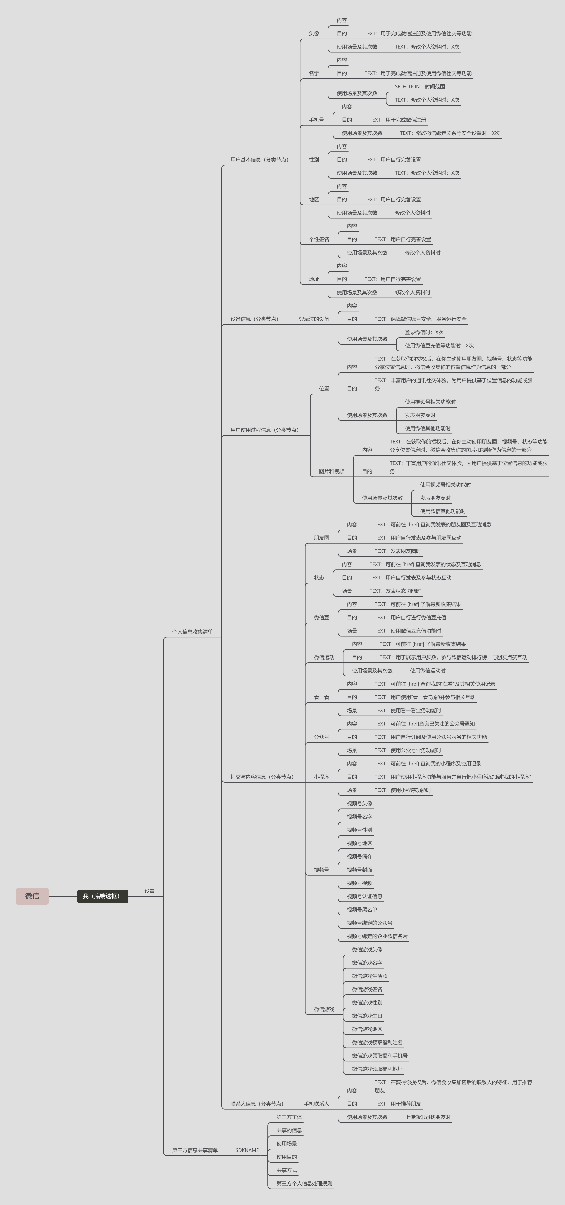
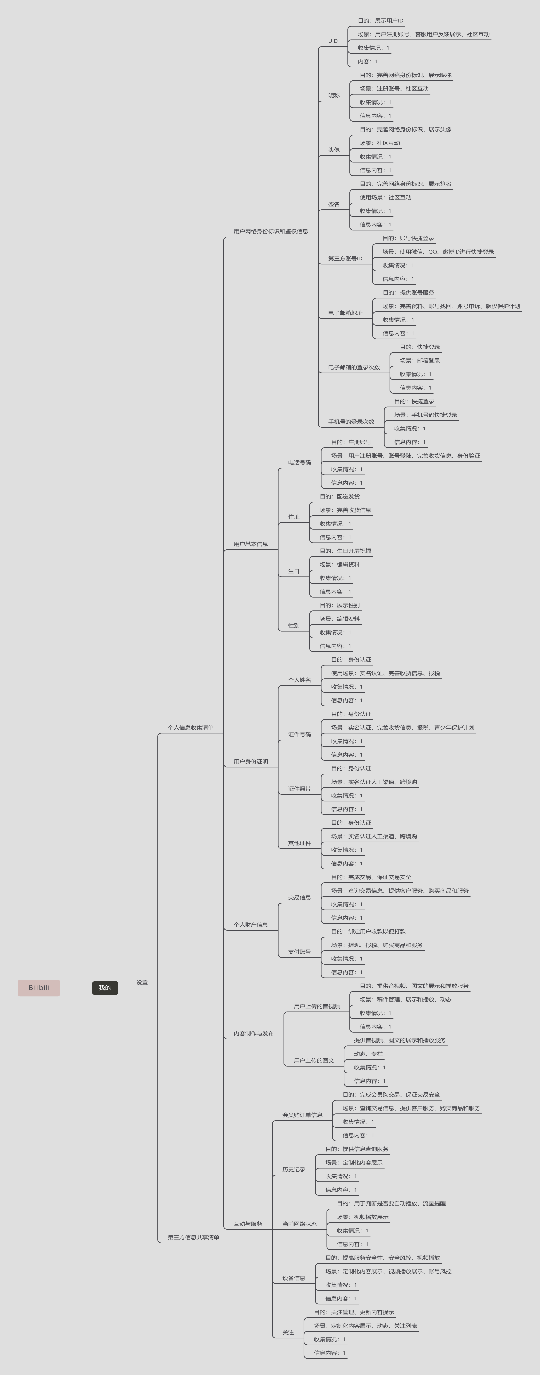
# 2023-3-30

## 需求分析

## 逻辑结构

### UML图



### 解释

#### 实体解释

* 软件
* 个人信息收集清单

各个软件的个人信息手机清单Button内，体系复杂且多样，并且带有多种结构，如：Button跳转+表结构展示 / Label展示分类+Button跳转+label显示内容 / etc.

由于结构多样且跳转复杂，且有少量app带有深层路径，因此采用可扩展性强的树状结构存储，并且，将跳转的button和展示分类用的label设为不带内容的中间节点，将能够跳转后显示内容的button/表格型数据设为带内容的叶节点，以此存储每个app个人信息收集清单的数据。

* 第三方信息共享清单

每个app的第三方信息共享清单均是由跳转button进入页面，表格呈现清单内容，无层次结构，因此采用常规存储方式。

第三方信息共享类型每个app设定不同，大体分为SDK、关联商、合作商。

#### 字段解释

- Software:

- **Software\_ID**：作为唯一标识软件的主键

- Software\_Name：软件名称

- Company\_Name：公司名称

- Software\_Version：软件版本号

- Software\_Trace：软件到达双清单的路径

- PersonalList:

- **PLNode\_ID**：唯一标识个人信息清单节点的主键

- Software\_ID：标识软件的外键

- PLNode\_Name：节点名称

- PLNode\_Type：节点类型，有2种（无内容节点0，有内容叶节点1）

- PLNode\_Relation

- **Parent\_ID**：树形结构父节点ID

- **Child\_ID**：树形结构子节点ID

- PLNode\_Attributions（有内容叶节点的具体信息）

- **PLNode\_ID**：唯一标识个人信息清单节点的主键

- PLNode\_Name：节点名称

- PLNode\_Purpose：收集目的

- PlNode\_UseCase：收集场景

- PLNode\_Collect\_Situation：收集情况是否展示

- PLNode\_Content：收集内容是否展示

- ThirdList

- **TLNode\_ID**：唯一标识第三方共享清单的主键

- Software\_ID：标识软件的外键

- TLNode\_Type：节点类型（SDK，合作商，关联商，etc.）

- TLNode\_OS：（iOS，Android）

- TLNode\_Name：第三方共享信息工具名称

- TLNode\_Company：第三方公司名称

- TLNode\_InformationContent：共享信息内容

- TLNode\_Purpose：收集目的

- TLNode\_UseCase：收集场景

- TLNode\_Way：收集方式

- TLNode\_Detailed：隐私政策细节

## 数据库实施

### 可拓展性探求

针对53款app的双清单人工考虑是否满足数据库的设计模式。



### 数据填入

对于数据库数据填充，我们采用的方式是用幕布人工填入半结构化文本（兼有思维导图的功能）；导出为opml，使用python脚本解析并填入数据库。

#### 填入效果

Software



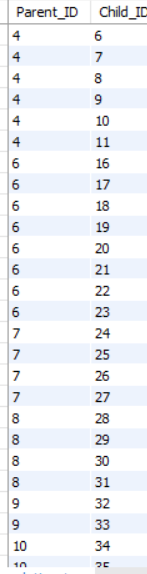
PersonalList



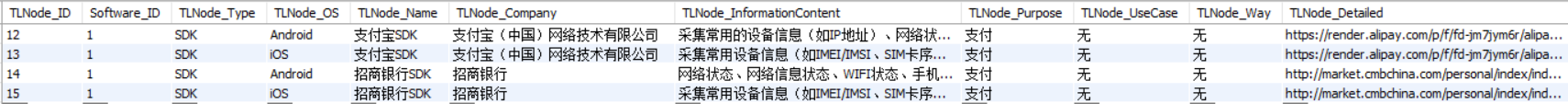
PLNode\_Attributions



PLNode\_Relation



ThirdList



## 附1 数据库

use demo;

create table software (

     Software\_ID int ,

     Software\_Name nvarchar(20),

     Company\_Name nvarchar(60),

     Software\_Version nvarchar(20),

     Software\_Trace nvarchar(40),

     Primary key(Software\_ID)

     );

*-- 0:没有内容的节点 1:有内容的叶节点*

 create table PersonalList (

    PLNode\_ID int,

    Software\_ID int,

    PLNode\_Name nvarchar(20),

    PLNode\_Type tinyint,

    Primary key(PLNode\_ID),

*foreign key*(Software\_ID) *references* software(Software\_ID),

*Check* ( PLNode\_Type in (0, 1) )

 );

create table PLNode\_Relation (

    Parent\_ID int,

    Child\_ID int,

    Primary key(Parent\_ID, Child\_ID),

*foreign key*(Parent\_ID) *references* personallist(PLNode\_ID),

*foreign key*(Child\_ID) *references* personallist(PLNode\_ID)

 );

*-- 0：无 1：有*

create table PLNode\_Attributions (

    PLNode\_ID int,

    PLNode\_Purpose nvarchar(100),

    PlNode\_UseCase nvarchar(100),

    PLNode\_Collect\_Situation tinyint,

    PLNode\_Content tinyint,

    Primary key(PLNode\_ID),

*foreign key*(PLNode\_ID) *references* personallist(PLNode\_ID),

*check* ( PLNode\_Collect\_Situation in (0, 1)),

*check* ( PLNode\_Content in (0, 1) )

 );

create table ThirdList (

    TLNode\_ID int,

    Software\_ID int,

    TLNode\_Type nvarchar(20),

    TLNode\_OS nvarchar(20),

    TLNode\_Name nvarchar(100),

    TLNode\_Company nvarchar(100),

    TLNode\_InformationContent nvarchar(300),

    TLNode\_Purpose nvarchar(100),

    TLNode\_UseCase nvarchar(100),

    TLNode\_Way nvarchar(100),

    TLNode\_Detailed nvarchar(300),

    primary key(TLNode\_ID),

*foreign key*(Software\_ID) *references* software(Software\_ID)

 );

## 附2 脚本

**Main.py**

import xml.dom.minidom as XmlDocument  
import xml  
from db import DataBase  
  
  
def isLeafNodeAttribution(element):  
 if "：" in element.getAttribute("text"):  
 return True  
 else:  
 return False  
  
  
def isLeafNode(PLNode):  
 if PLNode.purpose is not None:  
 return True  
 else:  
 return False  
  
  
class Node:  
 @staticmethod  
 def findNode(lst, name):  
 for node in lst:  
 if node.name == name:  
 return node  
  
 def \_\_init\_\_(self, name, id):  
 self.name = name  
 self.id = id  
 self.purpose = None  
 self.usecase = None  
 self.collectSituation = None  
 self.content = None  
  
 def setLeafAttributions(self, purpose, usecase, collectSituation, content):  
 self.purpose = purpose  
 self.usecase = usecase  
 self.collectSituation = collectSituation  
 self.content = content  
  
 def setLeafAttribution(self, text):  
 temp = text.split("：")  
 if "目的" in temp[0]:  
 self.purpose = temp[1]  
 elif "场景" in temp[0]:  
 self.usecase = temp[1]  
 elif "情况" in temp[0]:  
 self.collectSituation = temp[1]  
 elif "内容" in temp[0]:  
 self.content = temp[1]  
  
  
class ShareNode:  
 def \_\_init\_\_(self):  
 self.id = 0  
 self.name = 0  
 self.type = 0  
 self.os = 0  
 self.com = 0  
 self.content = 0  
 self.purpose = 0  
 self.scene = 0  
 self.collect = 0  
 self.url = 0  
  
 def setInfo(self, infoLst):  
 self.id = infoLst[0]  
 self.name = infoLst[1]  
 self.type = infoLst[2]  
 self.os = infoLst[3]  
 self.com = infoLst[4]  
 self.content = infoLst[5]  
 self.purpose = infoLst[6]  
 self.scene = infoLst[7]  
 self.collect = infoLst[8]  
 self.url = infoLst[9]  
  
 def \_\_str\_\_(self) -> str:  
 return f"""  
 编号：{self.id}  
 名字：{self.name}  
 类型：{self.type}  
 操作系统：{self.os}  
 公司：{self.com}  
 信息内容：{self.content}  
 收集目的：{self.purpose}  
 收集场景：{self.scene}  
 收集方式:{self.collect}  
 隐私政策：{self.url}  
 """  
  
  
def parseOpml(filepath):  
 *"""解析opml  
 返回[[PLNode],[TLNode],[NodeRelation]]"""* doc = XmlDocument.parse(filepath)  
  
 # 读取NodeID  
 with open("NodeID.txt", "r") as fp:  
 nodeId = int(fp.readline())  
  
 print(nodeId)  
  
 # 获取body元素的数据  
 node = doc.getElementsByTagName('body')[0]  
 nodeLst = []  
 shareNodeLst = []  
 relationLst = []  
 personNodeId = -1  
 shareNodeId = -1  
  
 # 获取头部outline数据，实例化进入列表  
 elementLst = [i for i in node.childNodes if isinstance(i, xml.dom.minidom.Element)]  
 print(elementLst)  
 PLNode = Node(elementLst[0].getAttribute("text"), nodeId)  
 nodeLst.append(PLNode)  
 nodeId += 1  
  
 while len(elementLst) != 0:  
 newElementLst = []  
 for parent in elementLst:  
  
 # 获取父节点信息  
 parent\_name = parent.getAttribute("text")  
 parentNode = Node.findNode(nodeLst, name=parent\_name)  
 parent\_id = parentNode.id  
 if parent\_id == shareNodeId:  
 for child in parent.childNodes:  
 if not isinstance(child, xml.dom.minidom.Element):  
 continue  
 shareNodeInfo = [nodeId, child.getAttribute("text")]  
 relationLst.append((parent\_id, nodeId))  
 nodeId += 1  
 for attribute in child.childNodes:  
 if not isinstance(attribute, xml.dom.minidom.Element):  
 continue  
 shareNodeInfo.append(attribute.getAttribute("text").split("：")[1])  
 s = ShareNode()  
 s.setInfo(shareNodeInfo)  
 shareNodeLst.append(s)  
 else:  
 for child in parent.childNodes:  
 if isinstance(child, xml.dom.minidom.Element):  
 child\_name = child.getAttribute("text")  
  
 # 判断节点是否为属性内容  
 if isLeafNodeAttribution(child):  
 parentNode.setLeafAttribution(child\_name)  
 else:  
 if child\_name == "第三方信息共享清单":  
 shareNodeId = nodeId  
 elif child\_name == "个人信息收集清单":  
 personNodeId = nodeId  
 newElementLst.append(child)  
 PLNode = Node(child\_name, nodeId)  
 nodeLst.append(PLNode)  
 relationLst.append((parent\_id, nodeId))  
 nodeId += 1  
 elementLst = list(newElementLst)  
  
 # 写入NodeID  
 with open("NodeID.txt", "w") as fp:  
 fp.write(str(nodeId + 1))  
  
 for PLNode in nodeLst:  
 print(f"ID : {PLNode.id}, NAME: {PLNode.name}, Purpose:{PLNode.purpose}")  
 for i in shareNodeLst:  
 print(i)  
 print(relationLst)  
  
 return [nodeLst, shareNodeLst, relationLst]  
  
  
def InsertAllNodes(filepath, db):  
 # 解析Opml  
 reslst = parseOpml(filepath)  
 PLNodeLst = reslst[0]  
 TLNodeLst = reslst[1]  
 PLNodeRelation = reslst[2]  
  
 softwareNode = PLNodeLst[0]  
 softwareID = softwareNode.id  
 del PLNodeLst[0]  
  
 # 清理节点  
 traceNode = [PLNodeLst[0], PLNodeLst[1]]  
 PLNodeLst.remove(traceNode[0])  
 PLNodeLst.remove(traceNode[1])  
  
 values = []  
 for index in range(len(PLNodeRelation)):  
 for Node in traceNode:  
 if Node.id in PLNodeRelation[index]:  
 values.append(PLNodeRelation[index])  
  
 values = set(values)  
  
 for value in values:  
 PLNodeRelation.remove(value)  
  
 trace = f"{traceNode[0].name}-{traceNode[1].name}"  
  
 print(PLNodeRelation)  
 print(trace)  
  
 # 清理第三方节点  
 id = None  
 for PLNode in PLNodeLst:  
 if PLNode.name == "第三方信息共享清单":  
 id = PLNode.id  
 PLNodeLst.remove(PLNode)  
 print(id)  
 values = []  
 for relation in PLNodeRelation:  
 if id == relation[0]:  
 values.append(relation)  
  
 print(values)  
 for value in values:  
 PLNodeRelation.remove(value)  
  
 # Software 入库  
 db.insert\_software(softwareNode, trace)  
  
 # PLNode 入库  
 for PLNode in PLNodeLst:  
 if isLeafNode(PLNode):  
 db.insert\_PLLeafNode(PLNode, softwareID)  
 else:  
 db.insert\_PLNode(PLNode, softwareID)  
  
 # TLNode 入库  
 for TLNode in TLNodeLst:  
 db.insert\_TLNode(TLNode, softwareID)  
  
 # PLNodeRelation 入库  
 for relation in PLNodeRelation:  
 db.insert\_Relation(relation)  
  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 db = DataBase()  
 InsertAllNodes("bilibili.opml", db)

**db.py**

import pymysql  
  
  
class DataBase:  
 def \_\_init\_\_(self):  
 self.conn = pymysql.connect(host='101.43.94.40', user='root', password='2028295zY@', database='demo')  
 self.cursor = self.conn.cursor()  
  
 def query\_strip(self):  
 *"""将查询结果转换成二维列表"""* data = self.cursor.fetchall()  
 return [[j.strip() if type(j) == str else j for j in i] for i in data]  
  
 def insert\_software(self, softwareNode, trace):  
 *"""  
 :param softwareNode:  
 :param trace:  
 :功能：插入软件  
 """* sqlstr = f"insert into software(software\_ID, software\_Name, software\_trace) values" \  
 f"({softwareNode.id},'{softwareNode.name}','{trace}')"  
 self.cursor.execute(sqlstr)  
 self.conn.commit()  
  
 def insert\_PLLeafNode(self, PLNode, softwareID):  
 *"""  
  
 :param PLNode:  
 :param softwareID:  
 :return:  
 """* sqlstr = f"insert into PersonalList(PLNode\_ID, Software\_ID, PLNode\_Name, PLNode\_Type)" \  
 f"values ({PLNode.id},{softwareID},'{PLNode.name}',1)"  
 self.cursor.execute(sqlstr)  
 sqlstr = f"insert into PLNode\_Attributions(PLNode\_ID, PLNode\_Purpose, PLnode\_UseCase, " \  
 f"PLNode\_Collect\_Situation, PLNode\_Content)" \  
 f"values ({PLNode.id},'{PLNode.purpose}','{PLNode.usecase}',{PLNode.collectSituation},{PLNode.content})"  
 self.cursor.execute(sqlstr)  
 self.conn.commit()  
  
 def insert\_PLNode(self, PLNode, softwareID):  
 *"""  
  
 :param PLNode:  
 :param softwareID:  
 :return:  
 """* sqlstr = f"insert into PersonalList(PLNode\_ID, Software\_ID, PLNode\_Name, PLNode\_Type)" \  
 f"values ({PLNode.id},{softwareID},'{PLNode.name}',0)"  
 self.cursor.execute(sqlstr)  
 self.conn.commit()  
  
 def insert\_TLNode(self, TLNode, softwareID):  
 *"""  
  
 :param TLNode:  
 :param softwareID:  
 :return:  
 """* sqlstr = f"insert into ThirdList(TLNode\_ID, software\_ID, TLNode\_Type, TLNode\_OS, TLNode\_Name, TLNode\_Company, " \  
 f"TLNode\_InformationContent, TLNode\_Purpose, TLNode\_UseCase, TLNode\_Way, TLNode\_Detailed)" \  
 f"values({TLNode.id},{softwareID},'{TLNode.type}','{TLNode.os}','{TLNode.name}','{TLNode.com}','{TLNode.content}','{TLNode.purpose}','{TLNode.scene}','{TLNode.collect}','{TLNode.url}') "  
 self.cursor.execute(sqlstr)  
 self.conn.commit()  
  
 def insert\_Relation(self, relation):  
 *"""  
  
 :param relation:  
 :return:  
 """* sqlstr = f"insert into PLNode\_Relation(Parent\_ID, Child\_ID) " \  
 f"values({relation[0]},{relation[1]})"  
 self.cursor.execute(sqlstr)  
 self.conn.commit()