```
// 前 30 页
package com.gorden.dayexam

import android.app.Application
import com.gorden.dayexam.db.AppDatabase
import com.gorden.dayexam.executor.AppExecutors
import com.gorden.dayexam.repository.DataRepository
import com.jeremyliao.liveeventbus.LiveEventBus

class BasicApp: Application() {

    override fun onCreate() {
        super.onCreate()
        ContextHolder.application = this
        LiveEventBus
            .config()
            .setContext(this)
        DataRepository.init(AppDatabase.getInstance(this, AppExecutors))
    }

}

package com.gorden.dayexam

import android.os.Bundle
import androidx.appcompat.app.AppCompatActivity

open class BaseActivity: AppCompatActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        ContextHolder.onActivityCreated(this, savedInstanceState)
    }

    override fun onStart() {
        super.onStart()
        ContextHolder.onActivityStarted(this)
    }

    override fun onResume() {
        super.onResume()
        ContextHolder.onActivityResumed(this)
    }
}
package com.gorden.dayexam

import android.Manifest
import android.annotation.SuppressLint
import android.content.Intent
import android.content.pm.PackageManager
import android.os.Bundle
import android.provider.MediaStore
import android.util.Log
import android.view.*
import android.widget.ImageView
import android.widget.TextView
import androidx.appcompat.app.ActionBarDrawerToggle
import androidx.appcompat.widget.Toolbar
import androidx.constraintlayout.widget.ConstraintLayout
import androidx.core.app.ActivityCompat
import androidx.core.content.ContextCompat
import androidx.core.content.res.ResourcesCompat
import androidx.drawerlayout.widget.DrawerLayout
import androidx.preference.PreferenceManager
import com.gorden.dayexam.db.DefaultDataGenerator
import com.gorden.dayexam.repository.DataRepository
import com.gorden.dayexam.ui.EventKey
import com.gorden.dayexam.ui.book.BooksFragment
import com.gorden.dayexam.ui.dialog.element.ImageElementEditCard
import com.gorden.dayexam.ui.home.HomeFragment
```

```
import com.gorden.dayexam.ui.home.shortcut.FastQuestionSelectActivity
import com.gorden.dayexam.ui.home.shortcut.FastQuestionSelectActivity.Companion.CURRENT_POSITION
import com.gorden.dayexam.ui.home.shortcut.FastQuestionSelectActivity.Companion.PAPER_ID_KEY
import com.gorden.dayexam.ui.home.shortcut.SimpleQuestionViewHolder
import com.gorden.dayexam.ui.sheet.course.CourseSheetDialog
import com.gorden.dayexam.ui.sheet.search.SearchSheetDialog
import com.gorden.dayexam.ui.sheet.shortcut.ShortCutSheetDialog
import com.jeremyliao.liveeventbus.LiveEventBus
import com.leinardi.android.speeddial.SpeedDialActionItem
import com.leinardi.android.speeddial.SpeedDialView
import com.microsoft.appcenter.AppCenter
import com.microsoft.appcenter.analytics.Analytics
import com.microsoft.appcenter.crashes.Crashes
import kotlinx.android.synthetic.main.app_bar_main.*
import java.io.IOException

class MainActivity : BaseActivity() {

    private lateinit var drawerLayout: DrawerLayout
    private val courseSheet = CourseSheetDialog()
    private val shortCutSheet = ShortCutSheetDialog()
    private val searchSheet = SearchSheetDialog()
    private val homeFragment = HomeFragment()
    private val bookListFragment = BooksFragment()
    private var curCourseId = 0
    private var curCourseTitle = ""
    private var curPaperId = 0
    private var curQuestionId = 0
    private var lastHomepagePosition = -1

    // config 相关
    private var isFocusMode = false
    private var questionMarginTop = 0

    private lateinit var todayCount: TextView

    private var photoSelectCallback: ImageElementEditCard.PhotoSelectCallback? = null

    companion object {
        const val SELECT_QUESTION_REQUEST_CODE = 201
        const val SELECT_QUESTION_RESULT_CODE = 202

        const val SELECT_PHOTO_REQUEST_CODE = 301
    }

    @SuppressLint("ResourceType")
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        AppCenter.start(
            application, "f1dc024a-c1a1-4cf6-ae38-d5a264b6728b",
            Analytics::class.java, Crashes::class.java
        )
        Crashes.setEnabled(true)
        setContentView(R.layout.activity_main)
        drawerLayout = findViewById(R.id.drawer_layout)
        initToolBar()
        initFab()
        initFragment()
        registerEvent()
        observeDatabase()
        observeCurrentCourse()
        observeDContext()
        observeConfig()
        observeTodayStudyCount()
        checkScreenLight()
    }

    fun closeDrawerLayout() {
        drawerLayout.closeDrawer(Gravity.LEFT)
```

```
    }

    private fun initToolBar() {
        val toolbar: Toolbar = findViewById(R.id.toolbar)
        toolbar.setTitleTextAppearance(this, R.style.XWWKBoldTextAppearance)
        setSupportActionBar(toolbar)
        supportActionBar?.setDisplayHomeAsUpEnabled(true)
        supportActionBar?.setDisplayShowTitleEnabled(false)
        val toggle = ActionBarDrawerToggle(
            this,
            drawerLayout,
            toolbar,
            0,
            0
        )
        toggle.isDrawerIndicatorEnabled = true
        drawerLayout.addDrawerListener(toggle)
        toggle.syncState()
        toolbar.findViewById<TextView>(R.id.title).setOnClickListener {
            showCourseSheet()
        }
        toolbar.findViewById<ImageView>(R.id.title_drop_down).setOnClickListener {
            showCourseSheet()
        }
        todayCount = toolbar.findViewById(R.id.today_study_count)
    }

    private fun showCourseSheet() {
        courseSheet.show(
            supportFragmentManager,
            "course"
        )
    }

    private fun initFab() {
        val fab: SpeedDialView = findViewById(R.id.fab)
        fab.addActionItem(
            SpeedDialActionItem.Builder(
                R.id.float_button_reset_question_item,
                R.drawable.ic_baseline_refresh_24
            )
                .setFabBackgroundColor(
                    ResourcesCompat.getColor(
                        resources,
                        R.color.colorPrimaryDark,
                        theme
                    )
                )
                .setFabImageTintColor(
                    ResourcesCompat.getColor(
                        resources,
                        R.color.float_button_item_icon_color,
                        theme
                    )
                )
                .setLabel(getString(R.string.reset_current_question))
                .setLabelClickable(false)
                .setTheme(R.style.FloatButtonTextAppearance)
                .create()
        )
//          fab.addActionItem(
//              SpeedDialActionItem.Builder(
//                  R.id.float_button_favorite_question_item,
//                  R.drawable.ic_baseline_favorite_border_24
//              )
//                  .setFabBackgroundColor(
//                      ResourcesCompat.getColor(
//                          resources,
//                          R.color.colorPrimaryDark,
```

```
//                                  theme
//                              )
//                          )
//                          .setFabImageTintColor(
//                              ResourcesCompat.getColor(
//                                  resources,
//                                  R.color.float_button_item_icon_color,
//                                  theme
//                              )
//                          )
//                          .setLabel(getString(R.string.favorite_question))
//                          .setLabelClickable(false)
//                          .setTheme(R.style.FloatButtonTextAppearance)
//                          .create()
//              )
            fab.addActionItem(
                SpeedDialActionItem.Builder(
                    R.id.float_button_list_question_item,
                    R.drawable.ic_outline_format_list_numbered_24
                )
                    .setFabBackgroundColor(
                        ResourcesCompat.getColor(
                            resources,
                            R.color.colorPrimaryDark,
                            theme
                        )
                    )
                    .setFabImageTintColor(
                        ResourcesCompat.getColor(
                            resources,
                            R.color.float_button_item_icon_color,
                            theme
                        )
                    )
                    .setLabel(getString(R.string.list_question))
                    .setLabelClickable(false)
                    .setTheme(R.style.FloatButtonTextAppearance)
                    .create()
            )
            fab.setOnActionSelectedListener {
                when(it.id) {
                    R.id.float_button_reset_question_item -> {
                        LiveEventBus.get(EventKey.REFRESH_QUESTION, Int::class.java).post(0)
                        fab.close()
                    }
                    R.id.float_button_favorite_question_item -> {
                        LiveEventBus.get(EventKey.FAVORITE_QUESTION, Int::class.java).post(0)
                        fab.close()
                    }
                    R.id.float_button_list_question_item -> {
                        val intent = Intent(this, FastQuestionSelectActivity::class.java)
                        intent.putExtra(PAPER_ID_KEY, curPaperId)
                        lastHomepagePosition = homeFragment.currentPosition()
                        intent.putExtra(CURRENT_POSITION, lastHomepagePosition)
                        startActivityForResult(intent, SELECT_QUESTION_REQUEST_CODE)
                        fab.close()
                    }
                }
                return@setOnActionSelectedListener true
            }
        }

        override fun onActivityResult(requestCode: Int, resultCode: Int, data: Intent?) {
            super.onActivityResult(requestCode, resultCode, data)
            if (requestCode == SELECT_QUESTION_REQUEST_CODE && resultCode == SELECT_QUESTION_RESULT_CODE) {
                val selectPosition = data?.getIntExtra(SimpleQuestionViewHolder.SELECT_POSITION, -1) ?: -1
                if (selectPosition == -1 || selectPosition == lastHomepagePosition) {
                    return
                }
```

```
            homeFragment.setCurrentPosition(selectPosition)
        } else if (requestCode == SELECT_PHOTO_REQUEST_CODE) {
            val selectedImage = data?.data
            try {
                val bitmap = MediaStore.Images.Media.getBitmap(contentResolver, selectedImage)
                photoSelectCallback?.onSelect(bitmap)
            } catch (e: IOException) {
                Log.e("", "")
            }
        }
    }

    private fun initFragment() {
        supportFragmentManager
            .beginTransaction()
            .add(R.id.fragment_content, homeFragment)
            .add(R.id.book_list_container, bookListFragment)
            .commit()
    }

    override fun onCreateOptionsMenu(menu: Menu): Boolean {
        menuInflater.inflate(R.menu.main, menu)
        return true
    }

    override fun onOptionsItemSelected(item: MenuItem): Boolean {
        when (item.itemId) {
            R.id.action_settings -> {
                shortCutSheet.show(
                    supportFragmentManager,
                    "shortcut"
                )
            }
        }
        return true
    }

    private fun registerEvent() {
        LiveEventBus.get(EventKey.SEARCH_CLICKED, Int::class.java)
            .observe(this, {
                drawerLayout.closeDrawers()
                searchSheet.show(
                    supportFragmentManager,
                    "Search"
                )
            })

        LiveEventBus.get(
            EventKey.EDIT_SELECT_PHOTO_CLICK,
            ImageElementEditCard.PhotoSelectCallback::class.java
        )
            .observe(this, {
                this.photoSelectCallback = it
                selectPhoto()
            })
        LiveEventBus.get(EventKey.START_PROGRESS_BAR, Int::class.java).observe(this, {
            findViewById<View>(R.id.parsing_progress).visibility = View.VISIBLE
        })
        LiveEventBus.get(EventKey.END_PROGRESS_BAR, Int::class.java).observe(this, {
            drawerLayout.postDelayed({
                findViewById<View>(R.id.parsing_progress).visibility = View.GONE
            }, 1000)
        })
        LiveEventBus.get(EventKey.PAPER_MENU_ADD_QUESTION_FROM_FILE, EventKey.QuestionAddEventModel::class.java)
            .observe(this, {
                drawerLayout.closeDrawers()
            })
    }
```

```
private fun observeDatabase() {
    DataRepository.isDatabaseCreated().observe(this, {
        if (it) {
            DefaultDataGenerator.generate()
        }
    })
}

private fun observeCurrentCourse() {
    DataRepository.currentCourse().observe(this, {
        if (it == null) {
            toolbar.findViewById<TextView>(R.id.title).text = ""
            return@observe
        }
        if (it != null && it.id != curCourseId) {
            toolbar.findViewById<TextView>(R.id.title).text = it.title
            drawerLayout.openDrawer(Gravity.LEFT)
            curCourseId = it.id
            curCourseTitle = it.title
        }
        if (it != null && it.title != curCourseTitle) {
            toolbar.findViewById<TextView>(R.id.title).text = it.title
            curCourseTitle = it.title
        }
    })
}

private fun observeDContext() {
    DataRepository.getDContext().observe(this, {
        if (it != null) {
            curPaperId = it.curPaperId
            curQuestionId = it.curQuestionId
        }
    })
}

private fun observeConfig() {
    DataRepository.getConfig().observe(this, { config ->
        config?.let {
            this.isFocusMode = it.focusMode
            if (it.focusMode) {
                supportActionBar?.hide()
                val layoutParams =
                    (fragment_content.layoutParams as ConstraintLayout.LayoutParams)
                if (questionMarginTop == 0 && layoutParams.topMargin != 0) {
                    questionMarginTop = layoutParams.topMargin
                }
                layoutParams.topMargin = 0
                fragment_content.layoutParams = layoutParams
                fab.visibility = View.GONE
            } else {
                supportActionBar?.show()
                if (questionMarginTop != 0) {
                    val layoutParams =
                        (fragment_content.layoutParams as ConstraintLayout.LayoutParams)
                    layoutParams.topMargin = questionMarginTop
                    fragment_content.layoutParams = layoutParams
                }
                fab.visibility = View.VISIBLE
            }
        }

    })
}

private fun observeTodayStudyCount() {
    DataRepository.todayStudyCount().observe(this, {
        todayCount.text = it.toString()
    })
```

```
    }

    private fun selectPhoto() {
        //动态申请权限
        if (ContextCompat.checkSelfPermission(
                    this, Manifest.permission
                        .WRITE_EXTERNAL_STORAGE
                ) != PackageManager.PERMISSION_GRANTED){
                ActivityCompat.requestPermissions(
                    this,
                    arrayOf(Manifest.permission.WRITE_EXTERNAL_STORAGE), 1
                )
        }else{
            //执行启动相册的方法
            openAlbum();
        }
    }

    //启动相册的方法
    private fun openAlbum() {
        val intent = Intent(Intent.ACTION_PICK)
        intent.type = "image/*"
        ActivityCompat.startActivityForResult(this, intent, SELECT_PHOTO_REQUEST_CODE, null)
    }

    private fun checkScreenLight() {
        val sharedPreferences = PreferenceManager.getDefaultSharedPreferences(ContextHolder.currentActivity())
        val keepScreenOnKey = ContextHolder.application.resources.getString(R.string.keep_screen_light_key)
        val keepScreenOn = sharedPreferences.getBoolean(keepScreenOnKey, false)
        if (keepScreenOn) {
            window.addFlags(WindowManager.LayoutParams.FLAG_KEEP_SCREEN_ON)
        } else {
            window.clearFlags(WindowManager.LayoutParams.FLAG_KEEP_SCREEN_ON)
        }
    }

}

@file:Suppress("RECEIVER_NULLABILITY_MISMATCH_BASED_ON_JAVA_ANNOTATIONS")

package com.gorden.dayexam.backup

import android.Manifest
import android.content.pm.PackageManager
import android.widget.Toast
import androidx.core.app.ActivityCompat
import androidx.core.content.ContextCompat
import androidx.preference.PreferenceManager
import com.gorden.dayexam.BuildConfig
import com.gorden.dayexam.ContextHolder
import com.gorden.dayexam.R
import com.gorden.dayexam.db.AppDatabase
import com.gorden.dayexam.executor.AppExecutors
import com.gorden.dayexam.parser.image.ImageCacheManager
import com.gorden.dayexam.repository.DataRepository
import com.gorden.dayexam.ui.EventKey
import com.gorden.dayexam.utils.FileUtils
import com.jeremyliao.liveeventbus.LiveEventBus
import net.lingala.zip4j.ZipFile
import org.json.JSONObject
import java.io.File
import java.text.SimpleDateFormat
import java.util.*

object BackupManager {

    private const val SDCARD_BACKUP_FOLDER_NAME = "backup"
    private const val SDCARD_BACKUP_FOR_RECOVER_NAME = "recover_temp"
    private val CACHE_IMAGE_FOLDER =
```

```
        ContextHolder.application.cacheDir.path + File.separator + ImageCacheManager.PARSED_IMAGE_FOLDER_NAME
private val EXAM_DATABASE_PATH =
        ContextHolder.application.getDatabasePath(AppDatabase.DATABASE_NAME).absolutePath
private val DATABASE_FOLDER =
        ContextHolder.application.dataDir.absolutePath + File.separator + "databases"
private val BACKUP_TEMP_FOLDER =
        ContextHolder.application.cacheDir.path + File.separator + "backup_temp_extract"
private val TEMP_BACKUP_INFO_PATH =
        ContextHolder.application.cacheDir.path + File.separator + "backup_info.json"

fun backup() {
        //动态申请权限
        ContextHolder.currentActivity()?.let {
                if (ContextCompat.checkSelfPermission(
                                it, Manifest.permission
                                        .WRITE_EXTERNAL_STORAGE
                        ) != PackageManager.PERMISSION_GRANTED
                ) {
                        ActivityCompat.requestPermissions(
                                it,
                                arrayOf(Manifest.permission.WRITE_EXTERNAL_STORAGE), 1
                        )
                } else {
                        AppExecutors.diskIO().execute {
                                LiveEventBus.get(EventKey.START_PROGRESS_BAR, Int::class.java).post(0)
                                backupTo(SDCARD_BACKUP_FOLDER_NAME)
                                AppExecutors.mainThread().execute {
                                        val context = ContextHolder.application
                                        LiveEventBus.get(EventKey.END_PROGRESS_BAR, Int::class.java).post(0)
                                        Toast.makeText(context,
                                                context.resources.getString(R.string.backup_success),
                                                Toast.LENGTH_SHORT).show()
                                }
                        }
                }
        }
}

fun recover(backupFile: File, listener: BackupListener) {
        backupCurrent()
        AppExecutors.diskIO().execute {
                if (backupFile.exists() && backupFile.name.endsWith(".zip")) {
                        try {
                                checkBackupTemp()
                                val zipFile = ZipFile(backupFile.absolutePath)
                                zipFile.extractAll(BACKUP_TEMP_FOLDER)
                                val tempFile = File(BACKUP_TEMP_FOLDER)
                                tempFile.listFiles().forEach {
                                        if (it.name.equals(AppDatabase.DATABASE_NAME)) {
                                                val success = it.renameTo(File(EXAM_DATABASE_PATH))
                                                if (success) {
                                                        val shmFile =
                                                                File(DATABASE_FOLDER + File.separator + AppDatabase.DATABASE_NAME + "-shm")
                                                        if (shmFile.exists()) {
                                                                shmFile.delete()
                                                        }
                                                        val walFile =
                                                                File(DATABASE_FOLDER + File.separator + AppDatabase.DATABASE_NAME + "-wal")
                                                        if (walFile.exists()) {
                                                                walFile.delete()
                                                        }
                                                } else {
                                                        AppExecutors.mainThread().execute {
                                                                listener.onRecoverEnd(false, "恢复数据库异常")
                                                        }
                                                }
                                        } else if (it.name.equals(ImageCacheManager.PARSED_IMAGE_FOLDER_NAME) && it.isDirectory) {
                                                val imageCache = File(CACHE_IMAGE_FOLDER)
                                                if (imageCache.exists()) {
```

```
                                FileUtils.deleteDir(imageCache)
                            }
                            val success = it.renameTo(File(CACHE_IMAGE_FOLDER))
                            if (!success) {
                                AppExecutors.mainThread().execute {
                                    listener.onRecoverEnd(false, "恢复图片异常")
                                }
                            }
                        }
                    }
                    AppExecutors.mainThread().execute {
                        listener.onRecoverEnd(true, "")
                    }
                } catch (e: java.lang.Exception) {
                    AppExecutors.mainThread().execute {
                        listener.onRecoverEnd(false, "解析过程抛出异常")
                    }
                }

            } else {
                AppExecutors.mainThread().execute {
                    listener.onRecoverEnd(false, "文件格式出现问题")
                }
            }
        }
    }

    fun getAllBackupFiles(): List<File> {
        val externalPath = ContextHolder.application.getExternalFilesDir(null)?.absolutePath
        val parentPath = externalPath + File.separator + SDCARD_BACKUP_FOLDER_NAME
        val folder = File(parentPath)
        if (!folder.exists() || !folder.isDirectory) {
            return listOf()
        }
        return folder.listFiles().filter {
            it.name.endsWith(".zip")
        }.sortedBy {
            it.name
        }
    }

    private fun deleteEarliestBackup() {
        try {
            val backups = getAllBackupFiles()
            val maxSize = getMaxBackupSize()
            if (backups.size <= maxSize) {
                return
            }
            if (backups.size > maxSize) {
                for (i in 0..(backups.size - maxSize)) {
                    if (backups[i].exists()) {
                        backups[i].delete()
                    }
                }
            }
            val earliestZipFile = backups[0]
            if (earliestZipFile.exists() && earliestZipFile.isFile) {
                earliestZipFile.delete()
            }
        } catch (e: Exception) {

        }
    }

    private fun getMaxBackupSize(): Int {
        val sharedPreferences =
            PreferenceManager.getDefaultSharedPreferences(ContextHolder.currentActivity())
        val maxKey = ContextHolder.application.resources.getString(R.string.max_backup_size_key)
        return sharedPreferences.getInt(maxKey, 5)
```

```
    }

    private fun checkBackupTemp() {
        val tempFolder = File(BACKUP_TEMP_FOLDER)
        if (tempFolder.exists()) {
            FileUtils.deleteDirContent(tempFolder)
        } else {
            tempFolder.mkdir()
        }
    }

    private fun backupTo(path: String) {
        try {
            DataRepository.checkPoint()
            deleteEarliestBackup()
            val externalPath =
                ContextHolder.application.getExternalFilesDir(null)?.absolutePath
            val parentPath = externalPath + File.separator + path
            val folder = File(parentPath)
            if (!folder.exists() || !folder.isDirectory) {
                folder.mkdir()
            }
            val date = Date()
            val formatter = SimpleDateFormat("yyyy-MM-dd HH:mm:ss")
            val fileName = formatter.format(date)
            val destination = parentPath + File.separator + fileName + ".zip"
            val backInfo = JSONObject()
            backInfo.put("version_code", BuildConfig.VERSION_CODE)
            backInfo.put("version_name", BuildConfig.VERSION_NAME)
            val backInfoString = backInfo.toString()
            FileUtils.writeTo(TEMP_BACKUP_INFO_PATH, backInfoString)
            FileUtils.zip(listOf(EXAM_DATABASE_PATH, CACHE_IMAGE_FOLDER, TEMP_BACKUP_INFO_PATH), destination)
            val backupInfoFile = File(TEMP_BACKUP_INFO_PATH)
            if (backupInfoFile.exists()) {
                backupInfoFile.delete()
            }
        } catch (e: Exception) {

        }
    }

    // 恢复备份前现将当前的内容备份到`SDCARD_BACKUP_FOR_RECOVER_NAME`，以防出错无法恢复
    private fun backupCurrent() {
        AppExecutors.diskIO().execute {
            val externalPath = ContextHolder.application.getExternalFilesDir(null)?.absolutePath
            val recoverTemp = externalPath + File.separator + SDCARD_BACKUP_FOR_RECOVER_NAME
            val file = File(recoverTemp)
            if (file.exists() && file.isDirectory) {
                FileUtils.deleteDirContent(file)
            } else {
                file.mkdir()
            }
            backupTo(SDCARD_BACKUP_FOR_RECOVER_NAME)
        }
    }

    interface BackupListener {
        fun onRecoverEnd(success: Boolean, msg: String)
    }

}
package com.gorden.dayexam.db.converter

import android.annotation.SuppressLint
import androidx.room.TypeConverter
import java.lang.Exception
import java.text.SimpleDateFormat
import java.util.*
```

```
object DateConverter {
    @SuppressLint("SimpleDateFormat")
    @TypeConverter
    fun toDate(timestamp: String?): Date? {
        if (timestamp.isNullOrEmpty()) return null
        val formatter = SimpleDateFormat("yyyy-MM-dd HH:mm:ss:SSS")
        formatter.timeZone = TimeZone.getDefault()
        try {
            return formatter.parse(timestamp)
        } catch (e: Exception) {
            e.printStackTrace()
        }
        return null
    }

    @SuppressLint("SimpleDateFormat")
    @TypeConverter
    fun toTimestamp(date: Date): String {
        val formatter = SimpleDateFormat("yyyy-MM-dd HH:mm:ss:SSS")
        return formatter.format(date)
    }

}
package com.gorden.dayexam.db.dao

import androidx.lifecycle.LiveData
import androidx.room.*
import com.gorden.dayexam.db.entity.Book

@Dao
interface BookDao {

    @Insert
    fun insert(book: Book): Long

    @Insert
    fun insert(books: List<Book>)

    @Query("SELECT * FROM book WHERE id = :bookId")
    fun getById(bookId: Int): LiveData<Book>

    @Query("SELECT * FROM book WHERE id = :bookId")
    fun getEntity(bookId: Int): Book?

    @Query ("SELECT * FROM book WHERE courseId = :courseId ORDER BY position ASC")
    fun getBookEntitiesByCourseId(courseId: Int): List<Book>

    @Query ("SELECT * FROM book WHERE courseId = :courseId ORDER BY position ASC")
    fun getBookByCourseId(courseId: Int): LiveData<List<Book>>

    @Update
    fun update(books: List<Book>)

    @Update
    fun update(book: Book)

    @Query("SELECT position FROM book WHERE courseId = :courseId ORDER BY position DESC LIMIT 1")
    fun getMaxOrder(courseId: Int): Int

    @Query("DELETE FROM book WHERE id = :id")
    fun delete(id: Int)

    @Delete
    fun delete(books: List<Book>)

    @Delete
    fun delete(book: Book)
}
package com.gorden.dayexam.db.dao
```

```
import androidx.lifecycle.LiveData
import androidx.room.Dao
import androidx.room.Insert
import androidx.room.Query
import androidx.room.Update
import com.gorden.dayexam.db.entity.Config

@Dao
interface ConfigDao {
    @Insert
    fun insert(config: Config): Long

    @Query("SELECT * FROM config")
    fun getEntity(): Config

    @Query("SELECT * FROM config")
    fun get(): LiveData<Config>

    @Update
    fun update(config: Config)
}
package com.gorden.dayexam.db.dao

import androidx.room.Dao
import androidx.room.Delete
import androidx.room.Insert
import androidx.room.Query
import com.gorden.dayexam.db.entity.question.Content

@Dao
interface ContentDao {
    @Insert
    fun insert(content: Content): Long

    @Query("SELECT * FROM content WHERE questionId = :questionId AND contentType = 1")
    fun getBodyEntity(questionId: Int): Content

    @Query("SELECT * FROM content WHERE questionId = :questionId AND contentType = 2")
    fun getOptionEntity(questionId: Int): List<Content>

    @Query("SELECT * FROM content WHERE questionId = :questionId AND contentType = 3")
    fun getAnswerEntity(questionId: Int): Content

    @Delete
    fun delete(content: Content)

    @Query("DELETE FROM content WHERE questionId = :questionId")
    fun delete(questionId: Int)
}
package com.gorden.dayexam.db.dao

import androidx.lifecycle.LiveData
import androidx.room.*
import com.gorden.dayexam.db.entity.Book
import com.gorden.dayexam.db.entity.Course
import com.gorden.dayexam.db.entity.Paper
import com.gorden.dayexam.db.entity.question.Content
import com.gorden.dayexam.db.entity.question.Element
import com.gorden.dayexam.db.entity.question.Question

@Dao
interface CourseDao {

    @Query("SELECT * FROM course ORDER BY position ASC")
    fun getAllCourse(): LiveData<List<Course>>

    @Query("SELECT * FROM course ORDER BY position ASC")
    fun getAllCourseWithChildren(): List<CourseWithChildren>
```

```
    @Query("SELECT * FROM course")
    fun getAllCourseEntity(): List<Course>

    @Insert
    fun insert(course: Course): Long

    @Insert
    fun insert(list: List<Course>)

    @Update
    fun update(course: Course)

    @Delete
    fun delete(course: Course)

    @Query("SELECT * FROM course WHERE id = :id ORDER BY position DESC")
    fun getCourse(id: Int): LiveData<Course>

    @Query("SELECT * FROM course WHERE id = :id")
    fun getCourseEntity(id: Int): Course?

    @Query("SELECT position FROM course WHERE isRecycleBin = 0 ORDER BY position DESC LIMIT 1")
    fun getMaxOrder(): Int
}

data class CourseWithChildren(
    @Embedded val course: Course,
    @Relation(
        entity = Book::class,
        parentColumn = "id",
        entityColumn = "courseId"
    )
    val books: List<BookWithChildren>

)

data class BookWithChildren(
    @Embedded val book: Book,
    @Relation(
        entity = Paper::class,
        parentColumn = "id",
        entityColumn = "bookId"
    )
    val papers: List<PaperWithChildren>
)

data class PaperWithChildren(
    @Embedded val paper: Paper,
    @Relation(
        entity = Question::class,
        parentColumn = "id",
        entityColumn = "paperId"
    )
    val questions: List<QuestionWithContent>
)

data class QuestionWithChildren(
    @Embedded val question: Question,
    @Relation(
        entity = Content::class,
        parentColumn = "id",
        entityColumn = "questionId"
    )
    val contents: List<ContentWithChildren>
)

data class ContentWithChildren(
    @Embedded val content: Content,
```

```
    @Relation(
        entity = Element::class,
        parentColumn = "contentId",
        entityColumn = "parentId"
    )
    val elements: List<Element>
)
package com.gorden.dayexam.db.dao

import androidx.lifecycle.LiveData
import androidx.room.*
import androidx.sqlite.db.SupportSQLiteQuery
import com.gorden.dayexam.db.entity.DContext

@Dao
interface DContextDao {
    @Insert
    fun insert(dContext: DContext)

    @Update
    fun update(dContext: DContext)

    @Query("SELECT * FROM d_context WHERE id = 1")
    fun getDContext(): LiveData<DContext>

    @Query("SELECT * FROM d_context WHERE id = 1")
    fun getDContextEntity(): DContext

    // 非业务查询，用于将 wal 内容写入数据库，备份前调用
    @RawQuery
    fun checkpoint(supportSQLiteQuery: SupportSQLiteQuery?): Int
}
package com.gorden.dayexam.db.dao

import androidx.lifecycle.LiveData
import androidx.room.*
import com.gorden.dayexam.db.entity.Book
import com.gorden.dayexam.db.entity.Course
import com.gorden.dayexam.db.entity.Paper
import com.gorden.dayexam.db.entity.question.Content
import com.gorden.dayexam.db.entity.question.Element
import com.gorden.dayexam.db.entity.question.Question

@Dao
interface ElementDao {
    @Insert
    fun insert(element: Element): Long

    @Insert
    fun insert(elements: List<Element>)

    @Query("SELECT * FROM element WHERE parentId = :parentId")
    fun getElementsEntity(parentId: Int): List<Element>

    @Delete
    fun delete(elements: List<Element>)

    @Query("DELETE FROM element WHERE parentId = :parentId")
    fun delete(parentId: Int)

    @Query("SELECT * FROM element WHERE parentId = :contentId")
    fun getByContentId(contentId: Int): LiveData<List<Element>>

    @Query("DELETE FROM element WHERE parentId = :contentId")
    fun deleteByContentId(contentId: Int)

    @Query("SELECT * FROM element, content, question " +
            "WHERE element.parentId = content.contentId " +
            "AND content.questionId = question.id " +
```

```
            "AND question.id = :questionId")
    fun getByQuestionId(questionId: Int): LiveData<List<ElementWithContentAncestors>>

    @Query("SELECT * FROM element, content, question, d_context " +
            "WHERE element.elementType = 0 " +
            "AND element.content LIKE '%' || :key || '%' " +
            "AND element.parentId = content.contentId " +
            "AND content.questionId = question.id " +
            "AND question.paperId = d_context.curPaperId ")
    fun searchInPaper(key: String): List<ElementWithContentAncestors>

    @Query("SELECT * FROM element, content, question, paper, book, d_context " +
            "WHERE element.parentId = content.contentId " +
            "AND content.questionId = question.id " +
            "AND question.paperId = paper.id " +
            "AND paper.bookId = book.id " +
            "AND book.id = d_context.curBookId " +
            "AND element.elementType = 0 " +
            "AND element.content LIKE '%' || :key || '%' ")
    fun searchInBook(key: String): List<ElementWithContentAncestors>

    @Query("SELECT * FROM element, content, question, paper, book, course, d_context " +
            "WHERE element.parentId = content.contentId " +
            "AND content.questionId = question.id " +
            "AND question.paperId = paper.id " +
            "AND paper.bookId = book.id " +
            "AND book.courseId = course.id " +
            "AND course.id = d_context.curCourseId " +
            "AND element.elementType = 0 " +
            "AND element.content LIKE '%' || :key || '%' ")
    fun searchInCourse(key: String): List<ElementWithContentAncestors>

    @Query("SELECT * FROM element, content, question, paper, book, course, d_context " +
            "WHERE element.parentId = content.contentId " +
            "AND content.questionId = question.id " +
            "AND question.paperId = paper.id " +
            "AND paper.bookId = book.id " +
            "AND book.courseId = course.id " +
            "AND course.id = d_context.recycleBinId " +
            "AND element.elementType = 0 " +
            "AND element.content LIKE '%' || :key || '%' ")
    fun searchInRecycleBin(key: String): List<ElementWithContentAncestors>

    @Query("SELECT * FROM element, content, question, paper, book, course, d_context " +
            "WHERE element.parentId = content.contentId " +
            "AND content.questionId = question.id " +
            "AND question.paperId = paper.id " +
            "AND paper.bookId = book.id " +
            "AND book.courseId = course.id " +
            "AND element.elementType = 0 " +
            "AND element.content LIKE '%' || :key || '%' ")
    fun searchGlobal(key: String): List<ElementWithContentAncestors>
}

data class BookWithCourse(
    @Embedded val book: Book,
    @Relation(
        entity = Course::class,
        parentColumn = "courseId",
        entityColumn = "id"
    )
    val course: Course
)

data class PaperWithBookAndCourse(
    @Embedded val paper: Paper,
    @Relation(
        entity = Book::class,
        parentColumn = "bookId",
```

```
            entityColumn = "id"
        )
        val book: BookWithCourse
)

data class QuestionWithAncestors(
        @Embedded val question: Question,
        @Relation(
                entity = Paper::class,
                parentColumn = "paperId",
                entityColumn = "id"
        )
        val paper: PaperWithBookAndCourse
)

data class ContentWithAncestors(
        @Embedded val content: Content,
        @Relation(
                entity = Question::class,
                parentColumn = "questionId",
                entityColumn = "id"
        )
        val question: QuestionWithAncestors
)

data class ElementWithContentAncestors(
        @Embedded val element: Element,
        @Relation(
                entity = Content::class,
                parentColumn = "parentId",
                entityColumn = "contentId"
        )
        val content: ContentWithAncestors
)
package com.gorden.dayexam.db.dao

import androidx.lifecycle.LiveData
import androidx.room.*
import com.gorden.dayexam.db.entity.Paper

@Dao
interface PaperDao {

        @Insert
        fun insert(papers: List<Paper>)

        @Insert
        fun insert(paper: Paper): Long

        @Update
        fun update(papers: List<Paper>)

        @Delete
        fun delete(papers: List<Paper>)

        @Update
        fun update(paper: Paper)

        @Delete
        fun delete(paper: Paper)

        @Query("DELETE FROM paper WHERE id = :id")
        fun delete(id: Int)

        @Query("SELECT * FROM paper WHERE id = :id")
        fun getById(id: Int): LiveData<Paper>

        @Query("SELECT * FROM paper WHERE id = :id")
        fun getEntityById(id: Int): Paper?
```

```
    @Query("SELECT * FROM paper WHERE bookId = :bookId ORDER BY position ASC")
    fun getEntityByBookIdOrderByPosition(bookId: Int): List<Paper>

    @Query("SELECT * FROM paper WHERE bookId = :bookId ORDER BY editTime DESC")
    fun getEntityByBookIdOrderByEditTime(bookId: Int): List<Paper>

    @Query("SELECT position FROM paper WHERE bookId = :bookId ORDER BY position DESC LIMIT 1")
    fun getMaxPosition(bookId: Int): Int

    @Query("SELECT * FROM paper WHERE bookId = :bookId ORDER BY position ASC")
    fun getByBookId(bookId: Int): LiveData<List<Paper>>
}
package com.gorden.dayexam.db.dao

import androidx.lifecycle.LiveData
import androidx.room.*
import com.gorden.dayexam.db.entity.question.Content
import com.gorden.dayexam.db.entity.question.Element
import com.gorden.dayexam.db.entity.question.Question

@Dao
interface QuestionDao {

    @Insert
    fun insert(question: Question): Long

    @Insert
    fun insert(questions: List<Question>)

    @Query ("SELECT * FROM question WHERE paperId = :paperId")
    fun getByPaperId(paperId: Int): LiveData<List<Question>>

    @Query ("SELECT * FROM question WHERE paperId = :paperId")
    fun getEntityByPaperId(paperId: Int): List<Question>

    @Query ("SELECT * FROM question WHERE paperId = :paperId ORDER BY questionPosition")
    fun getEntityWithContentByPaperId(paperId: Int): List<QuestionWithContent>

    @Query ("SELECT * FROM question WHERE id = :id")
    fun getEntityById(id: Int): Question

    @Query ("SELECT * FROM question WHERE id = :id")
    fun getEntityWithContentById(id: Int): QuestionWithContent

    // 找到 position 的上一个试题
    @Query("SELECT * FROM question WHERE paperId = :paperId AND questionPosition < :position ORDER BY questionPosition DESC LIMIT
1")
    fun getPrePositionQuestion(paperId: Int, position: Int): Question?

    // 找到 position 的下一个试题
    @Query("SELECT * FROM question WHERE paperId = :paperId AND questionPosition > :position ORDER BY questionPosition ASC LIMIT
1")
    fun getNextPositionQuestion(paperId: Int, position: Int): Question?

    @Query ("SELECT COUNT() FROM question WHERE paperId = :paperId")
    fun getCountWithPaperId(paperId: Int): Long

    @Query("SELECT questionPosition FROM question WHERE paperId = :paperId ORDER BY questionPosition DESC LIMIT 1")
    fun getMaxPosition(paperId: Int): Int

    @Update
    fun update(questions: List<Question>)

    @Update
    fun update(question: Question)

    @Delete
    fun delete(questions: List<Question>)
```

```
}

data class QuestionWithContent(
    @Embedded val question: Question,
    @Relation(
        entity = Content::class,
        parentColumn = "id",
        entityColumn = "questionId"
    )
    val contents: List<ContentWithElement>)

data class ContentWithElement(
    @Embedded val content: Content,
    @Relation(
        parentColumn = "contentId",
        entityColumn = "parentId"
    )
    var elements: List<Element>
)
package com.gorden.dayexam.db.dao

import androidx.lifecycle.LiveData
import androidx.room.Dao
import androidx.room.Insert
import androidx.room.Query
import com.gorden.dayexam.db.entity.StudyRecord

@Dao
interface StudyRecordDao {

    @Insert
    fun insert(studyRecord: StudyRecord): Long

    @Query("SELECT COUNT(id) FROM study_record WHERE questionId = :questionId")
    fun getQuestionStudyCount(questionId: Int): Long

    @Query("SELECT COUNT(id) FROM study_record WHERE paperId = :paperId")
    fun getPaperStudyCount(paperId: Int): Long

    @Query("SELECT * FROM study_record WHERE questionId = :questionId ORDER BY createTime DESC LIMIT 1")
    fun getLast(questionId: Int): StudyRecord?

    @Query("SELECT COUNT(id) FROM study_record WHERE createTime > :date")
    fun getStudyCountAfter(date: String): LiveData<Long>
}
package com.gorden.dayexam.db.dao

import androidx.lifecycle.LiveData
import androidx.room.*
import com.gorden.dayexam.db.entity.StudyStatus

@Dao
interface StudyStatusDao {

    @Insert
    fun insert(studyStatus: StudyStatus)

    @Insert
    fun insert(studyStatuses: List<StudyStatus>)

    @Query("SELECT * FROM study_status WHERE type = :type AND contentId = :contentId")
    fun queryEntityByTypeAndContentId(type: Int, contentId: Int): StudyStatus

    @Query("SELECT * FROM study_status WHERE type = :type AND contentId = :contentId")
    fun queryByTypeAndContentId(type: Int, contentId: Int): LiveData<StudyStatus>

    @Update
    fun update(studyStatus: StudyStatus)
```

```
    @Delete
    fun delete(studyStatus: StudyStatus)

    @Query("DELETE FROM study_status WHERE id = :id")
    fun delete(id: Int)

    @Query("DELETE FROM study_status WHERE type = :type AND contentId = :contentId")
    fun delete(type: Int, contentId: Int)
}
package com.gorden.dayexam.db.entity.question

import androidx.room.Entity
import androidx.room.PrimaryKey
import java.util.*

@Entity(tableName = "content")
data class Content(
    val questionId: Int,
    val contentType: Int
) {
    companion object {
        const val BODY_TYPE = 1
        const val OPTION_TYPE = 2
        const val ANSWER_TYPE = 3
    }
    @PrimaryKey(autoGenerate = true) var contentId: Int = 0
    var createTime = Date()
    var editTime = Date()
}
package com.gorden.dayexam.db.entity.question

import androidx.room.Entity
import androidx.room.PrimaryKey
import java.util.*

@Entity(tableName = "element")
data class Element (
    val elementType: Int,
    val content: String,
    var parentId: Int,
    var position: Int) {
    companion object {
        const val TEXT = 0
        const val PICTURE = 1
    }
    @PrimaryKey(autoGenerate = true) var id: Int = 0
    var createTime = Date()
    var editTime = Date()
}
package com.gorden.dayexam.db.entity.question

import androidx.room.Entity
import androidx.room.PrimaryKey
import java.util.*

@Entity (tableName = "question")
data class Question(
    var paperId: Int,
    val questionType: Int,
    var questionPosition: Int
) {
    @PrimaryKey(autoGenerate = true) var id: Int = 0
    var isFavorite = false
    var createTime = Date()
    var editTime = Date()
}

package com.gorden.dayexam.db.entity.tip
```

```
import androidx.room.Entity
import androidx.room.PrimaryKey
import java.util.*

@Entity(tableName = "tip")
data class Tip (
    val parentId: Int,
    val content: String )
{
    @PrimaryKey(autoGenerate = true) var id: Int = 0
    var createTime = Date()
    var editTime = Date()
}
package com.gorden.dayexam.db.entity

import androidx.room.Entity
import androidx.room.PrimaryKey
import java.util.*

@Entity (tableName = "book")
data class Book(
    var title: String,
    var description: String,
    var courseId: Int,
    var position: Int
) {
    @PrimaryKey(autoGenerate = true) var id: Int = 0
    var createTime = Date()
    var editTime = Date()
    var uuid = UUID.randomUUID()!!.toString()
}
package com.gorden.dayexam.db.entity

import androidx.room.Entity
import androidx.room.PrimaryKey
import java.util.*

@Entity(tableName = "Config")
data class Config (
    var rememberMode: Boolean,
    var focusMode: Boolean,
    var onlyFavorite: Boolean,
    var sortByAccuracy: Boolean)
{
    @PrimaryKey(autoGenerate = true) var id: Int = 0
    var createTime = Date()
    var editTime = Date()
}
package com.gorden.dayexam.db.entity

import androidx.room.Entity
import androidx.room.PrimaryKey
import java.util.*

@Entity (tableName = "course")
data class Course(
    var title: String,
    var description: String,
    var position: Int
) {
    @PrimaryKey (autoGenerate = true) var id: Int = 0
    var createTime = Date()
    var editTime = Date()
    var isRecycleBin = false
}
package com.gorden.dayexam.db.entity

import androidx.room.Entity
```

```
import androidx.room.PrimaryKey

@Entity(tableName = "d_context")
data class DContext(
    var curCourseId: Int,
    var curBookId: Int,
    var curPaperId: Int,
    var curQuestionId: Int,
    var recycleBinId: Int,
    var recycleBookId: Int,
    var recyclePaperId: Int
) {
    @PrimaryKey(autoGenerate = true) var id: Int = 0
    // 内容的版本号，当修改引起了试题内容的变更后，可以主动加 1 来主动触发全局更新，如果认为不需要触发全局更新，可以不管
    var version: Int = 0
}
package com.gorden.dayexam.db.entity

import androidx.room.Entity
import androidx.room.PrimaryKey
import java.util.*

@Entity (tableName = "paper")
data class Paper(
    var title: String,
    var description: String,
    var bookId: Int,
    var position: Int
) {
    @PrimaryKey(autoGenerate = true) var id: Int = 0
    var createTime = Date()
    var editTime = Date()
}

package com.gorden.dayexam.db.entity

import androidx.room.Entity
import androidx.room.PrimaryKey
import java.lang.StringBuilder
import java.util.*

@Entity(tableName = "study_record")
data class StudyRecord (
    val paperId: Int,
    val questionId: Int,
    val content: String,
    val correct: Int) {

    companion object {
        const val IN_CORRECT = 0
        const val CORRECT = 1
        const val NOT_AVAILABLE = -1

        fun parseFromBoolean(isCorrect: Boolean): Int {
            return if (isCorrect) CORRECT else IN_CORRECT
        }
    }

    @PrimaryKey(autoGenerate = true) var id: Int = 0
    var createTime = Date()
    var editTime = Date()
}

package com.gorden.dayexam.db.entity

import androidx.room.Entity
import androidx.room.PrimaryKey
import java.util.*
```

```
//    Three type of StudyStatus
const val CourseStatus = 1
const val BookStatus = 2
const val PaperStatus = 3

@Entity(tableName = "study_status")
data class StudyStatus(
        var type: Int,
        // contentId means course's id or Book's id or Paper's id
        var contentId: Int,
        // curChild means Book's id or    Paper's id or Question's id
        var curChild: Int
) {
        @PrimaryKey(autoGenerate = true) var id: Int = 0
        var createTime = Date()
        var editTime = Date()
}
package com.gorden.dayexam.db

import android.content.Context
import androidx.lifecycle.MutableLiveData
import androidx.room.Database
import androidx.room.Room
import androidx.room.RoomDatabase
import androidx.room.TypeConverters
import androidx.room.migration.Migration
import androidx.sqlite.db.SupportSQLiteDatabase
import com.gorden.dayexam.db.converter.*
import com.gorden.dayexam.db.dao.*
import com.gorden.dayexam.db.entity.*
import com.gorden.dayexam.db.entity.Paper
import com.gorden.dayexam.db.entity.question.*
import com.gorden.dayexam.executor.AppExecutors

@Database(
        entities = [DContext::class, Course::class, Book::class, Paper::class, Question::class,
                StudyStatus::class, Content::class, Element::class,
                        StudyRecord::class, Config::class],
        version = 1
)
@TypeConverters(
        DateConverter::class
)
abstract class AppDatabase : RoomDatabase() {

        abstract fun dContextDao(): DContextDao
        abstract fun courseDao(): CourseDao
        abstract fun bookDao(): BookDao
        abstract fun paperDao(): PaperDao
        abstract fun questionDao(): QuestionDao
        abstract fun studyStatusDao(): StudyStatusDao
        abstract fun elementDao(): ElementDao
        abstract fun contentDao(): ContentDao
        abstract fun studyRecordDao(): StudyRecordDao
        abstract fun configDao(): ConfigDao

        val isDatabaseCreated = MutableLiveData<Boolean>()
        val isDatabaseOpened = MutableLiveData<Boolean>()

        private fun setDatabaseCreated() {
                isDatabaseCreated.postValue(true)
        }

        private fun setDatabaseOpened() {
                isDatabaseOpened.postValue(true)
        }

        companion object {
                const val DATABASE_NAME = "exam-db"
```

```
        private var sInstance: AppDatabase? = null
        fun getInstance(
            context: Context,
            executors: AppExecutors
        ): AppDatabase {
            if (sInstance == null) {
                synchronized(AppDatabase::class.java) {
                    if (sInstance == null) {
                        sInstance = buildDatabase(context.applicationContext, executors)
                    }
                }
            }
            return sInstance!!
        }

        /**
         * Build the database. [Builder.build] only sets up the database configuration and
         * creates a new instance of the database.
         * The SQLite database is only created when it's accessed for the first time.
         */
        private fun buildDatabase(
            appContext: Context,
            executors: AppExecutors
        ): AppDatabase {
            return Room.databaseBuilder(appContext, AppDatabase::class.java, DATABASE_NAME)
                .addCallback(object : Callback() {
                    override fun onCreate(db: SupportSQLiteDatabase) {
                        super.onCreate(db)
                        getInstance(appContext, executors).setDatabaseCreated()
                    }

                    override fun onOpen(db: SupportSQLiteDatabase) {
                        super.onOpen(db)
                        getInstance(appContext, executors).setDatabaseOpened()
                    }
                })
                .addMigrations(MIGRATION_1_2)
                .build()
        }

        private val MIGRATION_1_2: Migration = object : Migration(1, 2) {
            override fun migrate(database: SupportSQLiteDatabase) {
            }
        }
    }
}
package com.gorden.dayexam.db

class DefaultData {
    companion object {
        const val DEFAULT_IMAGE = "default_data_image_xiahoudun_jpeg"

        const val FILL_IN_BODY_1 =
            """
下面是一个填空题的格式样例，填空题只有题干和答案两部分，这两部分都可以是图文混排的格式，将 word 中，将您的填空题编辑成分隔符内的格式就可以正常解析啦，注意填空和答案的前面要加&&哦

>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
&&填空
夏侯惇能抗能打，并且其每一个技能，都是开团的利器，能给对手带来很强的压迫感。当他扔出大刀，就意味着一场战斗的开始。一旦夏侯惇和他的大刀杀入敌阵，将会对敌人造成毁灭性打击。即便夏侯惇伤害在敌阵中遭遇围攻，他的护盾也能够抵御部分伤害。一级时夏侯惇的护盾能抵抗_____点伤害
&&答案
800
            """
        const val FILL_IN_BODY_2 = DEFAULT_IMAGE
        const val FILL_IN_BODY_3 = "<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<"
        const val FILL_IN_ANSWER = "800 点"
```

```
        const val TRUE_FALSE_BODY_1 =
            """
```

下面是一个判断题的格式样例，判断题只有题干和答案两部分，这两部分都可以是图文混排的格式，将 word 中，将您的判断题编辑成分隔符内的格式就可以正常解析啦，注意判断和答案的前面要加&&哦，并且答案的内容只能是对或者错哦

```
>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
&&判断
夏侯惇能抗能打，并且其每一个技能，都是开团的利器，能给对手带来很强的压迫感。即便夏侯惇伤害在敌阵中遭遇围攻，他的护盾也能够抵御部分伤害。一级时夏侯惇的护盾能抵抗 800 点伤害，对么
&&答案
错
            """
        const val TRUE_FALSE_BODY_2 = DEFAULT_IMAGE
        const val TRUE_FALSE_BODY_3 = "<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<"
        const val TRUE_FALSE_ANSWER = "错"

        const val SINGLE_CHOICE_BODY_1 =
            """
```

下面是一个单选题的格式样例，单选题有题干、选项、答案三部分，这三部分都可以是图文混排的格式，将 word 中，将您的单选题编辑成分隔符内的格式就可以正常解析啦，注意单选和答案的前面要加&&哦，并且答案的内容只能是一个大写英文字符哦，选项内也可以插入图片

```
>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
&&单选
夏侯惇能抗能打，并且其每一个技能，都是开团的利器，能给对手带来很强的压迫感。即便夏侯惇伤害在敌阵中遭遇围攻，他的护盾也能够抵御部分伤害。一级时夏侯惇的护盾能抵抗多少点伤害
&&选项
500 点
&&选项
600 点
&&选项
700 点
            """
        const val SINGLE_CHOICE_BODY_2 = DEFAULT_IMAGE
        const val SINGLE_CHOICE_BODY_3 =
            """
&&选项
800 点
&&答案
B
            """
        const val SINGLE_CHOICE_BODY_4 = "<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<"
        const val SINGLE_CHOICE_OPTION_1 = "500 点"
        const val SINGLE_CHOICE_OPTION_2 = "600 点"
        const val SINGLE_CHOICE_OPTION_3 = "700 点"
        const val SINGLE_CHOICE_OPTION_3_IMAGE = DEFAULT_IMAGE
        const val SINGLE_CHOICE_OPTION_4 = "800 点"
        const val SINGLE_CHOICE_ANSWER = "B"

        const val MULTI_CHOICE_BODY_1 =
            """
```

下面是一个多选题的格式样例，多选题有题干、选项、答案三部分，这两部分都可以是图文混排的格式，将 word 中，将您的多选题编辑成分隔符内的格式就可以正常解析啦，注意多选和答案的前面要加&&哦，并且答案的内容只能是一个或多个大写英文字符哦，选项内也可以插入图片

```
>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
&&多选
夏侯惇能抗能打，并且其每一个技能，都是开团的利器，能给对手带来很强的压迫感。即便夏侯惇伤害在敌阵中遭遇围攻，他的护盾也能够抵御部分伤害。下面那个装备适合夏侯惇呢？
&&选项
梦魇
&&选项
虚无法杖
&&选项
冰心
            """
        const val MULTI_CHOICE_BODY_2 = DEFAULT_IMAGE
        const val MULTI_CHOICE_BODY_3 =
```

```
                """
&&选项
魔女斗篷
&&答案
B
                """
            const val MULTI_CHOICE_BODY_4 = "<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<"
            const val MULTI_CHOICE_OPTION_1 = "梦魇"
            const val MULTI_CHOICE_OPTION_2 = "虚无法杖"
            const val MULTI_CHOICE_OPTION_3 = "冰心"
            const val MULTI_CHOICE_OPTION_3_IMAGE = DEFAULT_IMAGE
            const val MULTI_CHOICE_OPTION_4 = "魔女斗篷"
            const val MULTI_CHOICE_ANSWER = "CD"


            const val ESSAY_BODY_1 =
                """
下面是一个问答题的格式样例，问答题只有题干和答案两部分，这两部分都可以是图文混排的格式，将 word 中，将您的问答题编辑成分隔符内的格式就可以正常解析啦，注意问答和答案的前面要加&&哦

>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
&&问答
夏侯惇是个什么样的英雄
&&答案
夏侯惇能抗能打，并且其每一个技能，都是开团的利器，能给对手带来很强的压迫感。当他扔出大刀，就意味着一场战斗的开始。一旦夏侯惇和他的大刀杀入敌阵，将会对敌人造成毁灭性打击。即便夏侯惇伤害在敌阵中遭遇围攻，他的护盾也能够抵御部分伤害。
                """
            const val ESSAY_BODY_2 = DEFAULT_IMAGE
            const val ESSAY_BODY_3 = "<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<"
            const val ESSAY_ANSWER =
                """
夏侯惇能抗能打，并且其每一个技能，都是开团的利器，能给对手带来很强的压迫感。当他扔出大刀，就意味着一场战斗的开始。一旦夏侯惇和他的大刀杀入敌阵，将会对敌人造成毁灭性打击。即便夏侯惇伤害在敌阵中遭遇围攻，他的护盾也能够抵御部分伤害。
                """
    }
}
package com.gorden.dayexam.db

import com.gorden.dayexam.ContextHolder
import com.gorden.dayexam.R
import com.gorden.dayexam.db.entity.*
import com.gorden.dayexam.db.entity.question.*
import com.gorden.dayexam.db.entity.question.Content.Companion.ANSWER_TYPE
import com.gorden.dayexam.db.entity.question.Content.Companion.BODY_TYPE
import com.gorden.dayexam.db.entity.question.Content.Companion.OPTION_TYPE
import com.gorden.dayexam.executor.AppExecutors
import com.gorden.dayexam.model.QuestionType

object DefaultDataGenerator {

    private lateinit var mDatabase: AppDatabase

    fun init(mDatabase: AppDatabase) {
        this.mDatabase = mDatabase
    }

    fun generate() {
        generateCourse("天天背题")
    }

    private fun generateCourse(title: String) {
        AppExecutors.diskIO().execute {
            mDatabase.runInTransaction {
                val course = Course(title, "", 1)
                val courseId = mDatabase.courseDao().insert(course)

                val book = Book("给用户的一封信", "", courseId.toInt(), 1)
                val bookId = mDatabase.bookDao().insert(book).toInt()
                val courseStatus = StudyStatus(CourseStatus, courseId.toInt(), bookId)
```

```
                mDatabase.studyStatusDao().insert(courseStatus)

                val letterPaper = Paper("用户亲启", "", bookId, 1)
                val letterPaperId = mDatabase.paperDao().insert(letterPaper).toInt()
                val letterCurQuestionId = insertALetterForUser(letterPaperId, 1)
                val letterPaperStatus = StudyStatus(PaperStatus, letterPaperId, letterCurQuestionId)
                mDatabase.studyStatusDao().insert(letterPaperStatus)

                val demoPaper = Paper("教程", "", bookId, 2)
                val demoPaperId = mDatabase.paperDao().insert(demoPaper).toInt()
                val demoCurQuestionId = insertFillInBlankQuestion(demoPaperId, 1)
                insertTrueOrFalseQuestion(demoPaperId, 2)
                insertSingleChoiceQuestion(demoPaperId, 3)
                insertMultiChoiceQuestion(demoPaperId, 4)
                insertEssayQuestion(demoPaperId, 5)
                val demoPaperStatus = StudyStatus(PaperStatus, demoPaperId, demoCurQuestionId)
                mDatabase.studyStatusDao().insert(demoPaperStatus)

                val bookStatus = StudyStatus(BookStatus, bookId, letterPaperId)
                mDatabase.studyStatusDao().insert(bookStatus)

                val recycleBin = generateRecycleBin()
                mDatabase.dContextDao().insert(DContext(
                    courseId.toInt(),
                    bookId,
                    letterPaperId,
                    letterCurQuestionId,
                    recycleBin[0],
                    recycleBin[1],
                    recycleBin[2])
                )
                mDatabase.configDao().insert(
                    Config(
                        rememberMode = false,
                        focusMode = false,
                        onlyFavorite = false,
                        sortByAccuracy = false
                    )
                )
            }
        }
    }

    private fun generateRecycleBin(): IntArray {
        val recycleBin = Course("废纸篓", "找回您删除的内容", Int.MAX_VALUE)
        recycleBin.isRecycleBin = true
        val recycleBinId = mDatabase.courseDao().insert(recycleBin).toInt()
        val book = Book("已删除的试卷", "", recycleBinId, 1)
        val bookId = mDatabase.bookDao().insert(book).toInt()
        val courseStatus = StudyStatus(CourseStatus, recycleBinId, bookId)
        mDatabase.studyStatusDao().insert(courseStatus)
        val paper = Paper("回收站说明", "", bookId, 1)
        val paperId = mDatabase.paperDao().insert(paper).toInt()
        val bookStatus = StudyStatus(BookStatus, bookId, paperId)
        mDatabase.studyStatusDao().insert(bookStatus)
        val questionId = insertRecycleBinFillInBlankQuestion(paperId, 1)
        val paperStatus = StudyStatus(PaperStatus, paperId, questionId)
        mDatabase.studyStatusDao().insert(paperStatus)
        return intArrayOf(recycleBinId, bookId, paperId)
    }

    private fun generatePaper(bookId: Int): List<Paper> {
        return listOf(
            Paper("用户亲启", "", bookId, 1),
            Paper("教程", "", bookId, 2)
        )
    }

    private fun insertALetterForUser(paperId: Int, order: Int): Int {
```

```
        val question = Question(paperId, QuestionType.FILL_BLANK, order)
        val questionId = mDatabase.questionDao().insert(question).toInt()
        val body = Content(questionId, BODY_TYPE)
        val bodyId = mDatabase.contentDao().insert(body).toInt()
        val bodyElement = Element(Element.TEXT, ContextHolder.application.resources.getString(R.string.letter_for_users), bodyId, 1)
        mDatabase.elementDao().insert(bodyElement)
        val answer = Content(questionId, ANSWER_TYPE)
        val answerId = mDatabase.contentDao().insert(answer).toInt()
        val answerElement = Element(Element.TEXT, "", answerId, 1)
        mDatabase.elementDao().insert(answerElement)
        return questionId
    }

    private fun insertFillInBlankQuestion(paperId: Int, order: Int): Int {
        val question = Question(paperId, QuestionType.FILL_BLANK, order)
        val questionId = mDatabase.questionDao().insert(question).toInt()
        val body = Content(questionId, BODY_TYPE)
        val bodyId = mDatabase.contentDao().insert(body).toInt()
        val bodyElement = Element(Element.TEXT, DefaultData.FILL_IN_BODY_1, bodyId, 1)
        mDatabase.elementDao().insert(bodyElement)

        val bodyElement1 = Element(Element.PICTURE, DefaultData.FILL_IN_BODY_2, bodyId, 2)
        mDatabase.elementDao().insert(bodyElement1)

        val bodyElement2 = Element(Element.TEXT, DefaultData.FILL_IN_BODY_3, bodyId, 2)
        mDatabase.elementDao().insert(bodyElement2)

        val answer = Content(questionId, ANSWER_TYPE)
        val answerId = mDatabase.contentDao().insert(answer).toInt()
        val answerElement = Element(Element.TEXT, DefaultData.FILL_IN_ANSWER, answerId, 1)
        mDatabase.elementDao().insert(answerElement)
        return questionId
    }

    private fun insertTrueOrFalseQuestion(paperId: Int, order: Int): Int {
        val question = Question(paperId, QuestionType.TRUE_FALSE, order)
        val questionId = mDatabase.questionDao().insert(question).toInt()
        val body = Content(questionId, BODY_TYPE)
        val bodyId = mDatabase.contentDao().insert(body).toInt()
        val bodyElement = Element(Element.TEXT, DefaultData.TRUE_FALSE_BODY_1, bodyId, 1)
        mDatabase.elementDao().insert(bodyElement).toInt()
        val bodyElement1 = Element(Element.PICTURE, DefaultData.TRUE_FALSE_BODY_2, bodyId, 2)
        mDatabase.elementDao().insert(bodyElement1).toInt()
        val bodyElement2 = Element(Element.TEXT, DefaultData.TRUE_FALSE_BODY_3, bodyId, 2)
        mDatabase.elementDao().insert(bodyElement2).toInt()
        val answer = Content(questionId, ANSWER_TYPE)
        val answerId = mDatabase.contentDao().insert(answer).toInt()
        val answerElement = Element(Element.TEXT, DefaultData.TRUE_FALSE_ANSWER, answerId, 1)
        mDatabase.elementDao().insert(answerElement)
        return questionId
    }

    private fun insertSingleChoiceQuestion(paperId: Int, order: Int): Int {
        val question = Question(paperId, QuestionType.SINGLE_CHOICE, order)
        val questionId = mDatabase.questionDao().insert(question).toInt()
        val body = Content(questionId, BODY_TYPE)
        val bodyId = mDatabase.contentDao().insert(body).toInt()
        val bodyElement1 = Element(Element.TEXT, DefaultData.SINGLE_CHOICE_BODY_1, bodyId, 1)
        mDatabase.elementDao().insert(bodyElement1)
        val bodyElement2 = Element(Element.PICTURE, DefaultData.SINGLE_CHOICE_BODY_2, bodyId, 2)
        mDatabase.elementDao().insert(bodyElement2)
        val bodyElement3 = Element(Element.TEXT, DefaultData.SINGLE_CHOICE_BODY_3, bodyId, 3)
        mDatabase.elementDao().insert(bodyElement3)
        val bodyElement4 = Element(Element.TEXT, DefaultData.SINGLE_CHOICE_BODY_4, bodyId, 3)
        mDatabase.elementDao().insert(bodyElement4)

        val optionAId = mDatabase.contentDao().insert(Content(questionId, OPTION_TYPE)).toInt()
        mDatabase.elementDao().insert(Element(Element.TEXT, DefaultData.SINGLE_CHOICE_OPTION_1, optionAId, 1))
```

```
        val optionBId = mDatabase.contentDao().insert(Content(questionId, OPTION_TYPE)).toInt()
        mDatabase.elementDao().insert(Element(Element.TEXT, DefaultData.SINGLE_CHOICE_OPTION_2, optionBId, 1))

        val optionCId = mDatabase.contentDao().insert(Content(questionId, OPTION_TYPE)).toInt()
        mDatabase.elementDao().insert(Element(Element.TEXT, DefaultData.SINGLE_CHOICE_OPTION_3, optionCId, 1))
        mDatabase.elementDao().insert(Element(Element.PICTURE, DefaultData.SINGLE_CHOICE_OPTION_3_IMAGE, optionCId, 2))

        val optionDId = mDatabase.contentDao().insert(Content(questionId, OPTION_TYPE)).toInt()
        mDatabase.elementDao().insert(Element(Element.TEXT, DefaultData.SINGLE_CHOICE_OPTION_4, optionDId, 4))

        val answer = Content(questionId, ANSWER_TYPE)
        val answerId = mDatabase.contentDao().insert(answer).toInt()
        val answerElement = Element(Element.TEXT, DefaultData.SINGLE_CHOICE_ANSWER, answerId, 1)
        mDatabase.elementDao().insert(answerElement)
        return questionId
    }

    private fun insertMultiChoiceQuestion(paperId: Int, order: Int): Int {
        val question = Question(paperId, QuestionType.MULTIPLE_CHOICE, order)
        val questionId = mDatabase.questionDao().insert(question).toInt()
        val body = Content(questionId, BODY_TYPE)
        val bodyId = mDatabase.contentDao().insert(body).toInt()
        val bodyElement1 = Element(Element.TEXT, DefaultData.MULTI_CHOICE_BODY_1, bodyId, 1)
        mDatabase.elementDao().insert(bodyElement1)
        val bodyElement2 = Element(Element.PICTURE, DefaultData.MULTI_CHOICE_BODY_2, bodyId, 2)
        mDatabase.elementDao().insert(bodyElement2)
        val bodyElement3 = Element(Element.TEXT, DefaultData.MULTI_CHOICE_BODY_3, bodyId, 3)
        mDatabase.elementDao().insert(bodyElement3)
        val bodyElement4 = Element(Element.TEXT, DefaultData.MULTI_CHOICE_BODY_4, bodyId, 3)
        mDatabase.elementDao().insert(bodyElement4)

        val optionAId = mDatabase.contentDao().insert(Content(questionId, OPTION_TYPE)).toInt()
        mDatabase.elementDao().insert(Element(Element.TEXT, DefaultData.MULTI_CHOICE_OPTION_1, optionAId, 1))

        val optionBId = mDatabase.contentDao().insert(Content(questionId, OPTION_TYPE)).toInt()
        mDatabase.elementDao().insert(Element(Element.TEXT, DefaultData.MULTI_CHOICE_OPTION_2, optionBId, 1))

        val optionCId = mDatabase.contentDao().insert(Content(questionId, OPTION_TYPE)).toInt()
        mDatabase.elementDao().insert(Element(Element.TEXT, DefaultData.MULTI_CHOICE_OPTION_3, optionCId, 1))
        mDatabase.elementDao().insert(Element(Element.PICTURE, DefaultData.MULTI_CHOICE_OPTION_3_IMAGE, optionCId, 2))

        val optionDId = mDatabase.contentDao().insert(Content(questionId, OPTION_TYPE)).toInt()
        mDatabase.elementDao().insert(Element(Element.TEXT, DefaultData.MULTI_CHOICE_OPTION_4, optionDId, 4))

        val answer = Content(questionId, ANSWER_TYPE)
        val answerId = mDatabase.contentDao().insert(answer).toInt()
        val answerElement = Element(Element.TEXT, DefaultData.MULTI_CHOICE_ANSWER, answerId, 1)
        mDatabase.elementDao().insert(answerElement)
        return questionId
    }

    private fun insertEssayQuestion(paperId: Int, order: Int): Int {
        val question = Question(paperId, QuestionType.ESSAY_QUESTION, order)
        val questionId = mDatabase.questionDao().insert(question).toInt()
        val body = Content(questionId, BODY_TYPE)
        val bodyId = mDatabase.contentDao().insert(body).toInt()
        val bodyElement1 = Element(Element.TEXT, DefaultData.ESSAY_BODY_1, bodyId, 1)
        mDatabase.elementDao().insert(bodyElement1).toInt()
        val bodyElement2 = Element(Element.PICTURE, DefaultData.ESSAY_BODY_2, bodyId, 2)
        mDatabase.elementDao().insert(bodyElement2).toInt()
        val bodyElement3 = Element(Element.TEXT, DefaultData.ESSAY_BODY_3, bodyId, 2)
        mDatabase.elementDao().insert(bodyElement3).toInt()
        val answer = Content(questionId, ANSWER_TYPE)
        val answerId = mDatabase.contentDao().insert(answer).toInt()
        val answerElement = Element(Element.TEXT, DefaultData.ESSAY_ANSWER, answerId, 1)
        mDatabase.elementDao().insert(answerElement)
        return questionId
    }
```

```
    private fun insertRecycleBinFillInBlankQuestion(paperId: Int, order: Int): Int {
        val question = Question(paperId, QuestionType.FILL_BLANK, order)
        val questionId = mDatabase.questionDao().insert(question).toInt()
        val body = Content(questionId, BODY_TYPE)
        val bodyId = mDatabase.contentDao().insert(body).toInt()
        val bodyElement = Element(Element.TEXT, "废纸篓说明", bodyId, 1)
        mDatabase.elementDao().insert(bodyElement).toInt()
        val answer = Content(questionId, ANSWER_TYPE)
        val answerId = mDatabase.contentDao().insert(answer).toInt()
        val answerElement = Element(Element.TEXT, " “哪里有麻烦，哪里就有她” ", answerId, 1)
        mDatabase.elementDao().insert(answerElement)
        return questionId
    }

}
package com.gorden.dayexam.executor

import android.graphics.Bitmap
import android.os.Handler
import android.os.Looper
import com.gorden.dayexam.parser.image.ImageCacheManager
import java.util.concurrent.Executor
import java.util.concurrent.Executors

/**
 * Global executor pools for the whole application.
 *
 *
 * Grouping tasks like this avoids the effects of task starvation (e.g. disk reads don't wait behind
 * webservice requests).
 */
object AppExecutors {

    private val mDiskIO = Executors.newSingleThreadExecutor()
    private val mNetworkIO = Executors.newFixedThreadPool(3)
    private val mMainThread = MainThreadExecutor()

    fun diskIO(): Executor {
        return mDiskIO
    }

    fun networkIO(): Executor {
        return mNetworkIO
    }

    fun mainThread(): MainThreadExecutor {
        return mMainThread
    }

    class MainThreadExecutor : Executor {
        private val mainThreadHandler = Handler(Looper.getMainLooper())
        override fun execute(command: Runnable) {
            mainThreadHandler.post(command)
        }

        fun executeDelay(command: Runnable, delayTime: Long) {
            mainThreadHandler.postDelayed(command, delayTime)
        }

        fun removeCommand(command: Runnable) {
            mainThreadHandler.removeCallbacks(command)
        }
    }
}
package com.gorden.dayexam.model

class QuestionType {
    companion object {
        const val ERROR_TYPE = 0
```

```
        const val FILL_BLANK = 1
        const val TRUE_FALSE = 2
        const val SINGLE_CHOICE = 3
        const val MULTIPLE_CHOICE = 4
        const val ESSAY_QUESTION = 5
    }
}
package com.gorden.dayexam.parser.image

import android.graphics.Bitmap
import android.graphics.BitmapFactory
import com.gorden.dayexam.ContextHolder
import com.gorden.dayexam.executor.AppExecutors
import com.jakewharton.disklrucache.DiskLruCache
import java.io.*
import java.lang.Exception

object ImageCacheManager {

    const val PARSED_IMAGE_FOLDER_NAME = "parsed_image"
    private const val MAX_CACHE_SIZE = 10 * 1024 * 1024 * 1024L
    private const val VALUE_COUNT = 1
    private val CACHE_PARENT_FOLDER = ContextHolder.application.cacheDir.path + File.separator + PARSED_IMAGE_FOLDER_NAME

    private var diskLruCache: DiskLruCache? = null
    private var curCacheFolder: String = ""

    init {
        if (!File(CACHE_PARENT_FOLDER).exists()) {
            File(CACHE_PARENT_FOLDER).mkdir()
        }
    }

    // 试卷切换的时候需要更新
    fun setCacheFolder(bookId: String) {
        curCacheFolder = CACHE_PARENT_FOLDER + File.separator + bookId
        diskLruCache?.close()
        diskLruCache = DiskLruCache.open(
            File(curCacheFolder),
            0,
            VALUE_COUNT,
            MAX_CACHE_SIZE)
    }

    private fun getBitmap(imageUrl: String): Bitmap? {
        return performGet(imageUrl)
    }

    fun getImageFile(imageUrl: String): File {
        val imagePath = curCacheFolder + File.separator + imageUrl + ".0"
        return File(imagePath)
    }

    fun save(fileName: String, data: ByteArray) {
        if (diskLruCache?.get(fileName) != null) {
            return
        }
        performSave(fileName, data)
    }

    fun delete(fileName: String) {
        try {
            diskLruCache?.remove(fileName)
        } catch (e: IOException) {

        }
    }

    fun getAsync(url: String, callback: ImageLoaderCallback) {
```

```
// 后 30 页
package com.gorden.dayexam.ui.action

interface Action {
    fun start()
}
package com.gorden.dayexam.ui.action

import android.content.Context
import com.gorden.dayexam.R
import com.gorden.dayexam.repository.DataRepository
import com.gorden.dayexam.ui.dialog.EditTextDialog

class CreateBookAction(val context: Context, val curCourseId: Int): Action {
    override fun start() {
        EditTextDialog(context,
            context.resources.getString(R.string.dialog_create_book_title),
            context.resources.getString(R.string.dialog_create_book_subTitle),
            "",
            context.resources.getString(R.string.dialog_create_book_hint),
            editCallBack = object : EditTextDialog.EditCallBack {
                override fun onConfirmContent(dialog: EditTextDialog, content: String, subContent: String) {
                    if (curCourseId != 0) {
                        DataRepository.insertBook(content, curCourseId)
                        DataRepository.increaseContentVersion()
                    }
                }
            }).show()
    }
}
package com.gorden.dayexam.ui.action

import android.content.Context
import android.widget.Toast
import com.gorden.dayexam.R
import com.gorden.dayexam.repository.DataRepository
import com.gorden.dayexam.ui.dialog.EditTextDialog

class CreatePaperAction(val context: Context, val bookId: Int): Action {
    override fun start() {
        EditTextDialog(context,
            context.resources.getString(R.string.dialog_create_paper_title),
            context.resources.getString(R.string.dialog_create_paper_subTitle),
            "",
            context.resources.getString(R.string.dialog_create_paper_hint),
            editCallBack = object : EditTextDialog.EditCallBack {
                override fun onConfirmContent(dialog: EditTextDialog, content: String, subContent: String) {
                    if (content.isNotEmpty()) {
                        DataRepository.insertPaper(content, "", bookId)
                        DataRepository.increaseContentVersion()
                    } else {
                        val msg = context.resources.getString(R.string.create_paper_empty_title_msg)
                        Toast.makeText(context, msg, Toast.LENGTH_SHORT).show()
                    }
                }
            }).show()
    }
}
package com.gorden.dayexam.ui.action

import android.content.Context
import com.gorden.dayexam.R
import com.gorden.dayexam.db.entity.Book
import com.gorden.dayexam.repository.DataRepository
import com.gorden.dayexam.ui.dialog.EditTextDialog

class DeleteBookAction(val context: Context, val book: Book): Action {
    override fun start() {
        EditTextDialog(context,
```

```
                context.resources.getString(R.string.dialog_delete_book_title),
                "确定要删除" + book.title + "么?(可从废纸篓找回)",
                editCallBack = object : EditTextDialog.EditCallBack {
                    override fun onConfirmContent(dialog: EditTextDialog, content: String, subContent: String) {
                        DataRepository.deleteBook(book)
                        DataRepository.increaseContentVersion()
                    }
                }).show()
        }
}
package com.gorden.dayexam.ui.action

import android.content.Context
import android.widget.Toast
import com.gorden.dayexam.repository.DataRepository
import com.gorden.dayexam.repository.DeleteQuestionCallback

class DeleteCurrentQuestionAction(val context: Context): Action {
    override fun start() {
        DataRepository.deleteCurrentQuestion(object : DeleteQuestionCallback {
            override fun onFinished(success: Boolean, msg: String) {
                Toast.makeText(context, msg, Toast.LENGTH_SHORT).show()
                if (success) {
                    DataRepository.increaseContentVersion()
                }
            }
        })
    }
}
package com.gorden.dayexam.ui.action

import android.content.Context
import com.gorden.dayexam.R
import com.gorden.dayexam.db.entity.Paper
import com.gorden.dayexam.repository.DataRepository
import com.gorden.dayexam.repository.IsInRecycleBinCallback
import com.gorden.dayexam.ui.dialog.EditTextDialog

class DeletePaperAction(val context: Context, val paper: Paper): Action {
    override fun start() {
        DataRepository.isInRecycleBin(paper, object : IsInRecycleBinCallback {
            override fun onFinished(isInRecycleBin: Boolean) {
                if (isInRecycleBin) {
                    performDeleteRecyclePaper(context, paper)
                } else {
                    performDeleteCommonPaper(context, paper)
                }
            }
        })
    }

    private fun performDeleteCommonPaper(context: Context, paper: Paper) {
        EditTextDialog(context,
            context.resources.getString(R.string.dialog_delete_paper_title),
            "确定要删除" + paper.title + "么?(可从废纸篓找回)",
            editCallBack = object : EditTextDialog.EditCallBack {
                override fun onConfirmContent(dialog: EditTextDialog, content: String, subContent: String) {
                    DataRepository.deletePaper(paper, false)
                    DataRepository.increaseContentVersion()
                }
            }).show()
    }

    private fun performDeleteRecyclePaper(context: Context, paper: Paper) {
        EditTextDialog(context,
            context.resources.getString(R.string.dialog_delete_paper_title),
            "确定要彻底删除" + paper.title + "么?(无法找回)",
            editCallBack = object : EditTextDialog.EditCallBack {
                override fun onConfirmContent(dialog: EditTextDialog, content: String, subContent: String) {
```

```
                        DataRepository.deletePaper(paper, true)
                        DataRepository.increaseContentVersion()
                    }
                }).show()
    }

}
package com.gorden.dayexam.ui.action

import android.content.Context
import com.gorden.dayexam.R
import com.gorden.dayexam.db.entity.Book
import com.gorden.dayexam.repository.DataRepository
import com.gorden.dayexam.ui.dialog.EditTextDialog

class EditBookAction(val context: Context, val book: Book): Action {
    override fun start() {
        EditTextDialog(context,
            context.resources.getString(R.string.dialog_edit_book_title),
            context.resources.getString(R.string.dialog_edit_book_subTitle),
            book.title,
            "",
            editCallBack = object : EditTextDialog.EditCallBack {
                override fun onConfirmContent(dialog: EditTextDialog, content: String, subContent: String) {
                    if (content.isNotEmpty()) {
                        book.title = content
                        DataRepository.updateBook(book)
                        DataRepository.increaseContentVersion()
                    }
                }
            }).show()
    }
}
package com.gorden.dayexam.ui.action

import android.content.Context
import com.gorden.dayexam.R
import com.gorden.dayexam.db.entity.Paper
import com.gorden.dayexam.repository.DataRepository
import com.gorden.dayexam.ui.dialog.EditTextDialog

class EditPaperAction(val context: Context, val paper: Paper): Action {
    override fun start() {
        EditTextDialog(context,
            context.resources.getString(R.string.dialog_edit_paper_title),
            context.resources.getString(R.string.dialog_edit_paper_subTitle),
            paper.title,
            context.resources.getString(R.string.dialog_create_book_hint),
            editCallBack = object : EditTextDialog.EditCallBack {
                override fun onConfirmContent(dialog: EditTextDialog, content: String, subContent: String) {
                    paper.title = content
                    DataRepository.updatePaper(paper)
                    DataRepository.increaseContentVersion()
                }
            }).show()
    }
}
package com.gorden.dayexam.ui.action

import android.content.Context
import com.gorden.dayexam.db.entity.question.Element
import com.gorden.dayexam.ui.dialog.element.EditElementsDialog

class EditQuestionContentAction(val context: Context, val elements: List<Element>): Action {
    override fun start() {
        if (elements.isNotEmpty()) {
            EditElementsDialog(
                context,
                elements[0].parentId,
```

```
                    elements
                ).show()
            }
        }
}
package com.gorden.dayexam.ui.action

import android.content.Context
import android.widget.Toast
import androidx.appcompat.app.AlertDialog
import androidx.lifecycle.LifecycleOwner
import androidx.lifecycle.LiveData
import androidx.lifecycle.MutableLiveData
import com.gorden.dayexam.R
import com.gorden.dayexam.db.entity.Book
import com.gorden.dayexam.db.entity.Course
import com.gorden.dayexam.db.entity.Paper
import com.gorden.dayexam.repository.DataRepository

class MovePaperAction(private val owner: LifecycleOwner, val context: Context, val paperId: Int): Action {

    private val courses = MutableLiveData<List<Course>>()
    private var books: LiveData<List<Book>>? = null
    private var papers: LiveData<List<Paper>>? = null

    override fun start() {
        selectCourse()
    }

    private fun selectCourse() {
        courses.observe(owner,
            { it ->
                val resources = context.resources
                if (it.isEmpty()) {
                    Toast.makeText(
                        context,
                        resources.getString(R.string.no_course_found_msg),
                        Toast.LENGTH_SHORT
                    ).show()
                } else {
                    val titles = it.map {
                        it.title
                    }
                    AlertDialog.Builder(context, R.style.MyAlertDialogTheme)
                        .setTitle(resources.getString(R.string.select_target_course))
                        .setItems(titles.toTypedArray()
                        ) { p0, p1 ->
                            selectBook(it[p1])
                            p0.dismiss()
                        }.show()

                }
                courses.removeObservers(owner)
            }
        )
        DataRepository.getAllCourseExcludeRecycleBin(courses)
    }

    private fun selectBook(course: Course) {
        books = DataRepository.getAllBooks(course.id)
        books?.observe(
            owner, { it ->
                val resources = context.resources
                if (it.isEmpty()) {
                    Toast.makeText(
                        context,
                        resources.getString(R.string.no_book_found_msg),
                        Toast.LENGTH_SHORT
                    ).show()
```

```
                } else {
                    val titles = it.map {
                        it.title
                    }
                    AlertDialog.Builder(context, R.style.MyAlertDialogTheme)
                        .setTitle(resources.getString(R.string.select_target_book))
                        .setItems(titles.toTypedArray()
                        ) { p0, p1 ->
                            val book = it[p1]
                            selectPaper(book)
                            p0.dismiss()
                        }.show()

                }
                books?.removeObservers(owner)
            }
        )
    }

    private fun selectPaper(book: Book) {
        papers = DataRepository.getPapersByBookId(book.id)
        papers?.observe(
            owner, { it ->
                val resources = context.resources
                if (it.isEmpty()) {
                    Toast.makeText(
                        context,
                        resources.getString(R.string.no_paper_found_msg),
                        Toast.LENGTH_SHORT
                    ).show()
                } else {
                    val titles = it.map {
                        it.title
                    }
                    AlertDialog.Builder(context, R.style.MyAlertDialogTheme)
                        .setTitle(resources.getString(R.string.select_target_paper))
                        .setItems(titles.toTypedArray()
                        ) { p0, p1 ->
                            val paper = it[p1]
                            DataRepository.movePaper(paperId, paper.id)
                            DataRepository.increaseContentVersion()
                            p0.dismiss()
                        }.show()

                }
                papers?.removeObservers(owner)
            }
        )
    }
}
package com.gorden.dayexam.ui.action

import android.app.Activity
import android.app.AlarmManager
import android.app.PendingIntent
import android.content.Context
import android.content.Intent
import android.widget.Toast
import androidx.appcompat.app.AlertDialog
import com.gorden.dayexam.ContextHolder
import com.gorden.dayexam.MainActivity
import com.gorden.dayexam.R
import com.gorden.dayexam.backup.BackupManager
import com.gorden.dayexam.db.AppDatabase
import com.gorden.dayexam.executor.AppExecutors
import com.gorden.dayexam.ui.dialog.EditTextDialog
import java.io.File
import kotlin.system.exitProcess
```

```
class RecoverBackupAction(val context: Context): Action {

    override fun start() {
        selectCourse()
    }

    private fun selectCourse() {
        AppExecutors.diskIO().execute {
            val backups = BackupManager.getAllBackupFiles()
            AppExecutors.mainThread().execute {
                val titles = backups.map {
                    it.nameWithoutExtension
                }
                AlertDialog.Builder(context, R.style.MyAlertDialogTheme)
                    .setTitle(context.resources.getString(R.string.please_select_backup))
                    .setItems(
                        titles.toTypedArray()
                    ) { p0, p1 ->
                        confirm(backups[p1])
                        p0.dismiss()
                    }.show()
            }
        }
    }

    private fun confirm(backup: File) {
        EditTextDialog(context,
            context.resources.getString(R.string.confirm_recover),
            "确定要恢复" + backup.nameWithoutExtension + "的备份么,恢复后会自动重启",
            editCallBack = object : EditTextDialog.EditCallBack {
                override fun onConfirmContent(
                    dialog: EditTextDialog,
                    content: String,
                    subContent: String
                ) {
                    BackupManager.recover(backup, object : BackupManager.BackupListener {
                        override fun onRecoverEnd(success: Boolean, msg: String) {
                            if (success) {
                                System.exit(0)
                            } else {
                                Toast.makeText(
                                    ContextHolder.application,
                                    msg,
                                    Toast.LENGTH_SHORT
                                ).show()
                            }
                        }

                    })
                }
            }).show()
    }
}
package com.gorden.dayexam.ui.action

import android.content.Context
import android.graphics.Bitmap
import android.graphics.Canvas
import android.provider.MediaStore
import android.util.Log
import android.widget.ScrollView
import android.widget.Toast
import com.gorden.dayexam.MainActivity
import com.gorden.dayexam.R
import java.io.ByteArrayOutputStream
import java.util.*

class ScreenShotHomeQuestionAction(val context: Context): Action {
    override fun start() {
```

```
            if (context is MainActivity) {
                val decorView = context.window.decorView
                val scrollView = decorView.findViewByld<ScrollView>(R.id.question_content_scrollview)
                val bitMap = getScrollViewBitmap(scrollView)
                if (bitMap != null) {
                    saveImage(context, bitMap)
                }
                Toast.makeText(
                    context,
                    context.resources.getString(R.string.screen_shot_question_success),
                    Toast.LENGTH_SHORT
                ).
                show()
                return
            }
            Toast.makeText(
                context,
                context.resources.getString(R.string.screen_shot_question_failed),
                Toast.LENGTH_SHORT
            ).
            show()
        }

        private fun saveImage(context: Context, image: Bitmap) {
            val bytes = ByteArrayOutputStream()
            image.compress(Bitmap.CompressFormat.JPEG, 100, bytes)
            MediaStore.Images.Media.insertImage(
                context.contentResolver,
                image,
                "dayexam" + System.currentTimeMillis(),
                null
            )
        }

        /**
         * 截取 scrollview 的屏幕
         */
        fun getScrollViewBitmap(scrollView: ScrollView): Bitmap? {
            var h = 0
            // 获取 listView 实际高度
            for (i in 0 until scrollView.childCount) {
                h += scrollView.getChildAt(i).height
            }
            Log.d("TAG", "实际高度:$h")
            Log.d("TAG", "  高度:" + scrollView.height)
            // 创建对应大小的 bitmap
            val bitmap: Bitmap = Bitmap.createBitmap(
                scrollView.width, h,
                Bitmap.Config.ARGB_8888
            )
            val canvas = Canvas(bitmap)
            scrollView.draw(canvas)
            return bitmap
        }
    }
    package com.gorden.dayexam.ui.book.question

    import android.annotation.SuppressLint
    import android.view.LayoutInflater
    import android.view.ViewGroup
    import androidx.recyclerview.widget.ItemTouchHelper
    import androidx.recyclerview.widget.RecyclerView
    import com.gorden.dayexam.R
    import com.gorden.dayexam.db.entity.Book
    import com.gorden.dayexam.db.entity.Paper
    import com.gorden.dayexam.repository.model.PaperWithQuestion
    import com.gorden.dayexam.repository.DataRepository
    import com.gorden.dayexam.ui.book.DragCallback
    import java.util.*
```

```
class PaperAdapter: RecyclerView.Adapter<PaperViewHolder>(),
        DragCallback.OnItemTouchListener {
        private var papers = listOf<PaperWithQuestion>()
        var selectPosition = -1
        private var curBookId = -1
        private var book: Book? = null
        private var isRecycleBin: Boolean = false

        var isSortMode = false
        private var recyclerView: RecyclerView? = null
        private var itemTouchHelper: ItemTouchHelper? = null

        init {
                val touchCallback = DragCallback()
                touchCallback.listener = this
                itemTouchHelper = ItemTouchHelper(touchCallback)
        }

        override fun onAttachedToRecyclerView(recyclerView: RecyclerView) {
                super.onAttachedToRecyclerView(recyclerView)
                this.recyclerView = recyclerView
        }

        override fun onCreateViewHolder(parent: ViewGroup, viewType: Int): PaperViewHolder {
                val itemView = LayoutInflater.from(parent.context).inflate(R.layout.paper_item, parent, false)
                return PaperViewHolder(itemView)
        }

        override fun onBindViewHolder(holder: PaperViewHolder, position: Int) {
                val paperWithQuestion = papers[position]
                holder.onBind(paperWithQuestion, this.book!!, curBookId, this, isRecycleBin)
        }

        override fun getItemCount(): Int {
                return papers.size
        }

        @SuppressLint("NotifyDataSetChanged")
        fun setData(papers: List<PaperWithQuestion>, curPaperId: Int, curBookId: Int, book: Book, isRecycleBin: Boolean) {
                this.papers = papers
                this.isRecycleBin = isRecycleBin
                papers.forEachIndexed { index, paperWithQuestion ->
                        if (paperWithQuestion.paper.id == curPaperId) {
                                this.selectPosition = index
                        }
                }
                this.curBookId = curBookId
                this.book = book
                notifyDataSetChanged()
        }

        fun setSortMode() {
                isSortMode = true
                itemTouchHelper?.attachToRecyclerView(recyclerView)
                notifyDataSetChanged()
        }

        fun cancelSortMode() {
                isSortMode = false
                itemTouchHelper?.attachToRecyclerView(null)
                notifyDataSetChanged()
        }

        override fun onMove(fromPosition: Int, toPosition: Int) {
                papers[fromPosition].paper.position = toPosition + 1
                papers[toPosition].paper.position = fromPosition + 1
                Collections.swap(papers, fromPosition, toPosition)
                notifyItemMoved(fromPosition, toPosition)
```

```
        }

        override fun onSwiped(position: Int) {

        }

        override fun clearView() {
            val tPapers = mutableListOf<Paper>()
            papers.forEach {
                tPapers.add(it.paper)
            }
            DataRepository.updatePapers(tPapers)
            DataRepository.increaseContentVersion()
        }
}
package com.gorden.dayexam.ui.book.question

import android.annotation.SuppressLint
import android.graphics.drawable.ColorDrawable
import android.view.LayoutInflater
import android.view.MotionEvent
import android.view.View
import android.view.ViewGroup
import android.widget.PopupWindow
import android.widget.TextView
import androidx.recyclerview.widget.RecyclerView
import com.gorden.dayexam.ContextHolder
import com.gorden.dayexam.R
import com.gorden.dayexam.db.entity.Book
import com.gorden.dayexam.db.entity.Paper
import com.gorden.dayexam.repository.model.PaperWithQuestion
import com.gorden.dayexam.ui.EventKey
import com.gorden.dayexam.utils.ScreenUtils
import com.jeremyliao.liveeventbus.LiveEventBus
import java.lang.StringBuilder
import java.text.SimpleDateFormat

class PaperViewHolder(itemView: View): RecyclerView.ViewHolder(itemView)    {
        private val container:View = itemView.findViewById(R.id.paper_item_container)
        private val rippleContainer:View = itemView.findViewById(R.id.paper_ripple_item_container)
        private val title: TextView = itemView.findViewById(R.id.PaperTitle)
        private val desc: TextView = itemView.findViewById(R.id.paper_desc)
        private val record: TextView = itemView.findViewById(R.id.studyRecord)
        private val lastTouchDownXY = arrayOf(0f, 0f)

        @SuppressLint("ClickableViewAccessibility")
        fun onBind(paperWithQuestion: PaperWithQuestion, book: Book,curBookId: Int, adapter: PaperAdapter, isRecycleBin: Boolean) {
            val resources = itemView.context.resources
            if (adapter.isSortMode) {
                itemView.findViewById<View>(R.id.paper_drag_handle).visibility = View.VISIBLE
            } else {
                itemView.findViewById<View>(R.id.paper_drag_handle).visibility = View.GONE
            }
            title.text = paperWithQuestion.paper.title
            desc.text = generateDesc(paperWithQuestion)
            record.text = generateStudyRecordInfo(paperWithQuestion)
            if (adapterPosition == adapter.selectPosition && curBookId == book.id) {
                container.setBackgroundColor(resources.getColor(R.color.colorPrimaryDark))
            } else {
                container.setBackgroundColor(resources.getColor(R.color.colorPrimary))
            }
            rippleContainer.setOnClickListener {
                if (adapter.isSortMode) {
                    adapter.cancelSortMode()
                    return@setOnClickListener
                }
                adapter.selectPosition = position
                LiveEventBus.get(
                    EventKey.PAPER_CONTAINER_CLICKED,
```

```
                EventKey.PaperClickEventModel::class.java)
                .post(EventKey.PaperClickEventModel(book.id, paperWithQuestion.paper.id))
        }
        rippleContainer.setOnLongClickListener {
            if (adapter.isSortMode) {
                return@setOnLongClickListener true
            } else {
                popMenu(adapter, paperWithQuestion, isRecycleBin)
            }
            true
        }
        rippleContainer.setOnTouchListener { _, motionEvent ->
            if (motionEvent.actionMasked == MotionEvent.ACTION_DOWN) {
                this.lastTouchDownXY[0] = motionEvent.rawX
                this.lastTouchDownXY[1] = motionEvent.rawY
            }
            false
        }
    }

    private fun popMenu(adapter: PaperAdapter, paperWithQuestion: PaperWithQuestion, isRecycleBin: Boolean) {
        var menuLayoutId = R.layout.paper_item_menu_layout
        if (isRecycleBin) {
            menuLayoutId = R.layout.recycle_bin_paper_item_menu_layout
        }
        val resources = itemView.context.resources
        val menuView = LayoutInflater.from(itemView.context)
            .inflate(menuLayoutId, null)
        menuView.measure(ViewGroup.LayoutParams.WRAP_CONTENT, ViewGroup.LayoutParams.WRAP_CONTENT)
        val measureHeight = menuView.measuredHeight
        val popupWindow = PopupWindow(menuView, ViewGroup.LayoutParams.WRAP_CONTENT, measureHeight)
        popupWindow.isFocusable = true
        // 计算偏移位置
        val screenHeight = ScreenUtils.screenHeight()
        popupWindow.setBackgroundDrawable(ColorDrawable(resources.getColor(R.color.colorPrimary)))
        popupWindow.elevation = 200f
        if ((screenHeight - lastTouchDownXY[1]) < measureHeight) {
            popupWindow.showAsDropDown(itemView, lastTouchDownXY[0].toInt(),
                itemView.height / 2)
        } else {
            popupWindow.showAsDropDown(itemView, lastTouchDownXY[0].toInt(),
                -itemView.height / 2)
        }
        menuView.findViewById<View>(R.id.add_question_from_file)?.setOnClickListener {
            LiveEventBus.get(EventKey.PAPER_MENU_ADD_QUESTION_FROM_FILE, EventKey.QuestionAddEventModel::class.java)
                .post(EventKey.QuestionAddEventModel(paperWithQuestion.paper.bookId, paperWithQuestion.paper.id))
            popupWindow.dismiss()
        }
        menuView.findViewById<View>(R.id.edit_paper)?.setOnClickListener {
            LiveEventBus.get(EventKey.PAPER_MENU_EDIT_PAPER, Paper::class.java)
                .post(paperWithQuestion.paper)
            popupWindow.dismiss()
        }
        menuView.findViewById<View>(R.id.delete_paper)?.setOnClickListener {
            LiveEventBus.get(EventKey.PAPER_MENU_DELETE_PAPER, Paper::class.java)
                .post(paperWithQuestion.paper)
            popupWindow.dismiss()
        }
        menuView.findViewById<View>(R.id.sortByHand)?.setOnClickListener {
            adapter.setSortMode()
            popupWindow.dismiss()
        }
        menuView.findViewById<View>(R.id.move)?.setOnClickListener {
            LiveEventBus.get(EventKey.PAPER_MENU_MOVE_PAPER, Paper::class.java)
                .post(paperWithQuestion.paper)
            popupWindow.dismiss()
        }
    }
```

```
    private fun generateDesc(paperWithQuestion: PaperWithQuestion): String {
        val question = paperWithQuestion.question
        return if (question?.body == null || question.body.element.isEmpty()) {
            ContextHolder.application.resources.getString(R.string.none_question_desc)
        } else {
            question.body.element[0].content
        }
    }

    @SuppressLint("SimpleDateFormat")
    private fun generateStudyRecordInfo(paperWithQuestion: PaperWithQuestion): String {
        val result = StringBuilder()
        val resources = ContextHolder.application.resources
        val lastStudy = resources.getString(R.string.last_study_time)
        val total = resources.getString(R.string.total)
        val hasStudy = resources.getString(R.string.has_study)
        val shortQuestion = resources.getString(R.string.short_question)
        val count = resources.getString(R.string.count)
        val hasNotStart = resources.getString(R.string.has_not_start_study)
        if (paperWithQuestion.studyInfo.lastStudyDate != null) {
            result.append("$lastStudy ")
            val formatter = SimpleDateFormat("MM-dd HH:mm")
            result.append(formatter.format(paperWithQuestion.studyInfo.lastStudyDate))
            result.append(" ")
        } else {
            result.append("$hasNotStart ")
        }
        result.append(total)
        result.append(paperWithQuestion.questionCount)
        result.append("$shortQuestion ")
        result.append(hasStudy)
        result.append(paperWithQuestion.studyInfo.studyCount)
        result.append(count)
        return result.toString()
    }
}
package com.gorden.dayexam.ui.book

import android.annotation.SuppressLint
import android.view.LayoutInflater
import android.view.ViewGroup
import androidx.recyclerview.widget.RecyclerView
import com.gorden.dayexam.R
import com.gorden.dayexam.db.entity.Book
import com.gorden.dayexam.repository.model.BookWithPaper
import com.gorden.dayexam.repository.DataRepository
import java.util.*

class BooksAdapter: RecyclerView.Adapter<BookViewHolder>(), DragCallback.OnItemTouchListener {

    private var books = listOf<BookWithPaper>()
    private var curBookId = 0
    private var curPaperId = 0
    private var isRecycleBin = false
    private var showDragHandle = false

    private var allowUpdate = false

    override fun onCreateViewHolder(parent: ViewGroup, viewType: Int): BookViewHolder {
        val itemView = LayoutInflater.from(parent.context).inflate(R.layout.book_list_item, parent, false)
        return BookViewHolder(itemView)
    }

    override fun onBindViewHolder(holder: BookViewHolder, position: Int) {
        holder.setData(books[position], curBookId, curPaperId, showDragHandle, isRecycleBin)
    }

    override fun getItemCount(): Int {
        return books.size
```

```
    }

    @SuppressLint("NotifyDataSetChanged")
    fun setData(books: List<BookWithPaper>, curBookId: Int, curPaperId: Int, isRecycleBin: Boolean) {
        this.books = books
        this.curBookId = curBookId
        this.curPaperId = curPaperId
        this.isRecycleBin = isRecycleBin
        notifyDataSetChanged()
    }

    fun showDragHandle() {
        this.showDragHandle = true
        notifyDataSetChanged()
    }

    fun hideDragHandle() {
        this.showDragHandle = false
        notifyDataSetChanged()
    }

    override fun onMove(fromPosition: Int, toPosition: Int) {
        books[fromPosition].book.position = toPosition + 1
        books[toPosition].book.position = fromPosition + 1
        Collections.swap(books, fromPosition, toPosition)
        notifyItemMoved(fromPosition, toPosition)
        allowUpdate = true
    }

    override fun onSwiped(position: Int) {

    }

    override fun clearView() {
        if (allowUpdate) {
            val updateBooks = mutableListOf<Book>()
            books.forEach {
                updateBooks.add(it.book)
            }
            DataRepository.updateBooks(updateBooks)
        }
        allowUpdate = false
    }
}
package com.gorden.dayexam.ui.book

import android.annotation.SuppressLint
import android.content.Intent
import android.graphics.drawable.ColorDrawable
import android.os.Bundle
import android.view.*
import android.widget.ImageButton
import android.widget.PopupWindow
import android.widget.TextView
import androidx.fragment.app.Fragment
import androidx.lifecycle.ViewModelProvider
import androidx.recyclerview.widget.ItemTouchHelper
import androidx.recyclerview.widget.LinearLayoutManager
import androidx.recyclerview.widget.RecyclerView
import com.gorden.dayexam.MainActivity
import com.gorden.dayexam.R
import com.gorden.dayexam.db.entity.Book
import com.gorden.dayexam.db.entity.Paper
import com.gorden.dayexam.executor.AppExecutors
import com.gorden.dayexam.parser.BookParser
import com.gorden.dayexam.parser.image.ImageCacheManager
import com.gorden.dayexam.repository.DataRepository
import com.gorden.dayexam.ui.EventKey
import com.gorden.dayexam.ui.action.*
```

```
import com.jeremyliao.liveeventbus.LiveEventBus
import java.lang.Exception

class BooksFragment : Fragment() {

    private val requestCode = 1001

    private lateinit var bookViewModel: BookViewModel
    val adapter = BooksAdapter()
    private var curCourseId: Int = 0
    private var curBookId: Int = 0
    private var curPaperId: Int = 0
    private var isRecycleBin: Boolean = false

    private lateinit var moreMenu: ImageButton
    private lateinit var openSort: ImageButton
    private lateinit var openSearch: ImageButton
    private lateinit var courseTitle: TextView
    private lateinit var bookList: RecyclerView
    private lateinit var itemTouchHelper: ItemTouchHelper
    private var isDragMode = false

    private var targetPaperInfo: EventKey.QuestionAddEventModel? = null

    override fun onCreateView(
        inflater: LayoutInflater,
        container: ViewGroup?,
        savedInstanceState: Bundle?
    ): View? {
        val root = inflater.inflate(R.layout.fragment_book_list_layout, container, false)
        initBookList(root)
        initActionBar(root)
        return root
    }

    @SuppressLint("SetTextI18n")
    private fun initBookList(root: View) {
        bookList = root.findViewById(R.id.book_list)
        bookList.adapter = adapter
        bookList.layoutManager = LinearLayoutManager(this.context)

        val touchCallback = DragCallback()
        touchCallback.listener = adapter
        itemTouchHelper = ItemTouchHelper(touchCallback)

        bookViewModel = ViewModelProvider(this).get(BookViewModel::class.java)
        bookViewModel.getBookDetail().observe(viewLifecycleOwner, {
            if (it == null) {
                curCourseId = 0
                curBookId = 0
                curPaperId = 0
                adapter.setData(listOf(), curBookId, curPaperId, false)
                val bookString = context?.resources?.getString(R.string.book)
                courseTitle.text = bookString
            } else {
                curCourseId = it.courseId
                curBookId = it.bookId
                curPaperId = it.paperId
                isRecycleBin = it.isRecycleBin
                adapter.setData(it.books, curBookId, curPaperId, isRecycleBin)
                val bookString = context?.resources?.getString(R.string.book)
                courseTitle.text = "" + bookString + "(" + it.books.size + ")"
                resetActionButton(it.isRecycleBin)
            }
        })
        registerBookClickedEvent()
    }

    private fun initActionBar(root: View) {
```

```
        moreMenu = root.findViewById(R.id.moreMenu)
        moreMenu.setOnClickListener {
            showPopMenu()
        }
        courseTitle = root.findViewById(R.id.courseTitle)
        openSort = root.findViewById(R.id.openSort)
        openSort.setOnClickListener {
            isDragMode = if (isDragMode) {
                itemTouchHelper.attachToRecyclerView(null)
                adapter.hideDragHandle()
                false
            } else {
                itemTouchHelper.attachToRecyclerView(bookList)
                adapter.showDragHandle()
                true
            }
        }
        openSearch = root.findViewById(R.id.openSearch)
        openSearch.setOnClickListener {
            LiveEventBus.get(EventKey.SEARCH_CLICKED, Int::class.java)
                // 0 没有意义
                .post(0)
        }
    }

    private fun resetActionButton(isRecycleBin: Boolean) {
        openSort.visibility = if (isRecycleBin) View.GONE else View.VISIBLE
        moreMenu.visibility = if (isRecycleBin) View.GONE else View.VISIBLE
    }

    private fun showPopMenu() {
        val menuView = layoutInflater.inflate(R.layout.book_list_info_menu_layout, null)
        val popupWindow = PopupWindow(menuView, ViewGroup.LayoutParams.WRAP_CONTENT,
ViewGroup.LayoutParams.WRAP_CONTENT)
        popupWindow.isFocusable = true
        popupWindow.setBackgroundDrawable(ColorDrawable(resources.getColor(R.color.colorPrimary)))
        popupWindow.elevation = 200f
        popupWindow.showAsDropDown(moreMenu, 0, -moreMenu.height)
        val addBook = menuView.findViewById<View>(R.id.addBookContainer)
        addBook.setOnClickListener {
            createBook()
            popupWindow.dismiss()
        }
    }

    private fun createPaper(book: Book) {
        activity?.let {
            CreatePaperAction(it, book.id).start()
        }
    }

    private fun editPaper(paper: Paper) {
        activity?.let {
            EditPaperAction(it, paper).start()
        }
    }

    private fun deletePaper(paper: Paper) {
        activity?.let {
            DeletePaperAction(it, paper).start()
        }
    }

    private fun movePaper(paper: Paper) {
        MovePaperAction(this, requireActivity(), paper.id).start()
    }

    private fun createBook() {
        activity?.let {
```

```
            CreateBookAction(it, curCourseId).start()
        }
    }

    private fun deleteBook(book: Book) {
        activity?.let {
            DeleteBookAction(it, book).start()
        }
    }

    private fun editBook(book: Book) {
        activity?.let {
            EditBookAction(it, book).start()
        }
    }

    private fun toFileBrowser() {
        val intent = Intent(Intent.ACTION_GET_CONTENT)
        intent.type = "*/*"
        intent.putExtra(Intent.EXTRA_ALLOW_MULTIPLE, true)
        intent.addCategory(Intent.CATEGORY_OPENABLE)
        startActivityForResult(intent, requestCode, null)
    }

    override fun onActivityResult(requestCode: Int, resultCode: Int, data: Intent?) {
        if (requestCode == this.requestCode) {
            if (targetPaperInfo == null) {
                return
            }
            try {
                data?.data?.let {
                    LiveEventBus.get(EventKey.START_PROGRESS_BAR, Int::class.java).post(0)
                    AppExecutors.diskIO().execute {
                        val inputStream = context?.contentResolver?.openInputStream(it)
                        inputStream?.let { it1 ->
                            BookParser.parse(it1, targetPaperInfo?.bookId!!,
                                targetPaperInfo?.paperId!!
                            )
                            DataRepository.increaseContentVersion()
                        }
                        AppExecutors.mainThread().execute {
                            LiveEventBus.get(EventKey.END_PROGRESS_BAR, Int::class.java).post(0)
                        }
                    }
                }
            } catch (e: Exception) {
                // TODO 异常处理
            }
        }
    }

    private fun registerBookClickedEvent() {
        LiveEventBus
            .get(EventKey.PAPER_CONTAINER_CLICKED,
                EventKey.PaperClickEventModel::class.java)
            .observe(this, {
                    DataRepository.updateCourseStatus(
                        curCourseId,
                        it.bookId,
                        it.paperId)
                    ImageCacheManager.setCacheFolder(it.bookId.toString())
                (activity as MainActivity).closeDrawerLayout()
            })

        LiveEventBus.get(EventKey.PAPER_MENU_ADD_QUESTION_FROM_FILE, EventKey.QuestionAddEventModel::class.java)
            .observe(this, {
                targetPaperInfo = it
                toFileBrowser()
            })
```

```
            // paper 操作
            LiveEventBus.get(EventKey.PAPER_MENU_EDIT_PAPER, Paper::class.java)
                .observe(this, {
                    editPaper(it)
                })
            LiveEventBus.get(EventKey.PAPER_MENU_DELETE_PAPER, Paper::class.java)
                .observe(this, {
                    deletePaper(it)
                })
            // book 操作
            LiveEventBus.get(EventKey.CREATE_PAPER_CLICKED, Book::class.java)
                .observe(this, {
                    createPaper(it)
                })
            LiveEventBus.get(EventKey.DELETE_BOOK_CLICKED, Book::class.java)
                .observe(this, {
                    deleteBook(it)
                })
            LiveEventBus.get(EventKey.EDIT_BOOK_CLICKED, Book::class.java)
                .observe(this, {
                    editBook(it)
                })
            LiveEventBus.get(EventKey.PAPER_MENU_MOVE_PAPER, Paper::class.java)
                .observe(this, {
                    movePaper(it)
                })
        }

}
package com.gorden.dayexam.ui.book

import android.annotation.SuppressLint
import android.view.View
import android.widget.ImageButton
import android.widget.LinearLayout
import android.widget.TextView
import androidx.recyclerview.widget.DividerItemDecoration
import androidx.recyclerview.widget.LinearLayoutManager
import androidx.recyclerview.widget.RecyclerView
import com.gorden.dayexam.R
import com.gorden.dayexam.db.entity.Book
import com.gorden.dayexam.repository.model.BookWithPaper
import com.gorden.dayexam.ui.EventKey
import com.gorden.dayexam.ui.book.question.PaperAdapter
import com.jeremyliao.liveeventbus.LiveEventBus

@SuppressLint("UseCompatLoadingForDrawables")
class BookViewHolder(itemView: View): RecyclerView.ViewHolder(itemView) {
    private val titleContainer = itemView.findViewById<View>(R.id.bookInfoContainer)
    val title = itemView.findViewById<TextView>(R.id.bookTitle)!!
    private val paperList = itemView.findViewById<RecyclerView>(R.id.paperList)!!
    private val dragHandle = itemView.findViewById<ImageButton>(R.id.drag_handle)
    private val addPaper = itemView.findViewById<ImageButton>(R.id.addPaper)
    private val editBook = itemView.findViewById<ImageButton>(R.id.edit_paper)
    private val deleteBook = itemView.findViewById<ImageButton>(R.id.delete_paper)
    private var isDragMode = false
    private var isEditMode = false

    init {
        paperList.layoutManager = LinearLayoutManager(itemView.context)
        val adapter = PaperAdapter()
        paperList.adapter = adapter
        val divider = DividerItemDecoration(itemView.context, LinearLayout.VERTICAL)
        divider.setDrawable(itemView.context.resources.getDrawable(R.drawable.question_group_inset_recyclerview_divider, null))
        paperList.addItemDecoration(divider)
    }

    fun setData(data: BookWithPaper, curBookId: Int, curPaperId: Int, showDragHandle: Boolean, isRecycleBin: Boolean) {
        if (isRecycleBin) {
```

```
            setRecycleBinHolder(data, curBookId, curPaperId, isRecycleBin)
        } else {
            setCommonHolder(data, curBookId, curPaperId, showDragHandle, isRecycleBin)
        }
    }

    private fun setRecycleBinHolder(data: BookWithPaper, curBookId: Int, curPaperId: Int, isRecycleBin: Boolean) {
        addPaper.visibility = View.GONE
        title.text = data.book.title
        (paperList.adapter as PaperAdapter)
            .setData(data.papers, curPaperId, curBookId, data.book, isRecycleBin)
        titleContainer.setOnClickListener {
            switchPaperListVisibility()
        }
        titleContainer.isLongClickable = false
    }

    private fun setCommonHolder(data: BookWithPaper, curBookId: Int, curPaperId: Int, showDragHandle: Boolean, isRecycleBin: Boolean) {
        this.isDragMode = showDragHandle
        title.text = data.book.title
        if (showDragHandle) {
            setDragMode()
            cancelEditMode()
            hidePaperList()
        } else {
            cancelDragMode()
        }
        (paperList.adapter as PaperAdapter)
            .setData(data.papers, curPaperId, curBookId, data.book, isRecycleBin)
        titleContainer.setOnClickListener {
            if (isDragMode) {
                return@setOnClickListener
            }
            if (isEditMode) {
                cancelEditMode()
                return@setOnClickListener
            }
            switchPaperListVisibility()
        }
        titleContainer.setOnLongClickListener {
            if (isDragMode) {
                return@setOnLongClickListener true
            }
            if (isEditMode) {
                cancelEditMode()
                return@setOnLongClickListener true
            }
            setEditMode()
            true
        }
        addPaper.setOnClickListener {
            LiveEventBus.get(EventKey.CREATE_PAPER_CLICKED, Book::class.java)
                .post(data.book)
        }
        editBook.setOnClickListener {
            cancelEditMode()
            LiveEventBus.get(EventKey.EDIT_BOOK_CLICKED, Book::class.java)
                .post(data.book)
        }
        deleteBook.setOnClickListener {
            cancelEditMode()
            LiveEventBus.get(EventKey.DELETE_BOOK_CLICKED, Book::class.java)
                .post(data.book)
        }
    }

    private fun setDragMode() {
        dragHandle.visibility = View.VISIBLE
        addPaper.visibility = View.GONE
```

```
        }

        private fun cancelDragMode() {
            dragHandle.visibility = View.GONE
            addPaper.visibility = View.VISIBLE
        }

        private fun setEditMode() {
            editBook.visibility = View.VISIBLE
            deleteBook.visibility = View.VISIBLE
            isEditMode = true
        }

        private fun cancelEditMode() {
            editBook.visibility = View.GONE
            deleteBook.visibility = View.GONE
            isEditMode = false
        }

        private fun switchPaperListVisibility() {
            if (paperList.visibility == View.GONE) {
                paperList.visibility = View.VISIBLE
                itemView.findViewById<ImageButton>(R.id.expandIcon)
                    .setImageResource(R.drawable.ic_baseline_expand_more_24)
            } else {
                paperList.visibility = View.GONE
                itemView.findViewById<ImageButton>(R.id.expandIcon)
                    .setImageResource(R.drawable.ic_baseline_keyboard_arrow_right_24)
            }
        }

        private fun hidePaperList() {
            paperList.visibility = View.GONE
            itemView.findViewById<ImageButton>(R.id.expandIcon)
                .setImageResource(R.drawable.ic_baseline_keyboard_arrow_right_24)
        }
    }
    package com.gorden.dayexam.ui.book

    import android.app.Application
    import androidx.lifecycle.AndroidViewModel
    import androidx.lifecycle.LiveData
    import androidx.lifecycle.MutableLiveData
    import androidx.lifecycle.Transformations
    import com.gorden.dayexam.repository.DataRepository
    import com.gorden.dayexam.repository.model.BookWithPaper

    class BookViewModel(application: Application): AndroidViewModel(application) {

        private val dContext = DataRepository.getDContext()
        private val bookDetail = MutableLiveData<BookDetail>()

        private val currentBookDetail = Transformations.switchMap(dContext){
            it?.let {
                DataRepository.currentBookDetail(bookDetail)
                bookDetail
            }
        }

        fun getBookDetail(): LiveData<BookDetail> {
            return currentBookDetail
        }

    }

    data class BookDetail(
        var courseId: Int,
        var courseTitle: String,
        var bookId: Int,
```

```
    var paperId: Int,
    var isRecycleBin: Boolean,
    var books: List<BookWithPaper>
)
package com.gorden.dayexam.ui.book

import androidx.recyclerview.widget.ItemTouchHelper
import androidx.recyclerview.widget.RecyclerView

class DragCallback : ItemTouchHelper.Callback() {

    var listener: OnItemTouchListener? = null

    override fun getMovementFlags(
        recyclerView: RecyclerView,
        viewHolder: RecyclerView.ViewHolder
    ): Int {
        val dragFlag = ItemTouchHelper.UP or ItemTouchHelper.DOWN
        return makeMovementFlags(dragFlag, 0)
    }

    override fun onMove(
        recyclerView: RecyclerView,
        viewHolder: RecyclerView.ViewHolder,
        target: RecyclerView.ViewHolder
    ): Boolean {
        listener?.onMove(viewHolder.adapterPosition, target.adapterPosition)
        return true
    }

    override fun onSwiped(viewHolder: RecyclerView.ViewHolder, direction: Int) {
        listener?.onSwiped(viewHolder.adapterPosition)
    }

    override fun clearView(recyclerView: RecyclerView, viewHolder: RecyclerView.ViewHolder) {
        super.clearView(recyclerView, viewHolder)
        listener?.clearView()
    }

    override fun isItemViewSwipeEnabled(): Boolean {
        return true
    }

    /**
     * 移动交换数据的更新监听
     */
    interface OnItemTouchListener {
        //拖动 Item 时调用
        fun onMove(fromPosition: Int, toPosition: Int)

        //滑动 Item 时调用
        fun onSwiped(position: Int)

        fun clearView()
    }
}
package com.gorden.dayexam.ui.dialog.element

import android.app.Dialog
import android.content.Context
import android.graphics.Bitmap
import android.os.Bundle
import android.view.View
import android.view.ViewGroup
import androidx.constraintlayout.widget.ConstraintLayout
import androidx.recyclerview.widget.LinearLayoutManager
import androidx.recyclerview.widget.RecyclerView
import com.gorden.dayexam.R
import com.gorden.dayexam.db.entity.question.Element
```

```
import com.gorden.dayexam.executor.AppExecutors
import com.gorden.dayexam.parser.image.ImageCacheManager
import com.gorden.dayexam.repository.DataRepository
import com.gorden.dayexam.utils.BookUtils
import java.io.ByteArrayOutputStream
import java.io.IOException

class EditElementsDialog(
    context: Context
): Dialog(context) {

    private var elements: List<Element> = listOf()
    private var contentId = -1

    private lateinit var rootView: ConstraintLayout
    private lateinit var elementList: RecyclerView
    private lateinit var adapter: ElementAdapter

    constructor(
        context: Context,
        contentId: Int,
        elements: List<Element>
    ): this(context) {
        this.elements = elements
        this.contentId = contentId
    }

    private fun init() {
        setContentView(R.layout.dialog_edit_elements_layout)
        window?.setBackgroundDrawableResource(R.color.colorTransparent)
        window?.setDimAmount(0.8f)
        window?.setLayout(ViewGroup.LayoutParams.MATCH_PARENT, ViewGroup.LayoutParams.MATCH_PARENT)
        rootView = findViewById(R.id.edit_elements_container)
        findViewById<View>(R.id.done_edit_content).setOnClickListener {
            saveAllElement()
            dismiss()
        }
        elementList = findViewById(R.id.element_list)
        findViewById<View>(R.id.cancel_edit_content).setOnClickListener {
            dismiss()
        }
        adapter = ElementAdapter()
        elementList.adapter = adapter
        elementList.layoutManager = LinearLayoutManager(context)
        adapter.setData(contentId, elements)
    }

    private fun saveAllElement() {
        AppExecutors.diskIO().execute {
            var index = 0
            val result = adapter.getData().filter {
                // 删除被操作 delete 的图片文件
                if (it.element.elementType == Element.PICTURE && it.isDeleted) {
                    ImageCacheManager.delete(it.element.content)
                }
                // 过滤掉需要删除的 element
                !it.isDeleted
            }.map {
                when (it.element.elementType) {
                    Element.TEXT -> {
                        Element(
                            it.element.elementType,
                            it.newContent!!,
                            it.element.parentId,
                            ++index
                        )
                    }
                    Element.PICTURE -> {
                        if ((it.hasEdited && it.image != null)) {
```

```
                                    val fileName = BookUtils.generateImageName(
                                        "NewAdd" + ++index,
                                        System.currentTimeMillis()
                                    )
                                    try {
                                        val stream = ByteArrayOutputStream()
                                        it.image!!.compress(Bitmap.CompressFormat.PNG, 100, stream)
                                        val byteArray: ByteArray = stream.toByteArray()
                                        ImageCacheManager.save(fileName, byteArray)
                                        Element(
                                            it.element.elementType,
                                            fileName,
                                            it.element.parentId,
                                            index
                                        )
                                    } catch (e: IOException) {

                                    } finally {
                                        it.image!!.recycle()
                                    }
                                } else {
                                    it.element.position = ++index
                                    it.element
                                }
                            } else -> {

                            }
                        }
                    }
                    DataRepository.updateElementsByContentId(contentId, result as List<Element>)
                    DataRepository.increaseContentVersion()
                }
            }

            override fun onCreate(savedInstanceState: Bundle?) {
                super.onCreate(savedInstanceState)
                init()
            }

        }
        package com.gorden.dayexam.ui.dialog.element

        import android.view.View
        import androidx.recyclerview.widget.RecyclerView
        import com.gorden.dayexam.R

        class EditElementViewHolder(itemView: View): RecyclerView.ViewHolder(itemView) {
            val editCard: ElementEditCard = itemView.findViewById(R.id.edit_card)
        }
        package com.gorden.dayexam.ui.dialog.element

        import android.graphics.Bitmap
        import android.view.LayoutInflater
        import android.view.ViewGroup
        import android.widget.Toast
        import androidx.recyclerview.widget.RecyclerView
        import com.gorden.dayexam.R
        import com.gorden.dayexam.db.entity.question.Element
        import com.gorden.dayexam.ui.book.DragCallback

        class ElementAdapter: RecyclerView.Adapter<EditElementViewHolder>(),
            DragCallback.OnItemTouchListener  {

            private var data = mutableListOf<EditableElement>()
            private var contentId = -1
            var isEditing = false
            private var currentActionPosition = -1

            private val payLoadHide = "hide"
```

```
private val payLoadShow = "show"
private val toEditMode = "editMode"
private val toDeleteMode = "deleteMode"
private val toResetMode = "resetMode"

override fun getItemViewType(position: Int): Int {
    val element = this.data[position].element
    return element?.elementType ?: Element.TEXT
}

override fun onCreateViewHolder(parent: ViewGroup, viewType: Int): EditElementViewHolder {
    return if (viewType == Element.TEXT) {
        val itemView = LayoutInflater.from(parent.context)
            .inflate(R.layout.dialog_edit_text_element_item, parent, false)
        EditElementViewHolder(itemView)
    } else {
        val itemView = LayoutInflater.from(parent.context)
            .inflate(R.layout.dialog_edit_image_element_item, parent, false)
        EditElementViewHolder(itemView)
    }
}

override fun onBindViewHolder(holder: EditElementViewHolder, position: Int) {
    val context = holder.itemView.context
    val resources = context.resources
    holder.editCard.setElement(data[position], this, object : EditActionListener {
        override fun onStartEdit() {
            isEditing = true
        }
    })
    if (currentActionPosition == position) {
        holder.editCard.showAction()
    } else {
        holder.editCard.hideAction()
    }
    holder.editCard.setOnClickListener {
        if (isEditing) {
            Toast.makeText(holder.editCard.context,
                resources.getString(R.string.current_other_question_editing),
                Toast.LENGTH_SHORT).show()
            return@setOnClickListener
        }
        if (position == currentActionPosition) {
            holder.editCard.hideAction()
            currentActionPosition = -1
        } else {
            val oldActionPosition = currentActionPosition
            currentActionPosition = position
            notifyItemChanged(oldActionPosition, payLoadHide)
            notifyItemChanged(currentActionPosition, payLoadShow)
        }
    }
}

override fun onBindViewHolder(
    holder: EditElementViewHolder,
    position: Int,
    payloads: MutableList<Any>
) {
    if (payloads.isEmpty()) {
        onBindViewHolder(holder, position)
    } else {
        when (payloads[0].toString()) {
            payLoadHide -> {
                holder.editCard.hideAction()
            }
            payLoadShow -> {
                holder.editCard.showAction()
            }
```

```
                toEditMode -> {
                    holder.editCard.toEditMode()
                }
                toDeleteMode -> {
                    holder.editCard.toDeleteMode()
                }
                toResetMode -> {
                    holder.editCard.toResetMode()
                }
            }
        }
    }

    override fun getItemCount(): Int {
        return data.size
    }

    override fun onMove(fromPosition: Int, toPosition: Int) {

    }

    override fun onSwiped(position: Int) {

    }

    override fun clearView() {

    }

    fun getData(): List<EditableElement> {
        return data
    }

    fun setData(contentId: Int, elements: List<Element>) {
        this.contentId = contentId
        val editableElements = elements.map {
            EditableElement(it, false)
        }
        this.data.clear()
        this.data.addAll(editableElements)
        notifyDataSetChanged()
    }

    fun insertTextElementAfterCurrentPosition() {
        createEditableElement(Element.TEXT)?.let {
            this.data.add(currentActionPosition + 1, it)
            notifyItemInserted(currentActionPosition + 1)
            notifyItemRangeChanged(currentActionPosition + 1, this.data.size)
        }
    }

    fun insertImageElementAfterCurrentPosition() {
        createEditableElement(Element.PICTURE)?.let {
            this.data.add(currentActionPosition + 1, it)
            notifyItemInserted(currentActionPosition + 1)
            notifyItemRangeChanged(currentActionPosition + 1, this.data.size)
        }
    }

    fun currentItemEditMode() {
        notifyItemChanged(currentActionPosition, toEditMode)
    }

    fun currentItemDeleteMode() {
        notifyItemChanged(currentActionPosition, toDeleteMode)
    }

    fun currentItemResetMode() {
        notifyItemChanged(currentActionPosition, toResetMode)
```

```
    }

    private fun createEditableElement(elementType: Int): EditableElement? {
        if (currentActionPosition == -1) {
            return null
        }
        val currentElement = this.data[currentActionPosition].element ?: return null
        val editableElement = EditableElement(
            Element(
                elementType,
                "",
                currentElement.parentId,
                currentActionPosition + 1),
            true
        )
        editableElement.hasEdited = false
        editableElement.newContent = ""
        return editableElement
    }
}

data class EditableElement (
    var element: Element,
    val isNewAdd: Boolean) {
    var newContent: String? = element?.content
    var image: Bitmap? = null
    var hasEdited: Boolean = false
    var isDeleted: Boolean = false
}
package com.gorden.dayexam.ui.dialog.element

import android.content.Context
import android.util.AttributeSet
import android.widget.FrameLayout
import android.widget.ImageButton
import com.gorden.dayexam.R

abstract class ElementEditCard: FrameLayout {

    private lateinit var addTextBtn: ImageButton
    private lateinit var addImageBtn: ImageButton
    open lateinit var adapter: ElementAdapter

    constructor(context: Context) : super(context)

    constructor(context: Context, attrs: AttributeSet) : super(context, attrs)

    abstract fun setElement(editableElement: EditableElement, adapter: ElementAdapter, listener: EditActionListener)

    open fun setAction() {
        addTextBtn = findViewById(R.id.add_text_element_btn)
        addTextBtn.setOnClickListener {
            adapter.insertTextElementAfterCurrentPosition()
        }
        addImageBtn = findViewById(R.id.add_image_element_btn)
        addImageBtn.setOnClickListener {
            adapter.insertImageElementAfterCurrentPosition()
        }
    }

    abstract fun hideAction()

    abstract fun showAction()

    abstract fun toResetMode()

    abstract fun toEditMode()

    abstract fun toDeleteMode()
```

```
}
package com.gorden.dayexam.ui.dialog.element

import android.annotation.SuppressLint
import android.content.Context
import android.graphics.Bitmap
import android.graphics.drawable.Drawable
import android.util.AttributeSet
import android.view.LayoutInflater
import android.view.View
import android.view.ViewGroup
import android.widget.ImageButton
import android.widget.ImageView
import android.widget.LinearLayout
import com.bumptech.glide.Glide
import com.bumptech.glide.request.target.CustomTarget
import com.bumptech.glide.request.transition.Transition
import com.gorden.dayexam.R
import com.gorden.dayexam.parser.image.ImageCacheManager
import com.gorden.dayexam.ui.EventKey
import com.jeremyliao.liveeventbus.LiveEventBus

class ImageElementEditCard: ElementEditCard {

    private var imageContent: ImageView

    private var actionContainer: LinearLayout
    private var subActionContainer: LinearLayout
    private var editBtn: ImageButton
    private var resetBtn: ImageButton
    private var deleteBtn: ImageButton

    private var doneBtn: ImageButton
    private var cancelBtn: ImageButton

    private var listener: EditActionListener? = null

    private lateinit var editableElement: EditableElement

    constructor(context: Context) : super(context)

    constructor(context: Context, attrs: AttributeSet) : super(context, attrs)

    init {
        LayoutInflater.from(context).inflate(R.layout.image_element_edit_card_layout, this)
        imageContent = findViewById(R.id.image_element_content)

        actionContainer = findViewById(R.id.action_container)
        subActionContainer = findViewById(R.id.sub_action_container)
        editBtn = findViewById(R.id.edit_element_btn)
        resetBtn = findViewById(R.id.reset_element_btn)
        deleteBtn = findViewById(R.id.delete_element_btn)

        doneBtn = findViewById(R.id.done_edit_element_btn)
        cancelBtn = findViewById(R.id.cancel_element_edit)
        setAction()
    }

    @SuppressLint("CutPasteId")
    override fun setElement(
        editableElement: EditableElement,
        adapter: ElementAdapter,
        listener: EditActionListener
    ) {
        this.editableElement = editableElement
        this.listener = listener
        this.adapter = adapter
        imageContent.visibility = View.VISIBLE
```

```
        if (editableElement.newContent?.isNotBlank() == true) {
            loadCurImage()
        }
    }

    override fun setAction() {
        super.setAction()
        editBtn.setOnClickListener {
            LiveEventBus.get(EventKey.EDIT_SELECT_PHOTO_CLICK, PhotoSelectCallback::class.java)
                .post(object : PhotoSelectCallback {
                    override fun onSelect(image: Bitmap) {
                        editableElement.image = image
                        editableElement.hasEdited = true
                        adapter.currentItemEditMode()
                    }
                })
        }
        resetBtn.setOnClickListener {
            editableElement.newContent = editableElement.element?.content
            editableElement.image = null
            editableElement.hasEdited = false
            editableElement.isDeleted = false
            adapter.currentItemResetMode()
        }
        deleteBtn.setOnClickListener {
            editableElement.isDeleted = true
            editableElement.newContent = ""
            editableElement.image = null
            adapter.currentItemDeleteMode()
        }
        doneBtn.setOnClickListener {

        }
        cancelBtn.setOnClickListener {

        }
    }

    override fun hideAction() {
        actionContainer.visibility = GONE
    }

    override fun showAction() {
        actionContainer.visibility = VISIBLE
    }

    override fun toEditMode() {
        this.editableElement.image?.let { setImageBitmap(it) }
    }

    override fun toDeleteMode() {
        editBtn.visibility = View.GONE
        deleteBtn.visibility = View.GONE
        loadDefaultImage()
    }

    override fun toResetMode() {
        editBtn.visibility = View.VISIBLE
        deleteBtn.visibility = View.VISIBLE
        if (this.editableElement.newContent?.isEmpty() == true) {
            loadDefaultImage()
        } else {
            loadCurImage()
        }
    }

    private fun loadDefaultImage() {
        imageContent.scaleType = ImageView.ScaleType.CENTER
        val width = imageContent.measuredWidth
```

```
            val layoutParams = imageContent.layoutParams
            layoutParams.width = width
            layoutParams.height = ViewGroup.LayoutParams.WRAP_CONTENT
            imageContent.scaleType = ImageView.ScaleType.CENTER
            imageContent.layoutParams = layoutParams
            imageContent.setImageDrawable(context.getDrawable(R.drawable.edit_element_default_picture))
        }

        private fun loadCurImage() {
            var requestBuilder = Glide.with(context).asBitmap()
            val imageUrl = editableElement.newContent!!
            requestBuilder = if (imageUrl.startsWith("default_data_image")) {
                requestBuilder.load("file:///android_asset/image/$imageUrl")
            } else {
                val imageFile = ImageCacheManager.getImageFile(imageUrl)
                requestBuilder.load(imageFile)
            }
            requestBuilder
                .into(object : CustomTarget<Bitmap>() {
                    override fun onResourceReady(resource: Bitmap, transition: Transition<in Bitmap>?) {
                        // 这里返回的是像素值
                        imageContent.post {
                            setImageBitmap(resource)
                        }
                    }

                    override fun onLoadCleared(placeholder: Drawable?) {

                    }
                })
        }

        private fun setImageBitmap(bitmap: Bitmap) {
            val width = imageContent.measuredWidth
            val height = width * bitmap.height / bitmap.width
            val layoutParams = imageContent.layoutParams
            layoutParams.width = width
            layoutParams.height = height
            imageContent.scaleType = ImageView.ScaleType.FIT_XY
            imageContent.layoutParams = layoutParams
            imageContent.setImageBitmap(bitmap)
        }

        interface PhotoSelectCallback {
            fun onSelect(image: Bitmap)
        }

}
package com.gorden.dayexam.ui.dialog.element

import android.annotation.SuppressLint
import android.content.Context
import android.util.AttributeSet
import android.view.LayoutInflater
import android.view.View
import android.view.inputmethod.InputMethodManager
import android.widget.*
import com.gorden.dayexam.R

class TextElementEditCard: ElementEditCard {

        private var textContent: TextView
        private var editContent: EditText

        private var actionContainer: LinearLayout
        private var subActionContainer: LinearLayout
        private var editBtn: ImageButton
        private var resetBtn: ImageButton
        private var deleteBtn: ImageButton
```

```
private var doneBtn: ImageButton
private var cancelBtn: ImageButton

private var listener: EditActionListener? = null

private lateinit var editableElement: EditableElement

constructor(context: Context) : super(context)

constructor(context: Context, attrs: AttributeSet) : super(context, attrs)

init {
    LayoutInflater.from(context).inflate(R.layout.text_element_edit_card_layout, this)
    textContent = findViewById(R.id.text_element_content)
    editContent = findViewById(R.id.text_element_edit)

    actionContainer = findViewById(R.id.action_container)
    subActionContainer = findViewById(R.id.sub_action_container)
    editBtn = findViewById(R.id.edit_element_btn)
    resetBtn = findViewById(R.id.reset_element_btn)
    deleteBtn = findViewById(R.id.delete_element_btn)

    doneBtn = findViewById(R.id.done_edit_element_btn)
    cancelBtn = findViewById(R.id.cancel_element_edit)
    setAction()
}

@SuppressLint("CutPasteId")
override fun setElement(editableElement: EditableElement, adapter: ElementAdapter, listener: EditActionListener) {
    this.editableElement = editableElement
    this.listener = listener
    this.adapter = adapter
    editContent.visibility = View.GONE
    showTextContent()
}

override fun setAction() {
    super.setAction()
    editBtn.setOnClickListener {
        adapter?.isEditing = true
        adapter.currentItemEditMode()
    }
    resetBtn.setOnClickListener {
        editableElement.hasEdited = false
        editableElement.isDeleted = false
        editableElement.newContent = editableElement.element.content
        adapter.currentItemResetMode()
    }
    deleteBtn.setOnClickListener {
        editableElement.newContent = ""
        editableElement.isDeleted = true
        adapter.currentItemDeleteMode()
    }
    doneBtn.setOnClickListener {
        editableElement.newContent = editContent.text.toString()
        editContent.visibility = GONE
        showTextContent()
        actionContainer.visibility = View.VISIBLE
        subActionContainer.visibility = GONE
        adapter?.isEditing = false
        editableElement.hasEdited = true
    }
    cancelBtn.setOnClickListener {
        editContent.visibility = GONE
        textContent.visibility = VISIBLE
        textContent.text = editableElement.newContent
        editContent.setText(editableElement.newContent)
        actionContainer.visibility = View.VISIBLE
```

```
                subActionContainer.visibility = GONE
                adapter?.isEditing = false
            }
        }

        override fun hideAction() {
            actionContainer.visibility = GONE
        }

        override fun showAction() {
            actionContainer.visibility = VISIBLE
        }

        private fun showSoftInput() {
            val imm = (context.getSystemService(Context.INPUT_METHOD_SERVICE) as InputMethodManager)
            imm.showSoftInput(editContent, InputMethodManager.SHOW_IMPLICIT)
        }

        private fun showTextContent() {
            editContent.visibility = GONE
            textContent.visibility = VISIBLE
            if (editableElement.newContent.isNullOrEmpty()) {
                textContent.text = context.resources.getString(R.string.empty_paragraph)
            } else {
                textContent.text = editableElement.newContent
            }
        }

        private fun showEditContent() {
            textContent.visibility = View.GONE
            editContent.visibility = View.VISIBLE
            if (editableElement.newContent.isNullOrEmpty()) {
                editContent.setText("")
                editContent.hint = context.resources.getString(R.string.please_input)
            } else {
                editContent.setText(editableElement.newContent)
            }
            editContent.postDelayed({
                editContent.focusable = View.FOCUSABLE
                editContent.isFocusableInTouchMode = true
                editContent.requestFocus()
                editContent.setSelection(editContent.text.length);
                showSoftInput()
            }, 100)
        }

        override fun toEditMode() {
            actionContainer.visibility = View.GONE
            subActionContainer.visibility = VISIBLE
            showEditContent()
        }

        override fun toDeleteMode() {
            editBtn.visibility = View.GONE
            deleteBtn.visibility = View.GONE
            showTextContent()
        }

        override fun toResetMode() {
            editBtn.visibility = View.VISIBLE
            deleteBtn.visibility = View.VISIBLE
            showTextContent()
        }
    }

    interface EditActionListener {
        fun onStartEdit()
    }
```

```
package com.gorden.dayexam.repository.model

data class QuestionDetail (
    var courseTitle: String,
    var bookTitle: String,
    var bookId: Int,
    var paperTitle: String,
    var paperId: Int,
    var curQuestionId: Int,
    var questions: List<QuestionWithElement>
)

package com.gorden.dayexam.repository.model

import com.gorden.dayexam.db.entity.Book
import com.gorden.dayexam.db.entity.Paper
import java.util.*

data class BookWithPaper (
    val book: Book,
    var papers: List<PaperWithQuestion> )

data class PaperWithQuestion (
    val paper: Paper,
    val curQuestionId: Int,
    val questionCount: Int,
    val studyInfo: PaperStudyInfo,
    val question: QuestionWithElement? )

data class PaperStudyInfo (
    val studyCount: Int,
    val lastStudyDate: Date? )

package com.gorden.dayexam.repository.model

import com.gorden.dayexam.db.entity.question.Element

class QuestionWithElement (
    val id: Int,
    val type: Int,
    val body: BodyWithElement,
    val options: List<OptionItemWithElement>,
    val answer: AnswerWithElement )
{
    var realAnswer: RealAnswer? = null
    var studyCount = 0
}

data class BodyWithElement(
    val id: Int,
    val element: List<Element>
)

data class AnswerWithElement(
    val id: Int,
    val element: List<Element>
)

data class OptionItemWithElement(
    val id: Int,
    val element: List<Element>
)

data class RealAnswer(
    var answer: String = ""
)
```