# Day04 购物车 地址管理

**刷新底部购物栏**

1.筛选购物车商品(count > 0）

public List<GoodsInfo> getCartList(){  
 List<GoodsInfo> cartList = new ArrayList<>();  
 for(int j=0;j < mAllTypeGoodsList.size();j++){  
 GoodsInfo goodsInfo = mAllTypeGoodsList.get(j);  
 if(goodsInfo.getCount()>0){  
 //数量大于0  
 cartList.add(goodsInfo);  
 }  
 }  
 return cartList;  
}

2.刷新底部购物栏的数量与价格

public void updateCartUi(List<GoodsInfo> cartList) {  
 int count = 0;  
 float countPrice = 0.0f;  
 for(GoodsInfo goodsInfo :cartList){  
 count += goodsInfo.getCount();  
 countPrice += goodsInfo.getCount() \* goodsInfo.getNewPrice();  
 }  
 if(count >0){  
 mTvSelectNum.setVisibility(VISIBLE);  
  
 }else{  
 mTvSelectNum.setVisibility(GONE);  
 }  
 mTvSelectNum.setText(count + "");  
 mTvCountPrice.setText(PriceFormater.format(countPrice));  
  
 //处理配送（当价格超出起送价格，显示下单按钮）  
 if(countPrice > Float.parseFloat(mSeller.getSendPrice())){  
 //显示下单按钮  
 mTvSubmit.setVisibility(VISIBLE);  
 mTvSendPrice.setVisibility(GONE);  
 }else{  
 mTvSubmit.setVisibility(GONE);  
 mTvSendPrice.setVisibility(VISIBLE);  
 }  
}

**购物车**

1.弹出购物车

public void showOrCloseCart() {  
 if (bottomSheetView == null) {  
 //加载要显示的布局  
 bottomSheetView = LayoutInflater.from(this).inflate(R.layout.cart\_list, (ViewGroup) getWindow().getDecorView(), false);  
 mRvCart = (RecyclerView) bottomSheetView.findViewById(R.id.rvCart);  
 TextView tvClear = (TextView) bottomSheetView.findViewById(R.id.tvClear);  
 mRvCart.setHasFixedSize(true);  
 mRvCart.setLayoutManager(new LinearLayoutManager(this));  
 mCartRvAdapter = new CartRvAdapter(this);  
 mRvCart.setAdapter(mCartRvAdapter);  
 }  
 //判断BottomSheetLayout内容是否显示  
 if (mBottomSheetLayout.isSheetShowing()) {  
 //关闭内容显示  
 mBottomSheetLayout.dismissSheet();  
 } else {  
 //显示BottomSheetLayout里面的内容  
 //要给mCartRvAdapter设置数据  
 GoodsFragment goodsFragment = (GoodsFragment) mFragmentList.get(0);  
 List<GoodsInfo> cartList = goodsFragment.mGoodsFragmentPresenter.getCartList();  
 if (cartList != null && cartList.size() > 0) {  
 mCartRvAdapter.setCartList(cartList);  
 mBottomSheetLayout.showWithSheetView(bottomSheetView);  
 }  
 }  
}

2.购物车加减号按钮

private void doAddOperation() {  
 //1.改变购物车内部（数量、价格）  
 int count = mGoodsInfo.getCount();  
 count++;  
 mGoodsInfo.setCount(count); //已经在此行更改了数据  
 notifyDataSetChanged();  
}

//2.改变购物栏下方（数量和总价）  
((BusinessActivity) mContext).updateCartUi(mGoodsFragment.mGoodsFragmentPresenter.getCartList());  
//3.刷新左侧列表（红点值）  
 processRedDot(isAdd);  
//4.刷新右侧列表（数量）  
 mGoodsFragment.mGoodsAdapter.notifyDataSetChanged();

3.减号按钮bug修复

private void doMinusOperation() {  
 //1.改变购物车内部（数量、价格）  
 int count = mGoodsInfo.getCount();  
 count--;  
  
 if (count == 0) {  
 //当馒头变成0个的时候，移除馒头条目  
 mCartList.remove(mGoodsInfo);  
 //当最后一类商品被移除（购物车为空的时候），关闭购物车  
 if (mCartList.size() == 0) {  
 //关闭购物车  
 mActivity.showOrCloseCart();  
 }  
 }  
 mGoodsInfo.setCount(count); //已经在此行更改了数据  
 notifyDataSetChanged();  
}

4.清空购物车

public void clearCart() {  
 List<GoodsInfo> cartList = getCartList();  
 if(cartList!=null && cartList.size()>0){  
 for(GoodsInfo goodsInfo : cartList){  
 goodsInfo.setCount(0);  
 }  
 }  
}

//分别刷新 购物车内部、左右侧列表、下方购物栏  
mCartRvAdapter.notifyDataSetChanged();  
  
//TODO:左侧红点值需要手动清空（计算出来的)  
List<GoodsTypeInfo> goodsTypeInfoList = goodsFragment.mGoodsTypeRVAdapter.getGoodsTypeInfoList();  
if (goodsTypeInfoList != null && goodsTypeInfoList.size() > 0) {  
 for (GoodsTypeInfo goodsTypeInfo : goodsTypeInfoList) {  
 goodsTypeInfo.setRedCount(0);  
 }  
}  
goodsFragment.mGoodsTypeRVAdapter.notifyDataSetChanged();  
goodsFragment.mGoodsAdapter.notifyDataSetChanged();  
updateCartUi(goodsFragment.mGoodsFragmentPresenter.getCartList());

**购物车内存缓存**

1.点餐缓存javabean

public class CacheSelectedInfo {  
  
 private int sellerId; //田老师红烧肉  
 private int typeId; //粗粮主食  
 private int goodsId; //馒头  
 private int count; //几个馒头

}

2.全局缓存

ArrayList<CacheSelectedInfo> infos = new ArrayList<>();  
  
public int queryCacheSelectedInfoByGoodsId(int goodsId) {  
 int count = 0;  
 for (int i = 0; i < infos.size(); i++) {  
 CacheSelectedInfo info = infos.get(i);  
 if (info.getGoodsId() == goodsId) {  
 count = info.getCount();  
 break;  
 }  
 }  
 return count;  
}  
  
public int queryCacheSelectedInfoByTypeId(int typeId) {  
 int count = 0;  
 for (int i = 0; i < infos.size(); i++) {  
 CacheSelectedInfo info = infos.get(i);  
 if (info.getTypeId() == typeId) {  
 count = count + info.getCount();  
 }  
 }  
 return count;  
}  
  
public int queryCacheSelectedInfoBySellerId(int sellerId) {  
 int count = 0;  
 for (int i = 0; i < infos.size(); i++) {  
 CacheSelectedInfo info = infos.get(i);  
 if (info.getSellerId() == sellerId) {  
 count = count + info.getCount();  
 }  
 }  
 return count;  
}  
  
public void addCacheSelectedInfo(CacheSelectedInfo info) {  
 infos.add(info);  
}  
  
public void clearCacheSelectedInfo(int sellerId) {  
 for (int i = 0; i < infos.size(); i++) {  
 CacheSelectedInfo info = infos.get(i);  
 if (info.getSellerId() == sellerId) {  
 infos.remove(info);  
 }  
 }  
}  
  
public void deleteCacheSelectedInfo(int goodsId) {  
 for (int i = 0; i < infos.size(); i++) {  
 CacheSelectedInfo info = infos.get(i);  
 if (info.getGoodsId() == goodsId) {  
 infos.remove(info);  
 break;  
 }  
 }  
}  
  
public void updateCacheSelectedInfo(int goodsId, int operation) {  
 for (int i = 0; i < infos.size(); i++) {  
 CacheSelectedInfo info = infos.get(i);  
 if (info.getGoodsId() == goodsId) {  
 switch (operation) {  
 case Constant.ADD:  
 info.setCount(info.getCount() + 1);  
 break;  
 case Constant.MINUS:  
 info.setCount(info.getCount() - 1);  
 break;  
 }  
  
 }  
 }  
}

3.逐层读取缓存

int allcount = TakeoutApp.sInstance.queryCacheSelectedInfoBySellerId((int) mSeller.getId());  
boolean hasCacheSelectInfo = false;  
if (allcount > 0) {  
 hasCacheSelectInfo = true;  
}

intent.putExtra("hasCacheSelectInfo", hasCacheSelectInfo);

int aTypeCount = 0;  
if(hasInfo){  
 //这家店有已点的  
 aTypeCount = TakeoutApp.sInstance.queryCacheSelectedInfoByTypeId(goodsTypeInfo.getId());  
 goodsTypeInfo.setRedCount(aTypeCount); //该组缓存数量  
}

if(aTypeCount >0){  
 //当前类别有已点的餐，再查询馒头个数  
 int count = TakeoutApp.sInstance.queryCacheSelectedInfoByGoodsId(goodsInfo.getId()); //  
 goodsInfo.setCount(count); //该商品几个缓存  
}

4.商品页面保存缓存

//5.新增缓存  
 TakeoutApp.sInstance.addCacheSelectedInfo(new CacheSelectedInfo(mGoodsInfo.getSellerId(),mGoodsInfo.getTypeId(),mGoodsInf o.getId(),1));  
}else{  
 //只改变数量，更新缓存  
 TakeoutApp.sInstance.updateCacheSelectedInfo(mGoodsInfo.getId(), Constants.ADD);  
}

//5.删除缓存  
 TakeoutApp.sInstance.deleteCacheSelectedInfo(mGoodsInfo.getId());  
} else {  
 //只改变数量，更新  
 TakeoutApp.sInstance.updateCacheSelectedInfo(mGoodsInfo.getId(), Constant.MINUS);  
}

5.购物车保存缓存

//5.购物车中数量至少为1，所以增加的时候都是更新  
TakeoutApp.sInstance.updateCacheSelectedInfo(mGoodsInfo.getId(), Constants.ADD);

//删除缓存  
 TakeoutApp.sInstance.deleteCacheSelectedInfo(mGoodsInfo.getId());  
}else{  
 TakeoutApp.sInstance.updateCacheSelectedInfo(mGoodsInfo.getId(), Constants.MINUS);  
}

**地址管理**

1.地址JavaBean的ORM处理

@DatabaseTable(tableName = "t\_address")  
public class RecepitAddress implements Serializable{  
  
 @DatabaseField(generatedId = true) //自动增长  
 private int id;  
  
 @DatabaseField(columnName = "name")  
 private String name;  
 @DatabaseField(columnName = "sex")  
 private String sex;  
 @DatabaseField(columnName = "phone")  
 private String phone;  
 @DatabaseField(columnName = "phoneOther")  
 private String phoneOther;  
  
 @DatabaseField(columnName = "address")  
 private String address;  
 @DatabaseField(columnName = "detailAddress")  
 private String detailAddress;  
  
 @DatabaseField(columnName = "label")  
 private String label;  
  
 //多个地址对应用户  
 @DatabaseField(columnName = "userId")  
 private String userId;

}

2.数据库升级（非常重要）

public TakeoutOpenHelper(Context context) {  
 super(context, "takeout36.db", null, 2);  
}  
  
@Override  
public void onCreate(SQLiteDatabase database, ConnectionSource connectionSource) {  
 //创建数据库的时候需要创建表  
 try {  
 TableUtils.createTable(connectionSource, User.class);  
 TableUtils.createTable(connectionSource, RecepitAddress.class); //针对新用户有效  
 } catch (SQLException e) {  
 e.printStackTrace();  
 }  
}  
  
@Override  
public void onUpgrade(SQLiteDatabase database, ConnectionSource connectionSource, int oldVersion, int newVersion) {  
 //更新旧表的字段，增加新的表 adb install -r 覆盖安装  
 try {  
 TableUtils.createTable(connectionSource, RecepitAddress.class); //针对老用户升级  
 } catch (SQLException e) {  
 e.printStackTrace();  
 }  
}

3.封装Dao层

public class AddressDao {  
  
 private Dao<RecepitAddress, Integer> mAddressDao;  
  
 public AddressDao(Context context) {  
 TakeoutOpenHelper takeoutOpenHelper = new TakeoutOpenHelper(context);  
 try {  
 mAddressDao = takeoutOpenHelper.getDao(RecepitAddress.class);  
 } catch (SQLException e) {  
 e.printStackTrace();  
 }  
 }

4.新增地址

private void insertAddress() {  
 //保存当前收货地址，sqlite  
 String name = mEtName.getText().toString().trim();  
 String sex = "女士";  
 if (mRbMan.isChecked()) {  
 sex = "先生";  
 }  
 String phone = mEtPhone.getText().toString().trim();  
 String phoneOther = mEtPhoneOther.getText().toString().trim();  
 String address = mEtReceiptAddress.getText().toString().trim();  
 String detailAddress = mEtDetailAddress.getText().toString().trim();  
 String label = mTvLabel.getText().toString().trim();  
 RecepitAddress recepitAddress = new RecepitAddress(999, name, sex, phone, phoneOther, address, detailAddress, label, "36");  
 boolean isInsertSuccess = mAddressDao.insertAddress(recepitAddress);  
 if (isInsertSuccess) {  
 finish();  
 Toast.makeText(this, "新增地址成功", Toast.LENGTH\_SHORT).show();  
 } else {  
 Toast.makeText(this, "新增地址失败", Toast.LENGTH\_SHORT).show();  
 }  
}

5.查询地址

@Override  
protected void onStart() {  
 super.onStart();  
 //读出所有地址  
 List<RecepitAddress> addressList = mAddressDao.queryAllAddres();  
 if(addressList!=null && addressList.size()>0){  
 mAddressRvAdapter.setAddressList(addressList);  
 }  
}

**6**.修改地址

private void updateAddress() {  
 //更新  
 String name = mEtName.getText().toString().trim();  
 String sex = "女士";  
 if (mRbMan.isChecked()) {  
 sex = "先生";  
 }  
 String phone = mEtPhone.getText().toString().trim();  
 String phoneOther = mEtPhoneOther.getText().toString().trim();  
 String address = mEtReceiptAddress.getText().toString().trim();  
 String detailAddress = mEtDetailAddress.getText().toString().trim();  
 String label = mTvLabel.getText().toString().trim();  
 //把新的数据set到mAddressBean里  
 mAddress.setName(name);  
 mAddress.setSex(sex);  
 mAddress.setPhone(phone);  
 mAddress.setPhoneOther(phoneOther);  
 mAddress.setAddress(address);  
 mAddress.setDetailAddress(detailAddress);  
 mAddress.setLabel(label);  
 boolean isUpdateSuccess = mAddressDao.updateAddress(mAddress);  
 if (isUpdateSuccess) {  
 finish();  
 Toast.makeText(this, "更新地址成功", Toast.LENGTH\_SHORT).show();  
 } else {  
 Toast.makeText(this, "更新地址失败", Toast.LENGTH\_SHORT).show();  
 }  
}

7.删除地址

private void deleteAddress() {  
 AlertDialog.Builder builder = new AlertDialog.Builder(this);  
 builder.setTitle("确认删除这个地址么？");  
 builder.setPositiveButton("是的", new DialogInterface.OnClickListener() {  
 @Override  
 public void onClick(DialogInterface dialog, int which) {  
 boolean isOk = mAddressDao.deleteAddress(mAddress);  
 if (isOk) {  
 Toast.makeText(AddOrEditAddressActivity.this, "删除地址成功", Toast.LENGTH\_SHORT).show();  
 finish();  
 } else {  
 Toast.makeText(AddOrEditAddressActivity.this, "删除地址失败", Toast.LENGTH\_SHORT).show();  
 }  
 }  
 });  
 builder.setNegativeButton("不，保留", new DialogInterface.OnClickListener() {  
 @Override  
 public void onClick(DialogInterface dialog, int which) {  
  
 }  
 });  
 builder.show();  
}

## 每日作业

1.购物车展示

2.点餐的缓存

3.地址管理

Dagger2的源码分析

1.**public** GoodsFragmentComponent build() {  
 **if** (**goodsFragmentMoudule** == **null**) {  
 **throw new** IllegalStateException(  
 GoodsFragmentMoudule.**class**.getCanonicalName() + **" must be set"**);  
 }

必须挂载module对象（女孩），否则报错（不准领证）

1. 初始化（结婚后度蜜月）

**private void** initialize(**final** Builder builder) {  
  
 **this**.**providerGoodsFragmentPresenterProvider** =  
 GoodsFragmentMoudule\_ProviderGoodsFragmentPresenterFactory.*create*(  
 builder.**goodsFragmentMoudule**);  
  
 **this**.**goodsFragmentMembersInjector** =  
 GoodsFragment\_MembersInjector.*create*(**providerGoodsFragmentPresenterProvider**);  
}

1. 注入器工作

@Override  
**public void** in(GoodsFragment goodsFragment) {  
 **goodsFragmentMembersInjector**.injectMembers(goodsFragment);  
}

1. 具体注入

@Override  
**public void** injectMembers(GoodsFragment instance) {  
 **if** (instance == **null**) {  
 **throw new** NullPointerException(**"Cannot inject members into a null reference"**);  
 }  
 instance.**mGoodsFragmentPresenter** = **mGoodsFragmentPresenterProvider**.get();  
}

*mGoodsFragmentPresenter = new GoodsFragmentPresenter(this);*

注入器相当于 = 号， 提供器相当于new GoodsFragmentPresenter