* private R requestBody;  
  
 */\*\*  
 \* 输出  
 \*/* private S result;  
  
 */\*\*  
 \* 请求头  
 \*/* private AirHeader airHeader;  
  
 private HttpServletRequest request;  
 private HttpServletResponse response;

}

FlightStateEvent：

public class FlightStateEvent {  
 private FlightStateEnum flightStateEnum;

}

FlightStateEnum：

public enum FlightStateEnum {  
 WORKING, EXIT, FINISH;  
}

## handler：

### FlightValidHandler：

public class FlightSearchRequest extends BaseRequest {  
 private FlightQueryBean flightQueryBean;  
 private com.jd.airplane.gw.rpc.datacenter.FlightQueryBean bean;  
 //预定时用户选择的参数  
 private OrderFlightPara orderFlightPara;  
 //忽略处理的流程  
 private Set<Integer> ignoreSteps = new HashSet<Integer>();

}

**flightSearchRequest=**{"flightQueryBean":{"arrCity":"武汉","arrDate":"2018-10-19","arrTime":"","depCity":"青岛","depDate":"2018-10-16","lineType":"RT","queryAdtFlag":false,"queryAdtNum":0,"queryModule":"2","queryType":"jipiaoindexquery","returnType":"lowprice","secondQueryFlag":false,"source":"0","sourceId":"","sourceType":"web","uniqueKey":"","userPin":"jipiaojd","uuid":"124f5d967a674bc3a73f7fd2db257a881539582011305"},"ignoreSteps":[]}

#### 城市验证：

通过入参的城市中文名从内存中查询到？

#### 验证日期：

查询时间区段的校验，用于解决查询的时间超过一年的刷单问题  
\* 业务规则：  
\* 用户查询超过1年的机票信息时，按照自然年（例如：2014年11月5日-2014年11月04日）做最大日期限制  
\* 1.出发时间要大于或等于当天且小于当前时间 + 1年

\* 2.如果有返程时间，则要大于出发时间且小于当前时间 + 1年且 -1日  
\* 3.单程只校验出发日期， 往返还要再校验返程日期

往返是单程的出发到达日期交换。往返验证arrData（出发时间不能再当前时间之前，不能再当前时间+1年-1天之后）

queryModule：

OW("1", "单程"),  
RT\_GO("2", "往返去程"),  
RT\_RETURN("3", "往返回程");

## venderSpecific

VenderSpecificFlightDataHandler-----------**flightSearchRequest=**{"flightQueryBean":{"airways":"CA","arrAirportCode":"PVG","arrCity":"SHA","arrDate":"2019-02-13","bookingClass":"W","classNo":"","customCodes":["C9BJ10FXI"],"depAirportCode":"PEK","depCity":"PEK","depDate":"2019-02-13","flightNo":"CA1835","lineType":"OW","oneBox":"false","queryAdtFlag":true,"queryAdtNum":1,"queryModule":"1","returnType":"list","secondQueryFlag":true,"source":"3","sourceId":"621906","sourceType":"app","splitFlag":false,"uniqueKey":"98fb98fd-65d0-421c-bacd-a2f4c869d066uniqueKey","userPin":"JD\_14商城"},"ignoreSteps":[1]}

City city = dictData.getByName(name);  
return null == city ? dictData.getByCode(name) : city;

## Gw调用

### Gw中queryFlights

城市中文名

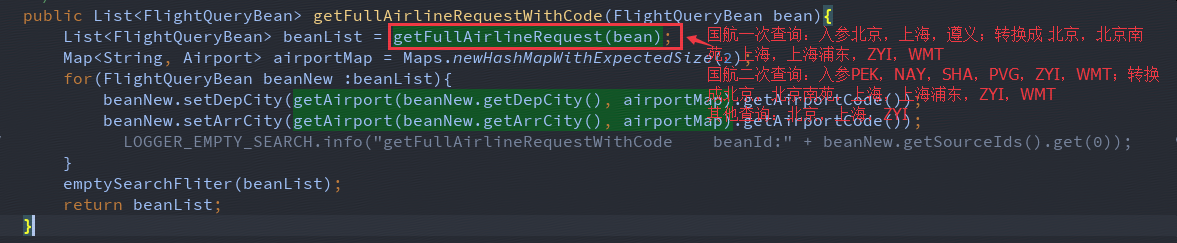
城市多机场：

com.jd.airplane.gw.manager.datacenter.impl.DataCenterFacadeImpl#queryFlights

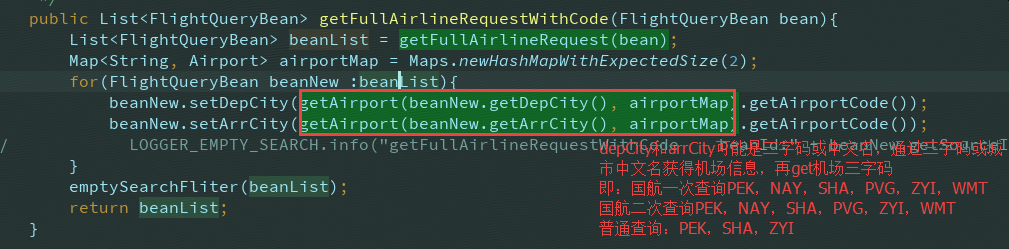
com.jd.airplane.gw.manager.datacenter.DataCenterGroup#getFullAirlineRequestWithCode通过城市名称获取机场三字码

【

【 com.jd.airplane.gw.manager.datacenter.DataCenterGroup#getFullAirlineRequest获取全部航线查询



其他查询入参是北京，上海，遵义



if 国航一次查询或深航或首航

com.jd.airplane.gw.manager.datacenter.DataCenterGroup#getSpecialCity获取特殊城市

getSpecialCity中：出发，到达

传入的参数是北京，上海，遵义中一个，则返回list<String>的值是北京，北京南苑，上海，上海浦东，ZYI，WMT。

若出发、到达都是多机场，如北京—上海，有4种组合

Else if 国航二次查询

FlightQueryBean中 DepAirportCode及arrAirportCode等于PVG或者NAY，则转换成中文名称。即NAY转成北京南苑

Else

DepCity、ArrCity是遵义，则转成ZYI

】

For循环将bean中的depCity和arrCity转换成机场三字码

】

## 缓存策略

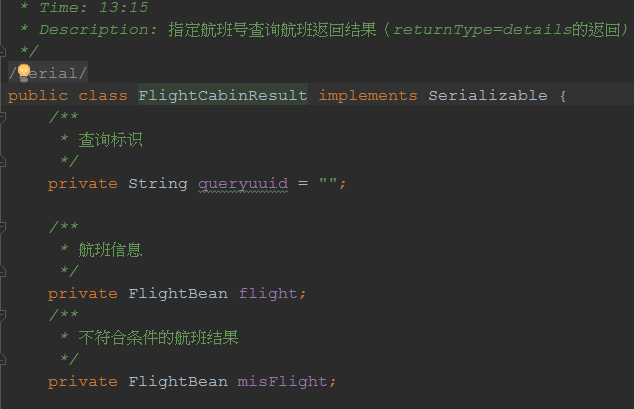
缓存1：航班查询公用的航班+舱位缓存：BASECACHE:depCity:arrCity:depDate:venderId

存储：

缓存2：航班下的舱位（13个舱位后）：QUERY\_FLIGHT\_CABINS\_uuid\_flightNo

存储：已完成，DetailsReturnHandler中15分钟

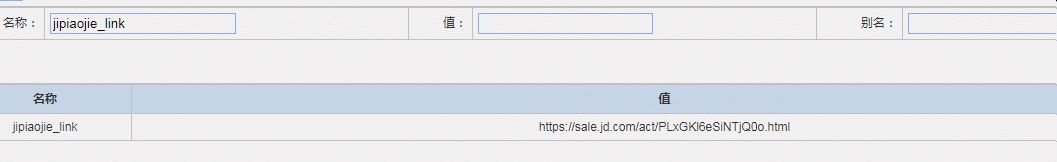




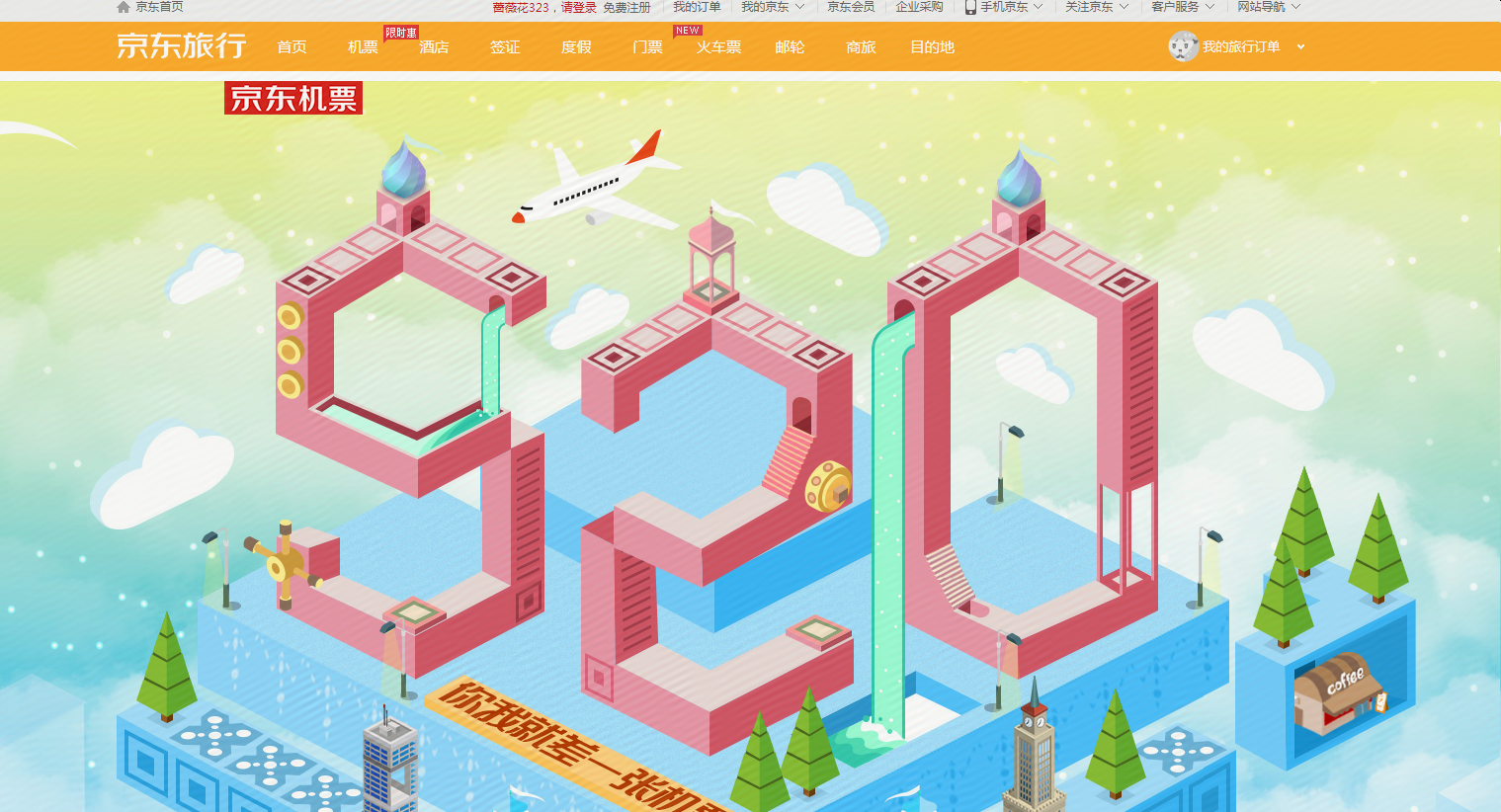
缓存时间延长：QUERY:uuid

LowPriceReturnHandler中

## 更换机票节链接：

Config中jipiaojie\_link

//sale.jd.com/act/shnvCUJR1cxDwAZ.html



<https://sale.jd.com/act/PLxGKl6eSiNTjQ0o.html>



## 振亮整理：

1、为提高前台列表查询速度，采用AB查询的设计方案：  
a 一次查询完成，只执行A查询的情况：  
       用户查询航班时，所有的有效商家航线都在Redis中缓存，则直接从缓存中抓取航班数据即可，此时一次查询结束，不再异步发起多线程调用商家接口，也不再发起B查询，此时只执行A查询；  
b 一次查询完成，AB查询都执行  
用户查询航班时，若A查询时Redis中缓存的有效商家数小于全部有效商家数，此时开启多线程异步调用商家接口（根据商家个数确定创建多少线程），只管调用商家接口，不管接口何时返回，若接口有数据返回则放进Redis里，A查询直接结束；  
随后会发起B查询，分别在1 、2、4秒发起三次B查询，并且每次限制最少商家数是2、4、6家，B查询直接从Redis中抓取有效商家的数据，根据“时间”或“商家数量”俩个维度判断是否停止查询，返回结果；  
第一次发起B查询：限制时间为1秒，有效商家数最少2家，满足其中一个条件返回；  
第二次发起B查询：限制时间为2秒，有效商家数最少4家，满足其中一个条件返回；  
第三次发起B查询：限制时间为4秒，有效商家数最少6家，满足其中一个条件返回；  
2、使用线程池创建多线程，并行查询数据：  
A查询时将缓存中没有被命中的商家，使用线程池开启多线程异步调用商家接口，不等待接口响应，只是将接口返回的数据放进Redis，避免影响界面加载航班数据的时间；  
多次发起B查询，直接从缓存中抓取商家数据，来补充接口返回的航班最新数据，数据放进Redis缓存中，也减少对商家接口调用次数；  
3、有效商家在Redis中的有效缓存时长，采用：f(n) = a + b \* x方式;  
a为基础有效时长，b为系数，x是商家查询的航班日期距离当前日期的差；若用户查询的航班日期距离当前日期越远，则有效缓存越长，反之则越短；  
4、使用管道设计模式，对搜索查询进行分层校验的设计：  
机票搜索查询的一系列业务按照管道模式设计，分别将参数校验处理器、航班查询处理器、航班舱位过滤器、航班规则处理器、航班查询后续处理器，以阀门的形式设置在管道上。依次执行校验，若前一个阀门没有通过，直接返回异常结果；

## 压测

你好，根据京东的要求，每年双十一前都要进行各种压力测试，针对国内机票业务线，需要对航班列表查询做一下压力测试，压测需求具体说明如下：

压测航线见附件**“压测航线”**

  压测航线起飞日期范围：**查询2018年11月1日-3日之间的航班**

  压测时间：**2018-10-29（周一） 22:00至 2018-10-30（周二）01:00**

压测url ：<http://jipiao.jd.com/search/queryFlight.action，get请求或post>请求

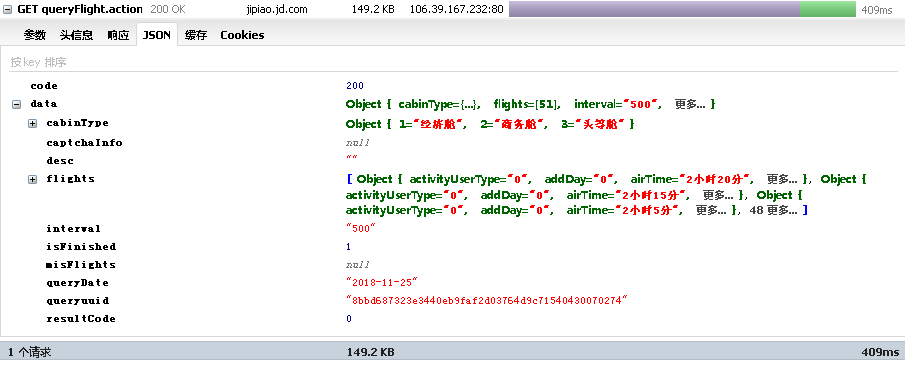
   请求参数：                  

|  |  |  |
| --- | --- | --- |
| **参数名称** | **参数说明** | **备注** |
| arrCity | 到达城市中文名 | 例如：上海 |
| arrDate | 到达日期 | 例如：2018-11-31 |
| arrTime | 到达时间 | 为空即可 |
| depCity | 出发城市中文名 | 例如：北京 |
| depDate | 起飞日期 | 例如：2018-11-31 |
| lineType | 行程类型 | 传值：OW |
| queryModule | 查询模式 | 传值：1 |
| queryType | 查询类型 | 传值：listquery |
| queryuuid | 查询唯一标示 | 使用A查询不传入参数，B查询使用A查询返回对应的 queryuuid |
| sourceId | 商家id | 为空即可 |
| uniqueKey | 唯一值 | 为空即可 |

**例如：**



**返回值说明：**



**flights 有值代表有航班结果**

**isFinished=1终止B查询**

**isFinished=0 继续B查询**

**code=200代码成功，其他失败**

**查询流程，先调用A查询，queryuuid为空，然后在调用B查询，queryuuid参数使用A查询返回结果对应的参数值传入。**