用户综合分析平台-day6~7笔记

模拟面试

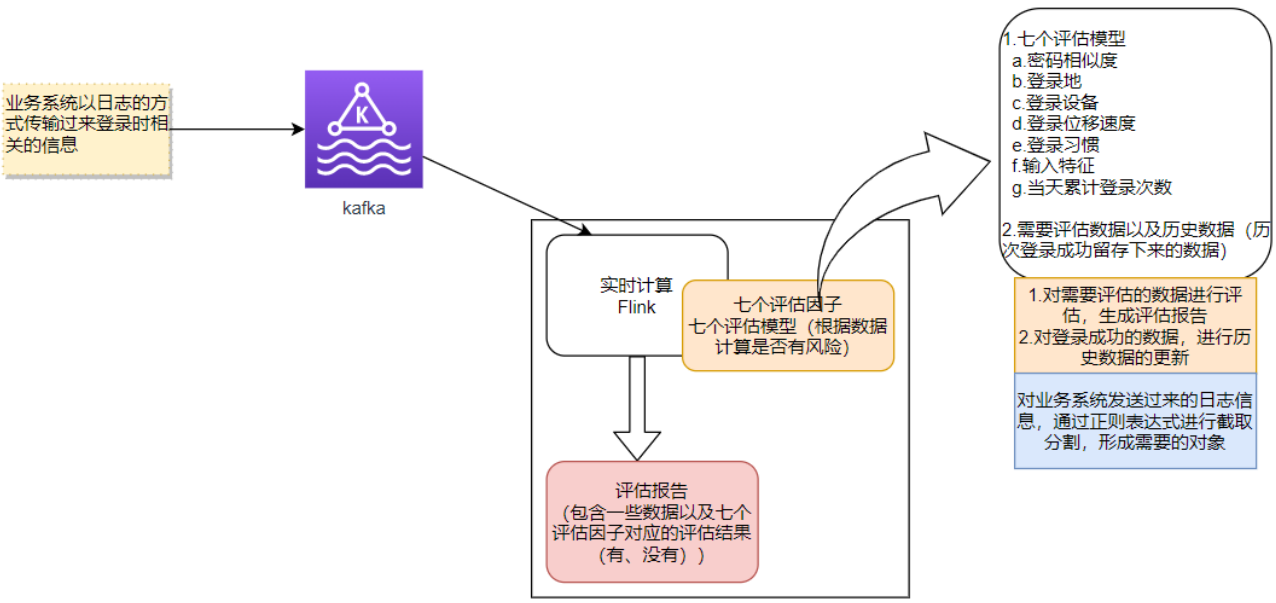
1. 2~3一组，相互提问，回答，总结
2. 做这件事件就是为了接下来的面试
3. 时间：8.30-9.30（10.00）

晚上：项目、面试题

项目：自己敲（结合着上课内容，结合着自己的理解，结合着课堂代码）

面试题：理解、同时结合面试题，把对应的知识点复习

用户登录风险模型



用户登录风险评估检查出非用户登录，或者账号被盗窃等潜在风险。通过对用户登录行为进行分析，提高了预测的准 确性。用户登录风险共分为以下七种模型：

- 用户登录地区风险识别：对用户登录地区的分析，挖掘异地登录存在的风险
- 设备来源风险识别：对用户登录时使用的设备进行分析，挖掘用户登录使用新设备时存在的风险
- 累计登录多次风险识别：对用户当天累计登录次数统计分析，挖掘当天累计登录多次存在的风险
- 用户登录习惯风险识别：通过分析用户登录时间，挖掘不在用户常用时间段登录存在的风险
- 密码相似度风险识别：通过对用户的乱序密码进行分析，挖掘密码相似度极低存在的风险
- 用户输入特征风险识别：通过对用户输入登录信息的按键习惯进行分析，挖掘按键差异性比较大存在的风险
- 用户登录位移速度风险识别：通过对用户最近两次的登录时间和登录地进行速度计算，挖掘登录位移速度异常存在的风险

根据用户登录数据并结合以上七种模型，生成登录风险评估报告，报告格式如下：

应用信息	用户ID	登录地区	经纬度	登录序列号	用户的输入特征	评估时间	登录设备	登录地区	设备	登录次数	习惯	密码相似度	输入特征	位移速度
WebApplication	zhangsan	Zhengzhou	113.65,34.76	UUID	[1500,2800,3100]	2020-06-19 10:23:16	设备信息	true	false	false	true	false	false	true

> 说明：
>
> 1. 上述报告格式中的前八列为用户登录数据，后七列为风险模型
> 2. 风险模型中的true表示该模型有风险、false表示该模型没有风险

大数据风险评估系统并不对用户的登录进行定性分析，只是当业务系统发送过来用户登录数据，由本系统对用户数据进行评估并生成评估报告。然后由业务系统根据评估报告对用户采取相应的惩罚措施。

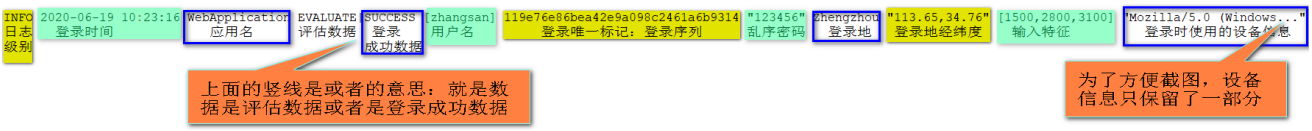
风险评估系统需要业务系统发送过来用户登录数据分为了两种：

- 需要评估的数据：本次登录的数据（在登录前需要生成评估报告的数据）。
- 登录成功的数据：登录成功产生的数据，用来作为下次评估登录的历史数据。系统留存最近的一些历史登录数据集，作为登录评估的标准

其对应的数据格式要求如下

```
INFO 2020-06-19 10:23:16 WebApplication EVALUATE|SUCCESS [zhangsan]
119e76e86bea42e9a098c2461a6b9314 "123456"
Zhengzhou "113.65,34.76"
[1500,2800,3100] "Mozilla/5.0 (Windows NT 6.1; Win64; x64) AppleWebKit/537.36 (KHTML, like
Gecko) Chrome/78.0.3904.108 Safari/537.36"
```

> 说明：
>
> EVALUATE表示需要评估的数据，SUCCESS表示登录成功的数据



数据提取

CarryXWang

1. 通过正则表达式拆分日志信息

   通过正则表达式在线测试工具`https://regex101.com/`完成原始数据拆分

- 正则表达式

```
INFO\s(\d{4}-\d{2}-\d{2}\s\d{2}:\d{2}:\d{2})\s([a-z0-9\u4e00-\u9fa5]*)\s(EVALUATE)\s\
[([a-z0-9\u4e00-\u9fa5]*)\]\s([a-z0-9]{32})\s\"([a-z0-9\.\-\,]{6,12})\"\s([a-z\u4e00-
\u9fa5]*)\s\"([0-9\.\,]*)\"\s\[([0-9\,\.]*)\]\s\"(.*)\"
```

- 测试数据

```
INFO 2020-06-19 10:23:16 WebApplication EVALUATE [zhangsan]
119e76e86bea42e9a098c2461a6b9314 "123456"
Zhengzhou "113.65,34.76"
[1500,2800,3100] "Mozilla/5.0 (Windows NT 6.1; Win64; x64) AppleWebKit/537.36 (KHTML,
like Gecko) Chrome/78.0.3904.108 Safari/537.36"
```

GENERATED CODE

```
import java.util.regex.Matcher;
import java.util.regex.Pattern;

final String regex = "INFO\\s(\\d{4}-\\d{2}-\\d{2}\\s\\d{2}:\\d{2}:\\d{2})\\s([a-z0-
9\\u4e00-\\u9fa5]*)\\s(EVALUATE)\\s\\[([a-z0-9\\u4e00-\\u9fa5]*)\\]\\s([a-z0-9]
{32})\\s\\\"([a-z0-9\\.\\-\\,]{6,12})\\\"\\s([a-z\\u4e00-\\u9fa5]*)\\s\\\"([0-
9\\.\\,]*)\\\"\\s\\[([0-9\\,\\.]*)\\]\\s\\\"(.*)\\\"";
final String string = "INFO 2020-06-19 10:23:16 WebApplication EVALUATE [zhangsan]
119e76e86bea42e9a098c2461a6b9314 \"123456\"\n"
    + "Zhengzhou \"113.65,34.76\"\n"
    + "[1500,2800,3100] \"Mozilla/5.0 (Windows NT 6.1; Win64; x64) AppleWebKit/537.36
(KHTML, like Gecko) Chrome/78.0.3904.108 Safari/537.36\"";

final Pattern pattern = Pattern.compile(regex, Pattern.MULTILINE |
Pattern.CASE_INSENSITIVE | Pattern.UNICODE_CASE);
final Matcher matcher = pattern.matcher(string);

while (matcher.find()) {

    System.out.println("Full match: " + matcher.group(0));
```

```
        for (int i = 1; i <= matcher.groupCount(); i++) {
            System.out.println("Group " + i + ": " + matcher.group(i));
        }
    }
```

代码运行结果

```
Full match: INFO 2020-06-19 10:23:16 WebApplication EVALUATE [zhangsan]
119e76e86bea42e9a098c2461a6b9314 "123456"
Zhengzhou "113.65,34.76"
[1500,2800,3100] "Mozilla/5.0 (Windows NT 6.1; Win64; x64) AppleWebKit/537.36 (KHTML,
like Gecko) Chrome/78.0.3904.108 Safari/537.36"
Group 1: 2020-06-19 10:23:16
Group 2: WebApplication
Group 3: EVALUATE
Group 4: zhangsan
Group 5: 119e76e86bea42e9a098c2461a6b9314
Group 6: 123456
Group 7: Zhengzhou
Group 8: 113.65,34.76
Group 9: 1500,2800,3100
Group 10: Mozilla/5.0 (Windows NT 6.1; Win64; x64) AppleWebKit/537.36 (KHTML, like
Gecko) Chrome/78.0.3904.108 Safari/537.36

Process finished with exit code 0
```

2. 把拆分后的日志信息封装成实体数据

```
import java.io.Serializable;

public class GeoPoint implements Serializable {
    private double longitude;//经度
    private double latitude;//纬度
    //getter/setter/constructor/toString
}
```

```
import java.io.Serializable;

public class EvaluateData implements Serializable {
    private long evaluateTime;//评估时间
    private String applicationName;//应用名
    private String userIdentify;//用户唯一标记-->用户名
    private String loginSequence;//登录序列-->登录的唯一标记
    private String ordernessPassword;//乱序密码：把密码的原本顺序打乱，保证数据的安全
    private String cityName;//登录城市
    private GeoPoint geoPoint;//登录地地理位置对象：包括经度和纬度
    private Double[] inputFeatures;//输入特征：每一个输入框花费了多长时间
    private String deviceInformation;//登录时使用的设备信息
    //getter/setter/constructor/toString

}
```

```
import java.io.Serializable;

public class LoginSuccessData implements Serializable {
    private long evaluateTime;
    private String applicationName;
    private String userIdentify;
    private String loginSequence;
    private String ordernessPassword;
    private String cityName;
    private GeoPoint geoPoint;//地理位置对象
    private Double[] inputFeatures;
    private String deviceInformation;
    //getter/setter/constructor/toString
}
```

3. 封装工具类，方便获取日志信息通过正则表达式截取之后的每一项数据

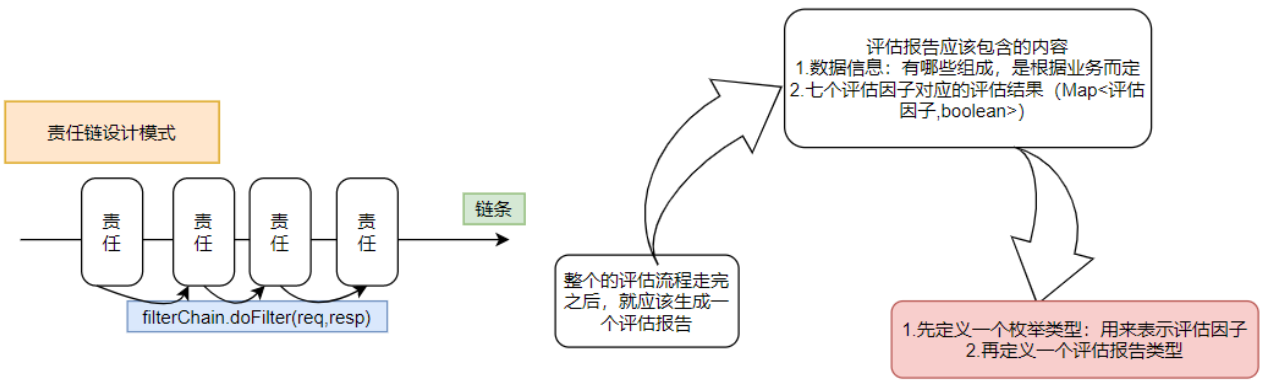
```
package com.baizhi.evaluate.util;

import com.baizhi.evaluate.entity.EvaluateData;
import com.baizhi.evaluate.entity.GeoPoint;
import com.baizhi.evaluate.entity.LoginSuccessData;

import java.text.ParseException;
import java.text.SimpleDateFormat;
import java.util.Arrays;
import java.util.Date;
import java.util.regex.Matcher;
import java.util.regex.Pattern;

public class EvaluateUtil {
    static final String EVALUATE_REGEX = "INFO\\s(\\d{4}-\\d{2}-
\\d{2}\\s\\d{2}:\\d{2}:\\d{2})\\s([a-z0-9\\u4e00-\\u9fa5]*)\\s(EVALUATE)\\s\\[([a-z0-
9\\u4e00-\\u9fa5]*)\\]\\s([a-z0-9]{32})\\s\\\"([a-z0-9\\.\\-\\,]{6,12})\\\"\\s([a-z\\u4e00-
\\u9fa5]*)\\s\\\"([0-9\\.\\,]*)\\\"\\s\\[([0-9\\,\\.]*)\\]\\s\\\"(.*)\\\"";
    static final String SUCCESS_REGEX = "INFO\\s(\\d{4}-\\d{2}-
\\d{2}\\s\\d{2}:\\d{2}:\\d{2})\\s([a-z0-9\\u4e00-\\u9fa5]*)\\s(SUCCESS)\\s\\[([a-z0-
9\\u4e00-\\u9fa5]*)\\]\\s([a-z0-9]{32})\\s\\\"([a-z0-9\\.\\-\\,]{6,12})\\\"\\s([a-z\\u4e00-
\\u9fa5]*)\\s\\\"([0-9\\.\\,]*)\\\"\\s\\[([0-9\\,\\.]*)\\]\\s\\\"(.*)\\\"";
//    static final String string = "INFO 2020-06-19 10:23:16 WebApp0汉子lication EVALUATE
[zhang第三方san] 119e76e86bea42e9a098c2461a6b9314 \"123456\"\n"
//            + "Zhengzhou \"113.65,34.76\"\n"
//            + "[1500,2800,3100] \"Mozilla/5.0 (Windows NT 6.1; Win64; x64)
AppleWebKit/537.36 (KHTML, like Gecko) Chrome/78.0.3904.108 Safari/537.36\"";

    static final Pattern EVALUATE_PATTERN = Pattern.compile(EVALUATE_REGEX,
Pattern.MULTILINE | Pattern.CASE_INSENSITIVE | Pattern.UNICODE_CASE);


    static final Pattern SUCCESS_PATTERN = Pattern.compile(SUCCESS_REGEX, Pattern.MULTILINE
```

```
| Pattern.CASE_INSENSITIVE | Pattern.UNICODE_CASE);
//    static final Matcher EVALUATE_MATCHER = EVALUATE_PATTERN.matcher(string);


    /**
     * 判断是否是评估数据;这个方法同时还能判断数据是否符合整体要求
     * @param data 日志信息(登录之前业务系统提交的数据，需要评估的数据)
     * @return Boolean类型，如果data是评估数据，就返回true
     */
    public static boolean isEvaluateData(String data){
        return EVALUATE_PATTERN.matcher(data).find();
    }

    /**
     * 判断是否是登录成功数据；这个方法同时还能判断数据是否符合整体要求
     * @param data 日志信息（登录成功之后业务系统提交的数据）
     * @return Boolean类型，如果data是登录成功数据，就返回true
     */
    public static boolean isSuccessData(String data){
        return SUCCESS_PATTERN.matcher(data).find();
    }

    /**
     * 获取到评估时间
     * @param data
     * @return
     */
    public static Date getEvaluateDate(String data){
        if(isEvaluateData(data)){
            Matcher matcher = EVALUATE_PATTERN.matcher(data);
            String group1 = matcher.group(1);
            SimpleDateFormat simpleDateFormat = new SimpleDateFormat("yyyy-MM-dd
HH:mm:ss");
            Date date = null;
            try {
                date = simpleDateFormat.parse(group1);
            } catch (ParseException e) {
                e.printStackTrace();
            }
            return date;
        }
        return null;
    }

//================================下面这个方法是这个工具类中的重点方法=====================
    /**
     * 可以获取到需要的任何一种数据（九个数据中的任何一个）
     * @param data 日志信息(登录之前业务系统提交的数据，需要评估的数据)
     * @param evaluateDataType  定义的枚举类型，用来限制在使用这个方法的时候需要传递的参数类型
     * @return
     */
    public static Object getData(String data,EvaluateDataType evaluateDataType){

        int index = evaluateDataType.getIndex();
```

```
        if(isEvaluateData(data)){
            Matcher matcher = EVALUATE_PATTERN.matcher(data);
            if(matcher.find()){
                String result = matcher.group(index);
                switch (index) {
                    case 1:
                        //需要获取的数据是评估时间
                        SimpleDateFormat simpleDateFormat = new SimpleDateFormat("yyyy-MM-
dd HH:mm:ss");
                        Date date = null;
                        try {
                            date = simpleDateFormat.parse(result);
                        } catch (ParseException e) {
                            e.printStackTrace();
                        }
                        return date;
                    case 8:
                        String[] split = result.split(",");
                        return new GeoPoint(Double.parseDouble(split[0]),
Double.parseDouble(split[1]));
                    case 9:
                        String[] inputFeaturesString = result.split(",");
                        //把string数组转换成double数组:通过JDK新特性Stream完成转换
                        double[] inputFeatures =
Arrays.asList(inputFeaturesString).stream().mapToDouble(Double::valueOf).toArray();
                        return inputFeatures;
                    default:
                        return result;
                }
            }

        }
        return null;
    }


    /**
     * 把所有的数据封装形成一个实体对象-->评估数据
     * @param data
     * @return
     */
    public static EvaluateData getEvaluateData(String data){
        return null;
    }

    /**
     * 把所有的数据封装形成一个实体对象-->登录成功数据
     * @param data
     * @return
     */
    public static LoginSuccessData getLoginSuccessData(String data){
        return null;

    }
```

```
    public static void main(String[] args) {

     final String data = "INFO 2020-06-19 10:23:16 WebApp0汉子lication EVALUATE [zhang第三方
san] 119e76e86bea42e9a098c2461a6b9314 \"123456\"\n"
            + "Zhengzhou \"113.65,34.76\"\n"
            + "[1500,2800,3100] \"Mozilla/5.0 (Windows NT 6.1; Win64; x64)
AppleWebKit/537.36 (KHTML, like Gecko) Chrome/78.0.3904.108 Safari/537.36\"";

     //getData(data,EvaluateDataType.GEO_POINT);
        /*Date date = (Date) getData(data, EvaluateDataType.EVALUATE_DATE);
        System.out.println(date);*/
        double[] result = (double[]) getData(data, EvaluateDataType.INPUT_FEATURES);
        for (double v : result) {
            System.out.println(v+2);
        }

//        System.out.println(isEvaluateData(data));
//        while (matcher.find()) {
//            System.out.println("Full match: " + matcher.group(0));
//            for (int i = 1; i <= matcher.groupCount(); i++) {
//                System.out.println("Group " + i + ": " + matcher.group(i));
//            }
//        }
    }
}
```

评估因子实现



- 相关实体类

```
import java.io.Serializable;
import java.util.List;
import java.util.Map;
import java.util.Set;


/**
```

```
 * 记录用户登录的历史数据
 */
public class HistoryData implements Serializable {
    //记录用户历史登录城市
    private Set<String> historyCities;
    //记录用户历史登录设备
    private List<String> historyDeviceInformations;
    //记录用户当天登录次数
    private Integer currentDayLoginCount;
    //记录用户历史登录习惯:key是星期，value是map(key是小时，value是登录次数)
    private Map<String, Map<String,Integer>> historyLoginTimeSlot;
    //记录用户历史密码
    private Set<String> historypasswords;
    //记录用户历史输入特征
    private List<Double[]> latestInputFeatures;
    //记录最近一次历史登录时间
    private long lastLoginTime;
    //记录最近一次历史登录经纬度
    private GeoPoint lastLoginGeoPoint;
    //getter/setter/constructor/toString
}
```

```
/**
 * 七种模型对应的七个风险因子
 */
public enum  RiskFactor {

    //登录地区评估
    AREA("area"),
    //设备评估
    DEVICE("device"),
    //登录次数评估
    TOTAL("total"),
    //登录时间段评估--登录习惯
    TIMESLOT("timeslot"),
    //密码相似度评估
    SIMILARITY("similarity"),
    //输入特征评估
    INPUTFEATURE("inputfeature"),
    //位移速度评估
    SPEED("speed");

    //风险因子对应的名字描述
    private String name;
    RiskFactor(String name){
        this.name=name;
    }
}
```

```
import java.util.*;
```

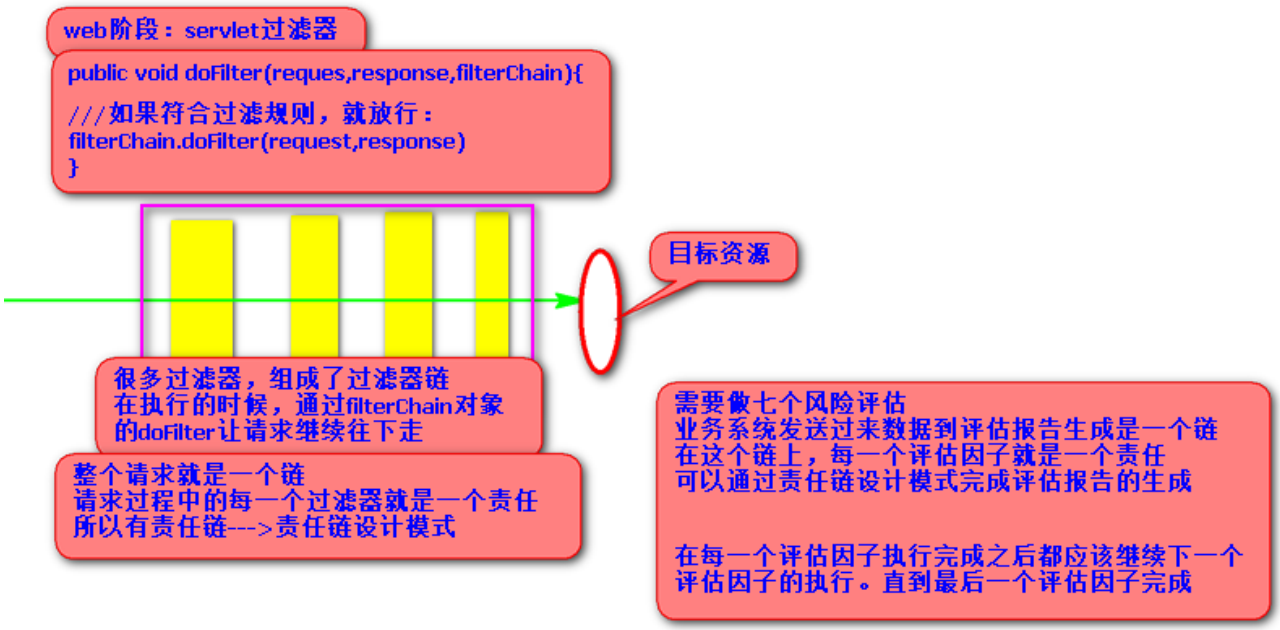
```java
/**
 * 评估报告类
 */
public class EvaluateReport {
    private String applicationName;
    private String userIdentify;
    private String loginSequence;
    private long evaluateTime;
    private String cityName;
    private GeoPoint geoPoint;

    public EvaluateReport() {
    }

    //用来存储评估因子
    //key就是评估因子；value是Boolean类型；表示该评估因子是否有风险。true:有风险，false:没有风险
    private Map<RiskFactor,Boolean> metrics=new HashMap<RiskFactor,Boolean>();

    public void signReport(RiskFactor riskFactor,boolean flag){
        metrics.put(riskFactor,flag);
    }

    public EvaluateReport(String applicationName, String userIdentify, String loginSequence,
long evaluateTime, String cityName, GeoPoint geoPoint) {
        this.applicationName = applicationName;
        this.userIdentify = userIdentify;
        this.loginSequence = loginSequence;
        this.evaluateTime = evaluateTime;
        this.cityName = cityName;
        this.geoPoint = geoPoint;

        //初始化所有风险因子都是false
        metrics.put(RiskFactor.AREA,false);
        metrics.put(RiskFactor.DEVICE,false);
        metrics.put(RiskFactor.SIMILARITY,false);
        metrics.put(RiskFactor.SPEED,false);
        metrics.put(RiskFactor.TIMESLOT,false);
        metrics.put(RiskFactor.INPUTFEATURE,false);
        metrics.put(RiskFactor.TOTAL,false);
    }

    public String getApplicationName() {
        return applicationName;
    }

    public String getUserIdentify() {
        return userIdentify;
    }

    public String getLoginSequence() {
        return loginSequence;

    }
```

```
    public long getEvaluateTime() {
        return evaluateTime;
    }

    public String getCityName() {
        return cityName;
    }

    public GeoPoint getGeoPoint() {
        return geoPoint;
    }

    public Map<RiskFactor, Boolean> getMetrics() {
        return metrics;
    }

    /**
     * 重新toString方法，评估因子按照字母顺序排列，如下所示
     * AREA DEVICE INPUTFEATURE SIMILARITY SPEED TIMESLOT TOTAL
     * true false    false        false    true   false     false
     * @return
     */

    @Override
    public String toString(){
        //先根据风险因子的名字做排序
        //把风险因子对应的值用空格拼接起来
        String report=metrics.keySet()
                .stream()
                .sorted((RiskFactor o1, RiskFactor o2) -> o1.name().compareTo(o2.name()))
                .map(riskFactor -> metrics.get(riskFactor)+"")
                .reduce((v1,v2)->v1+" "+v2)
                .get();
        return applicationName+" "+userIdentify+" "+loginSequence+" "+evaluateTime+" "+cityName+
" "+geoPoint.getLongitude()+","+geoPoint.getLatitude()+" "+report;

    }
}
```

- 各评估因子对应的抽象类

```
import com.baizhi.eveluate.entity.EvaluateData;
import com.baizhi.eveluate.entity.EvaluateReport;
import com.baizhi.eveluate.entity.HistoryData;
import com.baizhi.eveluate.entity.RiskFactor;
import com.baizhi.eveluate.util.EvaluateChain;

public abstract class Evaluate {
    //指定评估因子名字
    private RiskFactor riskFactor;
```

```
    public Evaluate(RiskFactor riskFactor) {
        this.riskFactor = riskFactor;
    }
    public RiskFactor getRiskFactor() {
        return riskFactor;
    }
    /**
     * 评估：这是一个抽象方法
     * @param evaluateData 需要评估的数据
     * @param historyData 历史数据
     * @param evaluateReport 评估报告
     * @param evaluateChain:这是一个评估链，用来驱动下一个Evaluate实例
     */
    public abstract void eval(EvaluateData evaluateData, HistoryData historyData, EvaluateReport
evaluateReport, EvaluateChain evaluateChain);
}
```



- 责任链类

```
import com.baizhi.eveluate.entity.EvaluateData;
import com.baizhi.eveluate.entity.EvaluateReport;
import com.baizhi.eveluate.entity.HistoryData;
import com.baizhi.eveluate.evaluate.Evaluate;

import java.util.List;

/**
 * 评估因子责任链类
 */
public class EvaluateChain {
    private int position=0;
    private List<Evaluate> evaluates;//该链上的所有责任
```

```
    public EvaluateChain(List<Evaluate> evaluates) {
        this.evaluates = evaluates;
    }

    public void doChain(EvaluateData evaluateData, HistoryData historyData, EvaluateReport
evaluateReport){
        if(position < evaluates.size()){
            //获取一个责任
            Evaluate evaluate = evaluates.get(position);
            position +=1;
            evaluate.eval(evaluateData,historyData,evaluateReport,this);
        }
    }
}
```

登录地区评估

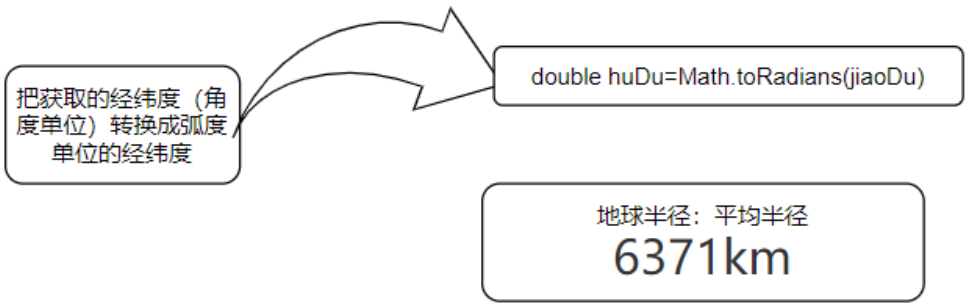
登录地评估：
1.保留N次最近登录成功的登录地
2.如果当前要评估数据的登录地不在留存的数据中，就认定有风险；否则，认定无风险

登录设备评估：
1.保留N次最近登录成功的设备信息
2.如果当前要评估数据的登录设备不在留存的数据中，就认定有风险；否则，认定无风险

当天累计登录次数评估：
如果登录次数超过了规定的登录次数，就认定有风险

登录位移速度评估：
1.需要上一次登录成功的地理位置以及时间
2.根据要评估数据的地理位置以及时间；计算距离、时间
3.根据球面距离数学公式计算：两个经纬度 间的球面距离

球面距离计算公式：d(x1,y1,x2,y2)=r*arccos(sin(x1)*sin(x2)+cos(x1)*cos(x2)*cos(y1-y2))，其中，x1，y1是纬度\经度的弧度单位，r为地球半径。 [2]



- 需要评估的数据中有登录地区
- 历史数据中有历次登录成功留存的登录地区

当要评估的登录地区不在留存的历史数据中，表示该评估因子有风险

```
import com.baizhi.eveluate.entity.EvaluateData;
import com.baizhi.eveluate.entity.EvaluateReport;
import com.baizhi.eveluate.entity.HistoryData;
import com.baizhi.eveluate.entity.RiskFactor;
import com.baizhi.eveluate.evaluate.Evaluate;
import com.baizhi.eveluate.util.EvaluateChain;

import java.util.Set;

/**
 * 登录地区评估
 */
public class AreaEvaluate extends Evaluate {
    public AreaEvaluate() {
        super(RiskFactor.AREA);
    }

    @Override
    public void eval(EvaluateData evaluateData, HistoryData historyData, EvaluateReport
evaluateReport, EvaluateChain evaluateChain) {

 evaluateReport.signReport(getRiskFactor(),doEval(evaluateData.getCityName(),historyData.getHist
oryCities()));
        evaluateChain.doChain(evaluateData,historyData,evaluateReport);
    }
    public boolean doEval(String cityName, Set<String> historyCities){
        if(historyCities==null || historyCities.size()==0){//说明是第一次使用
            return false;
        }else{//如果在当前登录城市登录过就没有风险
            return !historyCities.contains(cityName);
        }
    }
}
```

使用设备评估

- 需要的数据：当前登录使用的设备
- 历史数据：历次登录成功留存下来的所使用的设备

当前登录使用的设备，不在历史数据里面，就任务该评估因子有风险

```
import com.baizhi.eveluate.entity.EvaluateData;
import com.baizhi.eveluate.entity.EvaluateReport;
import com.baizhi.eveluate.entity.HistoryData;
import com.baizhi.eveluate.entity.RiskFactor;
import com.baizhi.eveluate.evaluate.Evaluate;
import com.baizhi.eveluate.util.EvaluateChain;

import java.util.List;
```

```
/**
 * 使用设备评估
 */
public class DeviceEvaluate extends Evaluate {
    public DeviceEvaluate() {
        super(RiskFactor.DEVICE);
    }
    @Override
    public void eval(EvaluateData evaluateData, HistoryData historyData, EvaluateReport
evaluateReport, EvaluateChain evaluateChain) {

 evaluateReport.signReport(getRiskFactor(),doEval(evaluateData.getDeviceInformation(),historyDat
a.getHistoryDeviceInformations()));
        evaluateChain.doChain(evaluateData,historyData,evaluateReport);
    }
    public boolean doEval(String deviceInformation, List<String> historyDeviceInformations){
        if(historyDeviceInformations==null || historyDeviceInformations.size()==0){//说明是第一次
使用
            return false;
        }else{//使用过当前登录设备，没有风险
            return !historyDeviceInformations.contains(deviceInformation);
        }
    }
}
```

累计登录次数评估

- 在程序中设定运行用户登录的最大次数==》就是一个阈值threshold
- 需要当天登录的次数：留存下来的，当天登录成功的次数

如果当天登录成功次数超过了设定的阈值，任务该评估因子有风险

```
import com.baizhi.eveluate.entity.EvaluateData;
import com.baizhi.eveluate.entity.EvaluateReport;
import com.baizhi.eveluate.entity.HistoryData;
import com.baizhi.eveluate.entity.RiskFactor;
import com.baizhi.eveluate.evaluate.Evaluate;
import com.baizhi.eveluate.util.EvaluateChain;

/**
 * 当天累计登录次数评估
 */
public class TotalEvaluate extends Evaluate {
    //设置阈值 允许用户最大登录次数
    private Integer threshold;
    public TotalEvaluate(Integer threshold) {
        super(RiskFactor.TOTAL);
        this.threshold=threshold;
    }
```

```
    @Override
    public void eval(EvaluateData evaluateData, HistoryData historyData, EvaluateReport
evaluateReport, EvaluateChain evaluateChain) {

 evaluateReport.signReport(getRiskFactor(),doEval(historyData.getCurrentDayLoginCount()));
        evaluateChain.doChain(evaluateData,historyData,evaluateReport);
    }
    public boolean doEval(Integer currentDayLoginCount){
        if(currentDayLoginCount==null){//说明是第一次使用
            return false;
        }else{//如果登录超过限定次数,认定有风险
            return currentDayLoginCount >= threshold;
        }
    }
}
```

登录习惯评估

```
package com.baizhi.em.evaluate;

import com.baizhi.em.entity.EvaluateData;
import com.baizhi.em.entity.EvaluateReport;
import com.baizhi.em.entity.HistoryData;
import com.baizhi.em.entity.RiskFactor;

import java.text.DecimalFormat;
import java.util.*;
import java.util.stream.Stream;

/**
 * 登录习惯的评估
 * 根据当前需要评估的数据的登录时间，结合历史留存的登录习惯(Map)完成评估
 */
public class TimeslotEvaluate extends Evaluate {
    private int countThreshold;
    //根据这个值判断用户的登录习惯有没有养成
    //如果用户的总登录次数大于了这个值，说明用户的登录习惯已经形成

    public TimeslotEvaluate(int countThreshold) {
        super(RiskFactor.TIMESLOT);
        this.countThreshold = countThreshold;
    }

    @Override
    public void eval(EvaluateData evaluateData, HistoryData historyData, EvaluateReport
evaluateReport, EvaluateChain evaluateChain) {

        //1.对登录习惯进行评估


        boolean
```

```
isRisk=doEval(evaluateData.getEvaluateTime(),historyData.getHistoryLoginTimeSlot());
        evaluateReport.signReport(RiskFactor.TIMESLOT,isRisk);
        //2.让评估继续
        evaluateChain.doChain(evaluateData, historyData, evaluateReport);
    }

    private boolean doEval(long evaluateTime, Map<String, Map<String, Integer>>
historyLoginTimeSlot) {

        //1.如果是第一次登录，没有风险；return false;
        if(historyLoginTimeSlot==null || historyLoginTimeSlot.size()==0){
            return false;
        }

        //2.如果登录总次数没有到达规定的值(this.countThreshold),用户的登录习惯还没有形成，没有风险
        //通过historyLoginTimeSlot计算总登录次数
        /*
        //通过流计算登录次数
        Collection<Map<String, Integer>> values = historyLoginTimeSlot.values();//获取到map中的所
有value（每一个value是一个Map）
        Stream<Integer> integerStream = values.stream().map(e -> e.values().stream().reduce((v1,
v2) -> v1 + v2).get());//一天的总登录次数
        Integer totalCount = integerStream.reduce((v1, v2) -> v1 + v2).get();//总登录次数
        */

        //用循环计算登录次数
        int totalCount=0;
        Collection<Map<String, Integer>> values = historyLoginTimeSlot.values();//获取到所有的
value
        for (Map<String, Integer> map : values) {
            Collection<Integer> values1 = map.values();//
            for (Integer integer : values1) {
                totalCount+=integer;
            }
        }

        if(totalCount<countThreshold){
            //登录习惯还没有养成
            return false;
        }

        //登录习惯已经形成啦，需要判断，当前的登录是否符合用户的登录习惯
        //1.如果要评估的数据的星期，在map中不存在；不符合用户的登录习惯，认定该评估项有风险
        //2.星期在map中存在，根据星期，从map中读取一个值（map2）;如果要评估的数据的小时，在map2中不存
在,不符合用户的登录习惯
        //3.小时在map2中存在

        //运用工具类计算出用户登录的时间为星期几
        String[] WEEKS={"星期日","星期一","星期二","星期三","星期四","星期五","星期六"};//定义一个数
组

        Date date = new Date(evaluateTime);//指定的时间

//        int date1 = date.getDate();//几号
```

```
        int day = date.getDay();//星期，0表示周日，1表示周一...6表示周六
        String week = WEEKS[day];//星期
        int hours = date.getHours();//小时

//        Calendar calendar = Calendar.getInstance();//创建一个Calendar对象；日历对象;获取到的就是当前时间的日历对象
//        calendar.setTimeInMillis(evaluateTime);
//        String dayOfWeek = WEEKS[calendar.get(Calendar.DAY_OF_WEEK) - 1];
//        //计算出用户登录的时间小时数 转换为二位数字  01 02 ... 24
//        DecimalFormat decimalFormat=new DecimalFormat("00");
//        String hourOfDay=  decimalFormat.format(calendar.get(Calendar.HOUR_OF_DAY));

        if(!historyLoginTimeSlot.containsKey(week)){//评估的数据的星期，在map中不存在
            //不符合用户的登录习惯
            return true;//有风险
        }

        Map<String, Integer> map2 = historyLoginTimeSlot.get(week);
        if(!map2.containsKey(hours)){//要评估的数据的小时，在map2中不存在
            //不符合用户的登录习惯
            return true;//有风险

        }
        //获取当前hours这个时间点的登录次数
        Integer currentHoursCount = map2.get(hours);
        //从map2中获取到所有小时的登录次数，按照升序排列，取三分之二位置处作为判断的阈值
        List<Integer> values1 = (List<Integer>) map2.values();//??????
        Collections.sort(values1);
        Integer threshold = values1.get(values1.size() * 2 / 3);//取一个可以判断的阈值：取排序之后的数据的三分之二位置作为判断的阈值

        //如果当前时间点的登录次数大于判断的阈值，就认定没有风险
        return !(currentHoursCount>threshold);
    }
}
```

密码相似度评估

```
import com.baizhi.eveluate.entity.EvaluateData;
import com.baizhi.eveluate.entity.EvaluateReport;
import com.baizhi.eveluate.entity.HistoryData;
import com.baizhi.eveluate.entity.RiskFactor;
import com.baizhi.eveluate.evaluate.Evaluate;
import com.baizhi.eveluate.util.EvaluateChain;

import java.util.*;
import java.util.stream.Collectors;


/**
```

```
 * 密码相似度评估
 */
public class SimilarityEvaluate extends Evaluate {
    private Double threshold;

    public SimilarityEvaluate(Double threshold) {
        super(RiskFactor.SIMILARITY);
        this.threshold = threshold;
    }

    @Override
    public void eval(EvaluateData evaluateData, HistoryData historyData, EvaluateReport
evaluateReport, EvaluateChain evaluateChain) {

 evaluateReport.signReport(getRiskFactor(),doEval(evaluateData.getOrdernessPassword(),historyDat
a.getHistorypasswords()));
        evaluateChain.doChain(evaluateData,historyData,evaluateReport);
    }

    /**
     * 通过用户输入密码与历史密码进行余弦相似度对比设定阈值判断是否输入密码有风险
     * @param ordernessPassword 用户当前输入密码(可以乱序)
     * @param historypasswords 用户历史密码(可以乱序)
     * @return
     */
    public boolean doEval(String ordernessPassword, Set<String> historypasswords){
        //如果用户没有历史密码将直接判断无风险
        if (historypasswords==null || historypasswords.size()==0){
            return false;
        }
        //构建词条库
        HashSet<Character> chars = new HashSet<>();
        //将历史密码放入词条库
        historypasswords.stream().forEach(historypassword->{
            for (char c:historypassword.toCharArray()) {
                chars.add(c);
            }
        });
        //将当前输入密码放入词条库，富化词条库
        for (char c:ordernessPassword.toCharArray()) {
            chars.add(c);
        }
        //对词条进行排序
        List<Character> wordBag = chars.stream().sorted().collect(Collectors.toList());
        System.out.println(wordBag);
        //将所有历史密码转换为特征向量
        List<Integer[]> vectors = historypasswords.stream().map(historypassword ->
converStringToVector(wordBag, historypassword)).collect(Collectors.toList());
        //将当前输入密码转换为特征向量
        Integer[] vector = converStringToVector(wordBag, ordernessPassword);
        //返回通过余弦相似定理 求得当前密码 与 历史密码的 特征向量的相似度百分比 高于阈值的 百分比
        List<Double> resultList = vectors.stream().map(v1 -> {

            double similarity = calculateSimilarity(v1, vector);
```

```
            System.out.println(similarity);
            return similarity;
        }).filter(similarity -> similarity >= threshold).collect(Collectors.toList());
        //如果高于阈值则判定无风险，如果低于判定有风险
        return resultList.size() == 0;
    }

    /**
     * 通过用户历史密码生成的词条库比对历史密码和当前密码生成特征向量
     * @param wordBag 用户历史密码生成的词条库
     * @param password 历史密码和当前密码
     * @return
     */
    private Integer[] converStringToVector(List<Character> wordBag,String password){
        //创建一个空的特征向量
        Integer[] vector = new Integer[wordBag.size()];
        //创建一个Map用来统计密码的相同字符数量
        HashMap<Character, Integer> charMap = new HashMap<>();
        //统计字符数量
        for (Character c : password.toCharArray()) {
            Integer count = 1;
            if (charMap.containsKey(c)){
                count+=charMap.get(c);
            }
            charMap.put(c,count);
        }
        //通过密码的字符数量，对比词条库生成特征向量
        for (int i = 0; i < wordBag.size(); i++) {
            Character c = wordBag.get(i);
            vector[i] = charMap.containsKey(c)?charMap.get(c):0;
        }
        return vector;
    }

    /**
     * 通过余弦相似定理 求得当前密码 与 历史密码的 特征向量的相似度百分比
     * @param v1 历史密码的特征向量
     * @param v2 当前密码的特征向量
     * @return
     */
    private double calculateSimilarity(Integer[] v1,Integer[] v2){
        //求得分子
        Double sum = 0.0;
        for (int i = 0; i < v1.length; i++) {
            sum+=v1[i]*v2[i];
        }
        //求得分母
        Integer powSum1 = Arrays.stream(v1).map(integer -> integer * integer).reduce((i1, i2) ->
i1 + i2).get();
        Integer powSum2 = Arrays.stream(v2).map(integer -> integer * integer).reduce((i1, i2) ->
i1 + i2).get();
        //计算相似百分比返回

        return sum/(Math.sqrt(powSum1)*Math.sqrt(powSum2));
```

```
    }

}
```

相关数学模型：余弦相似度

用户输入特征评估

```
import com.baizhi.eveluate.entity.EvaluateData;
import com.baizhi.eveluate.entity.EvaluateReport;
import com.baizhi.eveluate.entity.HistoryData;
import com.baizhi.eveluate.entity.RiskFactor;
import com.baizhi.eveluate.evaluate.Evaluate;
import com.baizhi.eveluate.util.EvaluateChain;

import java.util.ArrayList;
import java.util.Arrays;
import java.util.Comparator;
import java.util.List;

/**
 * 用户输入特征评估
 */
public class InputfeatureEvaluate extends Evaluate {
    public InputfeatureEvaluate() {
        super(RiskFactor.INPUTFEATURE);
    }

    @Override
    public void eval(EvaluateData evaluateData, HistoryData historyData, EvaluateReport
evaluateReport, EvaluateChain evaluateChain) {

 evaluateReport.signReport(getRiskFactor(),doEval(evaluateData.getInputFeatures(),historyData.ge
tLatestInputFeatures()));
        evaluateChain.doChain(evaluateData,historyData,evaluateReport);
    }
    /**
     * 1.计算圆心中点
     * 2.计算两两特征距离
     * 3.对距离进行排序（升序），取2/3处作为评估距离阈值 - threshold
     * 4.计算当前输入的特征距离中心点距离d
     * @param inputFeatures 当前登录的输入特征
     * @param latestInputFeatures 历史的输入特征
     * @return
     */
    public boolean doEval(Double[] inputFeatures, List<Double[]> latestInputFeatures){
        //至少保证历史记录有2条（含）以上
        if(latestInputFeatures==null || latestInputFeatures.size()<2){
            return false;
        }
        // 1.计算圆心中点

        Double[] sumInputs = latestInputFeatures.stream().reduce((v1, v2) -> {
```

```
            Double[] sumInputFeatures = new Double[v1.length];
            for (int i = 0; i < v1.length; i++) {
                if(sumInputFeatures[i]==null){
                    sumInputFeatures[i]=0.0;
                }
                sumInputFeatures[i] += v1[i] + v2[i];
            }
            return sumInputFeatures;
        }).get();
        Double[] centerVector=new Double[sumInputs.length];
        for (int i = 0; i < sumInputs.length; i++) {
            centerVector[i]=sumInputs[i]/latestInputFeatures.size();
        }
        System.out.println("中点:"+ Arrays.stream(centerVector).map(i->i+"").reduce((v1, v2)-
>v1+","+v2).get());
        //2.计算两两特征距离
        List<Double> distances=new ArrayList<>();
        //一定可以拿到 n*(n-1)/2  【n= latestInputFeatures.size()】
        for (int i = 0; i < latestInputFeatures.size() ; i++) {
            Double[] currentFeature=latestInputFeatures.get(i);
            for (int j = i+1; j < latestInputFeatures.size(); j++) {
                Double[] nextFeature=latestInputFeatures.get(j);
                //计算向量距离
                Double distance= calculateDistance(currentFeature,nextFeature);
                distances.add(distance);
            }
        }
        //3.对距离进行排序（升序），取2/3处作为评估距离阈值 - threshold
        distances.sort(new Comparator<Double>() {
            @Override
            public int compare(Double o1, Double o2) {
                if(o1==o2) {
                    return 0;
                }else{
                    return o1>o2?1:-1;
                }
            }
        });
        System.out.println("distances:"+distances);
        Integer n= latestInputFeatures.size();
        int position=(n*(n-1)/2)*2/3;
        Double thresholdDistance = distances.get(position);
        //4.计算当前输入的特征距离中心点距离d
        Double currentDistance = calculateDistance(inputFeatures, centerVector);

 System.out.println("threshold:"+thresholdDistance+","+"currentDistance: "+currentDistance);

        return currentDistance> thresholdDistance;
    }

    /**
     * 利用欧式距离计算两点之间距离

     * @param v1
```

```
     * @param v2
     * @return
     */
    private Double calculateDistance(Double[] v1,Double[] v2){
        Double sum=0.0;
        for (int i = 0; i < v1.length; i++) {
            sum+=(v1[i]-v2[i])*(v1[i]-v2[i]);
        }
        return Math.sqrt(sum);
    }
}
```

相关数学模型：欧几里得度量--->欧式距离

登录位移速度评估

- 需要评估的数据：本次登录的地理位置以及时间
- 历史数据：上一次成功登录的地址位置以及时间

有地理位置（经纬度）可以计算球面距离；有两次时间可以计算时间差；球面距离/时间差=速度

```
import com.baizhi.eveluate.entity.*;
import com.baizhi.eveluate.evaluate.Evaluate;
import com.baizhi.eveluate.util.EvaluateChain;

import static java.lang.Math.*;

/**
 * 登录位移速度评估
 */
public class SpeedEvaluate extends Evaluate {
    //速度阈值
    private Double thresholdSpeed;
    //地球半径
    private static final Double EARTH_RADIUS=6371.393;//千米
    public SpeedEvaluate(Double thresholdSpeed) {
        super(RiskFactor.SPEED);
        this.thresholdSpeed=thresholdSpeed;
    }

    @Override
    public void eval(EvaluateData evaluateData, HistoryData historyData, EvaluateReport
evaluateReport, EvaluateChain evaluateChain) {

 evaluateReport.signReport(getRiskFactor(),doEval(evaluateData.getEvaluateTime(),evaluateData.ge
tGeoPoint(),historyData.getLastLoginTime(),historyData.getLastLoginGeoPoint()));
        evaluateChain.doChain(evaluateData,historyData,evaluateReport);
    }

    /**
     *  判断用户速度是否正常
     * @param evaluateTime 当前登录时间
     * @param currentGeoPoint 当前登录经纬度
```

```
     * @param lastLoginTime 历史最近一次登录时间
     * @param lastLoginGeoPoint 历史最近一次登录经纬度
     * @return
     */
    public boolean doEval(long evaluateTime, GeoPoint currentGeoPoint, long lastLoginTime,
GeoPoint lastLoginGeoPoint){
        //如果没有历史经纬度直接返回false
        if(lastLoginGeoPoint==null){
            return false;
        }
        //计算当前登录时间 和历史登录时间  间隔了多长时间 转换为小时
        double diffTime=(evaluateTime-lastLoginTime)*1.0/(3600*1000);
        //计算两次登录之间的距离
        Double distance = calculateDistance(currentGeoPoint, lastLoginGeoPoint);
        //求出速度与阈值相比较
        double speed=distance/diffTime;
        return speed> thresholdSpeed;
    }

    /**
     * 根据经纬度 求算出两点之间在地球上的距离
     * @param point1
     * @param point2
     * @return
     */
    private Double calculateDistance(GeoPoint point1,GeoPoint point2){
        Double wA=toRadians(point1.getLatitude());//将角度转换为弧度
        Double jA=toRadians(point1.getLongitude());
        Double wB=toRadians(point2.getLatitude());
        Double jB=toRadians(point2.getLongitude());
        return EARTH_RADIUS * acos(cos(wA)*cos(wB)*cos(jB-jA)+sin(wA)*sin(wB));
    }
}
```

相关数学模型：球面距离

更新历史数据

- 接口

```
public interface Updater {
    public void update(LoginSuccessData loginSuccessData, HistoryData historyData,
UpdaterChain updaterChain);
}
```

- 责任链

```
/**

 * 历史记录责任链类
```

```
 */
public class UpdaterChain {
    private int position=0;
    private List<Updater> updaters;

    public UpdaterChain(List<Updater> updaters) {
        this.updaters = updaters;
    }

    public void doChain(LoginSuccessData loginSuccessData, HistoryData historyData){
        if(position < updaters.size()){
            //获取一个责任
            Updater updater = updaters.get(position);
            position +=1;
            updater.update(loginSuccessData,historyData,this);
        }
    }
}
```

登录地区

```
public class CitiesUpdates implements Updater {
    @Override
    public void update(LoginSuccessData loginSuccessData, HistoryData historyData, UpdaterChain
updaterChain) {
        doUpdate(loginSuccessData,historyData);

        //驱动下一个更新因素
        updaterChain.doChain(loginSuccessData,historyData);
    }

    /**
     * 保留所有用户登录过的城市
     * @param loginSuccessData
     * @param historyData
     */
    private void doUpdate(LoginSuccessData loginSuccessData, HistoryData historyData){
        String cityName = loginSuccessData.getCityName();
        Set<String> historyCities = historyData.getHistoryCities();
        if(historyCities==null){
            historyCities=new HashSet<String>();
        }
        historyCities.add(cityName);
        historyData.setHistoryCities(historyCities);
    }
}
```

使用设备

```
public class DeviceUpdates implements Updater {
    private Integer deviceCount=3;
```

```
    public DeviceUpdates(Integer deviceCount) {
        this.deviceCount = deviceCount;
    }

    @Override
    public void update(LoginSuccessData loginSuccessData, HistoryData historyData, UpdaterChain
updaterChain) {
        doUpdate(loginSuccessData,historyData);
        updaterChain.doChain(loginSuccessData,historyData);
    }

    /**
     * 保留所有用户最近deviceCount个设备信息
     * @param loginSuccessData
     * @param historyData
     */
    private void doUpdate(LoginSuccessData loginSuccessData, HistoryData historyData){
        String deviceInformation = loginSuccessData.getDeviceInformation();
        List<String> deviceInformations = historyData.getHistoryDeviceInformations();
        if(deviceInformations==null){
            deviceInformations=new ArrayList<String>();
        }
        if(!deviceInformations.contains(deviceInformation)){
            deviceInformations.add(deviceInformation);
            //判断一下集合大小是否达到阈值
            if(deviceInformations.size()>deviceCount){
                //如果使用过的设备总数比规定的存储的设备数多，就删除第一个
                deviceInformations.remove(0);
            }
        }
        historyData.setHistoryDeviceInformations(deviceInformations);
    }
}
```

登录习惯

```
public class TimeSlotUpdater implements Updater {
    @Override
    public void update(LoginSuccessData loginSuccessData, HistoryData historyData, UpdaterChain
updaterChain) {
        doUpdate(loginSuccessData,historyData);
        updaterChain.doChain(loginSuccessData,historyData);
    }
    public void doUpdate(LoginSuccessData loginSuccessData, HistoryData historyData) {
        //获取登录成功的时间
        long loginTime = loginSuccessData.getEvaluateTime();
        //运用工具类计算出用户登录的时间为星期几
        String[] WEEKS={"星期日","星期一","星期二","星期三","星期四","星期五","星期六"};
        Calendar calendar = Calendar.getInstance();
        calendar.setTimeInMillis(loginTime);

        //计算出用户登录的时间小时数 转换为二位数字  01 02 ... 24
```

```
        String dayOfWeek = WEEKS[calendar.get(Calendar.DAY_OF_WEEK) - 1];
        DecimalFormat decimalFormat=new DecimalFormat("00");
        String hourOfDay=  decimalFormat.format(calendar.get(Calendar.HOUR_OF_DAY));//01 02 ...
24
        //判断是否有过记录;map的key是星期几，value是map（key时间段，value登录次数）
        Map<String, Map<String, Integer>> historyLoginTimeSlot =
historyData.getHistoryLoginTimeSlot();
        if(historyLoginTimeSlot==null){
            historyLoginTimeSlot=new HashMap<>();
        }
        //更新用户的登录习惯
        if(!historyLoginTimeSlot.containsKey(dayOfWeek)){
            //本次成功登录的星期在历史数据中不存在
            //比如今天成功登录啦，今天是星期五，之前从来没有在星期五登录过


            //key是时间段，value是登录次数
            HashMap<String, Integer> timeSlot = new HashMap<String, Integer>();
            timeSlot.put(hourOfDay,1);
            historyLoginTimeSlot.put(dayOfWeek,timeSlot);
        }else{//包含dayOfWeek
            //比如今天成功登录啦，今天是星期五；之前在星期五登录过

            //key是时间段，value是登录次数
            Map<String, Integer> timeSlot = historyLoginTimeSlot.get(dayOfWeek);
            Integer count=0;
            if(timeSlot.containsKey(hourOfDay)){//含有时段
                count=timeSlot.get(hourOfDay);
            }
            timeSlot.put(hourOfDay,count+1);

        }
        historyData.setHistoryLoginTimeSlot(historyLoginTimeSlot);
    }
}
```

登录密码

```
public class PasswordsUpdates implements Updater {
    @Override
    public void update(LoginSuccessData loginSuccessData, HistoryData historyData, UpdaterChain
updaterChain) {
        doUpdate(loginSuccessData,historyData);
        updaterChain.doChain(loginSuccessData,historyData);
    }
    /**
     * 保留所有用户正常登录过的密码
     * @param loginSuccessData
     * @param historyData
     */
    private void doUpdate(LoginSuccessData loginSuccessData, HistoryData historyData){

        String ordernessPassword = loginSuccessData.getOrdernessPassword();
```

```
        Set<String> historyOrdernessPasswords = historyData.getHistorypasswords();
        if(historyOrdernessPasswords==null){
            historyOrdernessPasswords=new HashSet<String>();
        }
        historyOrdernessPasswords.add(ordernessPassword);

        historyData.setHistorypasswords(historyOrdernessPasswords);
    }
}
```

输入特征

```
public class LatestInputFeatures implements Updater {
    private Integer numCount=10;//阈值：只存储最近的10条数据

    public void setNumCount(Integer numCount) {
        this.numCount = numCount;
    }
    @Override
    public void update(LoginSuccessData loginSuccessData, HistoryData historyData, UpdaterChain
updaterChain) {
        doUpdate(loginSuccessData,historyData);
        updaterChain.doChain(loginSuccessData,historyData);
    }

    /**
     * 设置一个阈值 保证历史的记录中只存在numCount条记录
     * @param loginSuccessData
     * @param historyData
     */
    private void doUpdate(LoginSuccessData loginSuccessData, HistoryData historyData){
        Double[] inputFeatures = loginSuccessData.getInputFeatures();
        List<Double[]> latestInputFeatures = historyData.getLatestInputFeatures();
        if(latestInputFeatures==null){
            latestInputFeatures=new ArrayList<>();
        }
        latestInputFeatures.add(inputFeatures);
        if(latestInputFeatures.size()>numCount){
            latestInputFeatures.remove(0);
        }

        historyData.setLatestInputFeatures(latestInputFeatures);
    }
}
```

登录时间

历史数据中存储的是最近一次登录成功的时间；这个时间用来进行位移速度计算

```
/**
 * 记录最近一次历史登录时间
 */
public class LastLoginTime implements Updater {
    @Override
    public void update(LoginSuccessData loginSuccessData, HistoryData historyData, UpdaterChain
updaterChain) {
        doUpdate(loginSuccessData,historyData);
        updaterChain.doChain(loginSuccessData,historyData);
    }
    public void doUpdate(LoginSuccessData loginSuccessData, HistoryData historyData){
        historyData.setLastLoginTime(loginSuccessData.getEvaluateTime());
    }
}
```

登录地理位置

```
/**
 * 记录最近一次历史登录经纬度
 */
public class LastLoginGeoPoint implements Updater {
    @Override
    public void update(LoginSuccessData loginSuccessData, HistoryData historyData, UpdaterChain
updaterChain) {
        doUpdate(loginSuccessData,historyData);
        updaterChain.doChain(loginSuccessData,historyData);
    }
    public void doUpdate(LoginSuccessData loginSuccessData, HistoryData historyData) {
        historyData.setLastLoginGeoPoint(loginSuccessData.getGeoPoint());
    }
}
```

测试

```
//更新
//1.准备登录成功的日志信息===三条数据
//2.把数据形成HistoryData
//2.1循环successLog,把每一条日志转换成一个SuccessData对象;
// 通过更新链，把这个successData对象中的数据放入到HistoryData里面
```

```
//1.准备登录成功的日志信息===三条数据
//2.把数据形成HistoryData
//2.1循环successLog,把每一条日志转换成一个SuccessData对象;
// 通过更新链，把这个successData对象中的数据放入到HistoryData里面
```

- 更新历史数据：最后可以把历史数据显示出来
  - 准备一些登录成功的日志信息

```
INFO 2020-06-19 10:23:16 WebApplication success [zhangsan]
119e76e86bea42e9a098c2461a6b9314 "123456"
Zhengzhou "113.65,34.76"
[1500,2800,3100] "Mozilla/5.0 (Windows NT 6.1; Win64; x64) AppleWebKit/537.36 (KHTML,
like Gecko) Chrome/78.0.3904.108 Safari/537.36"

INFO 2020-06-19 14:23:16 WebApplication success [zhangsan]
119e76e86bea42e9a01ec2461a6b9314 "123457"
Zhengzhou "113.65,34.76"
[1000,8000,5300] "Mozilla/5.0 (Windows NT 6.1; Win64; x64) AppleWebKit/537.36 (KHTML,
like Gecko) Chrome/78.0.3904.108 Safari/537.36"

INFO 2020-06-19 14:23:16 WebApplication success [zhangsan]
119e76e86bef42e9a01fc2461a6b9314 "1234a7"
Zhengzhou "113.65,34.76"
[10000,80000,53000] "Mozilla/6.0 (Windows NT 6.1; Win64; x64) AppleWebKit/537.36 (KHTML,
like Gecko) Chrome/78.0.3904.108 Safari/537.36"
```

  - 通过工具类，把每一条日志信息解析成LoninSuccessData对象

```
/**
     * 把所有的数据封装形成一个实体对象-->登录成功数据
     * @param data
     * @return
     */
    public static LoginSuccessData getLoginSuccessData(String data){
        if(isSuccessData(data)){
            //获取时间
            long evaluateTime= (long)
getData(data,SUCCESS_PATTERN,EvaluateDataType.EVALUATE_DATE);
            String applicationName= (String)
getData(data,SUCCESS_PATTERN,EvaluateDataType.APPLICATION_NAME);
            String userIdentify = (String)
getData(data,SUCCESS_PATTERN,EvaluateDataType.USER_IDENTIFY);
            String loginSequence = (String)
getData(data,SUCCESS_PATTERN,EvaluateDataType.LOGIN_SEQUENCE);
            String ordernessPassword = (String)
getData(data,SUCCESS_PATTERN,EvaluateDataType.ORDERNESS_PASSWORD);
            String cityName = (String)
getData(data,SUCCESS_PATTERN,EvaluateDataType.AREA);

            GeoPoint geoPoint = (GeoPoint)
```

```
getData(data,SUCCESS_PATTERN,EvaluateDataType.GEO_POINT);
            double[] inputFeatures = (double[])
getData(data,SUCCESS_PATTERN,EvaluateDataType.INPUT_FEATURES);
            String deviceInformation = (String)
getData(data,SUCCESS_PATTERN,EvaluateDataType.DEVICE_INFORMATION);
            return new
LoginSuccessData(evaluateTime,applicationName,userIdentify,loginSequence,ordernessPassw
ord,cityName,geoPoint,inputFeatures,deviceInformation);
        }
        throw new RuntimeException("数据格式有误");
    }
```

> 相关的方法，参考EvaluateUtil.java

- 准备需要更改的updater对象

```
private List<Updater> evaluates=new ArrayList<Updater>();
updaters.add(new CitiesUpdates());
updaters.add(new DeviceUpdates(3));
updaters.add(new LatestInputFeatures());
updaters.add(new LastLoginGeoPoint());
updaters.add(new LastLoginTime());
updaters.add(new PasswordsUpdates());
updaters.add(new TimeSlotUpdater());
```

- 创建更新链

```
HistoryData historyData = new HistoryData();
for (int i = 0; i < logDatas.length; i++) {
    String data = logDatas[i];
    UpdaterChain updaterChain = new UpdaterChain(updaters);
    LoginSuccessData loginSuccessData= EvaluateUtil.parseSuccessData(data);
    updaterChain.doChain(loginSuccessData,historyData);
}
```

- 测试评估因子，需要历史数据，最后生成一个评估报告
  - 需要测试数据

```
INFO 2020-06-19 14:23:16 WebApplication evaluate [zhangsan]
119e76e86bef42e9a01fc2461a6b9314 "1234a7"
Zhengzhou "113.65,34.76"
[10000,80000,53000] "Mozilla/6.0 (Windows NT 6.1; Win64; x64) AppleWebKit/537.36 (KHTML,
like Gecko) Chrome/78.0.3904.108 Safari/537.36"
```

  - 需要解析测试数据的工具类方法

```
/**
   * 把所有的数据封装形成一个实体对象-->评估数据
```

```
     * @param data
     * @return
*/
public static EvaluateData getEvaluateData(String data){
    if(isEvaluateData(data)){
        //获取时间
        long evaluateTime= (long)
getData(data,EVALUATE_PATTERN,EvaluateDataType.EVALUATE_DATE);
        String applicationName= (String)
getData(data,EVALUATE_PATTERN,EvaluateDataType.APPLICATION_NAME);
        String userIdentify = (String)
getData(data,EVALUATE_PATTERN,EvaluateDataType.USER_IDENTIFY);
        String loginSequence = (String)
getData(data,EVALUATE_PATTERN,EvaluateDataType.LOGIN_SEQUENCE);
        String ordernessPassword = (String)
getData(data,EVALUATE_PATTERN,EvaluateDataType.ORDERNESS_PASSWORD);
        String cityName = (String)
getData(data,EVALUATE_PATTERN,EvaluateDataType.AREA);
        GeoPoint geoPoint = (GeoPoint)
getData(data,EVALUATE_PATTERN,EvaluateDataType.GEO_POINT);
        double[] inputFeatures = (double[])
getData(data,EVALUATE_PATTERN,EvaluateDataType.INPUT_FEATURES);
        String deviceInformation = (String)
getData(data,EVALUATE_PATTERN,EvaluateDataType.DEVICE_INFORMATION);
        return new
EvaluateData(evaluateTime,applicationName,userIdentify,loginSequence,ordernessPassword,
cityName,geoPoint,inputFeatures,deviceInformation);
    }

    throw new RuntimeException("数据格式有误");
}
```

> 相关的方法，参考EvaluateUtil.java

- 需要准备评估因子

```
private List<Evaluate> evaluates;
evaluates=new ArrayList<>();
evaluates.add(new AreaEvaluate());
evaluates.add(new DeviceEvaluate());
evaluates.add(new InputfeatureEvaluate());
evaluates.add(new SimilarityEvaluate(0.9));
evaluates.add(new SpeedEvaluate(750.0));
evaluates.add(new TimeSlotEvaluate(1));
evaluates.add(new TotalEvaluate(2));
```

- 需要准备评估报告

```
EvaluateReport evaluateReport = new EvaluateReport(
    evaluateData.getApplicationName(),
    evaluateData.getUserIdentify(),
    evaluateData.getLoginSequence(),
    evaluateData.getEvaluateTime(),
    evaluateData.getCityName(),
    evaluateData.getGeoPoint());
```

- 评估链

```
EvaluateChain evaluateChain = new EvaluateChain(evaluates);
evaluateChain.doChain(evaluateData,historyData,evaluateReport);
```

- 可以把评估报告显示出来

- 完整代码（包括更新历史数据以及评估报告）

```
public class UpdaterAndEvaluateTest {

    //登录成功数据
    String[] successDatas={
            "INFO 2020-06-19 10:23:16 WebApplication success [zhangsan]
119e76e86bea42e9a098c2461a6b9314 \"123456\"\n" +
                    "bj \"113.65,34.76\"\n" +
                    "[1500,2800,3100] \"Mozilla/5.0 (Windows NT 6.1; Win64; x64)
AppleWebKit/537.36 (KHTML, like Gecko) Chrome/78.0.3904.108 Safari/537.36\""
            ,
            "INFO 2020-06-19 14:23:16 WebApplication success [zhangsan]
119e76e86bea42e9a01ec2461a6b9314 \"123457\"\n" +
                    "Zhengzhou \"113.65,34.76\"\n" +
                    "[1000,1000,3300] \"Mozilla/5.0 (Windows NT 6.1; Win64; x64)
AppleWebKit/537.36 (KHTML, like Gecko) Chrome/78.0.3904.108 Safari/537.36\""
            ,
            "INFO 2020-06-19 14:23:15 WebApplication success [zhangsan]
119e76e86bef42e9a01fc2461a6b9314 \"1234a7\"\n" +
                    "zz \"113.65,34.76\"\n" +
                    "[1500,2800,3100] \"Mozilla/6.0 (Windows NT 6.1; Win64; x64)
AppleWebKit/537.36 (KHTML, like Gecko) Chrome/78.0.3904.108 Safari/537.36\""
    };

    //需要评估的数据
    String evaluateInputData="INFO 2020-06-19 14:23:16 WebApplication evaluate [zhangsan]
119e76e86bef42e9a01fc2461a6b9314 \"EFFFFFFFS\"\n" +
            "wlmq \"87.68,43.77\"\n" +
            "[10000,80000,53000] \"Mozilla/7.0 (Windows NT 6.1; Win64; x64) AppleWebKit/537.36
(KHTML, like Gecko) Chrome/78.0.3904.108 Safari/537.36\"";

    private List<Updater> updaters;
    private List<Evaluate> evaluates;


    @Before//被这个注解标记的方法，会在@Test执行之前执行
```

```
    public void beforeeeee(){
        System.out.println("=========================");
        updaters=new ArrayList<Updater>();
        updaters.add(new CitiesUpdates());
        updaters.add(new DeviceUpdates(3));
        updaters.add(new LatestInputFeatures());
        updaters.add(new LastLoginGeoPoint());
        updaters.add(new LastLoginTime());
        updaters.add(new PasswordsUpdates());
        updaters.add(new TimeSlotUpdater());

        //评估因子
        evaluates=new ArrayList<>();
        evaluates.add(new AreaEvaluate());
        evaluates.add(new DeviceEvaluate());
        evaluates.add(new InputfeatureEvaluate());
        evaluates.add(new SimilarityEvaluate(0.9));
        evaluates.add(new SpeedEvaluate(750.0));
        evaluates.add(new TimeSlotEvaluate(1));
        evaluates.add(new TotalEvaluate(2));
    }

    /**
     * 在Junit测试中还有一些注解，需要知道
     * 1. @After:在@Test标记的方法执行之后执行
     * 2. @BeforeClass:在类中的static方法执行之前
     * 3. @AfterClass:在类中的static方法执行之后
     */


    //历史数据
    HistoryData historyData=new HistoryData();

//    @Test
    @Before
    public void test(){

        System.out.println("88888888888888888888888888");
        //更新链
        for (String successData : successDatas) {
            Integer count = historyData.getCurrentDayLoginCount();
            int c = 0;
            if(count!=null){
                c=count;
            }
            historyData.setCurrentDayLoginCount(c+1);
            UpdaterChain updaterChain = new UpdaterChain(updaters);
            LoginSuccessData loginSuccessData= EvaluateUtil.getLoginSuccessData(successData);
            updaterChain.doChain(loginSuccessData,historyData);
        }

        //System.out.println(historyData);

    }
```

```
    @Test
    public void testEvaluate(){
        System.out.println("77777777777777777777777777");

        EvaluateData evaluateData = EvaluateUtil.getEvaluateData(evaluateInputData);

        EvaluateReport evaluateReport = new EvaluateReport(
                evaluateData.getApplicationName(),
                evaluateData.getUserIdentify(),
                evaluateData.getLoginSequence(),
                evaluateData.getEvaluateTime(),
                evaluateData.getCityName(),
                evaluateData.getGeoPoint());

        EvaluateChain evaluateChain = new EvaluateChain(evaluates);
        evaluateChain.doChain(evaluateData,historyData,evaluateReport);

        System.out.println(evaluateReport);

    }
}
```

通过欧几里得距离（欧式距离）完成输入特征的评估；

通过余弦相似度（性）完成乱序密码相似度的评估；

通过球面距离以及时间计算位移速度从而完成位移速度的评估