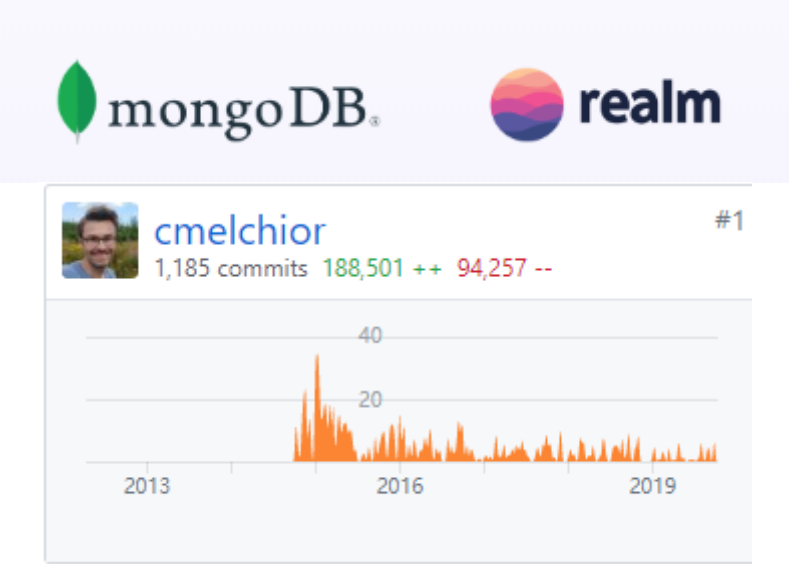
Realm-Java : Create reactive mobile apps in a fraction of the time



By main contributor Christian Melchior and other 83 contributors.

Abstract

Realm is a fast and cross-platform mobile database, running directly inside phones, tablets or wearables and designed to be an alternative to

Core Data and SQLite. The project was pioneered by 'Y Combinator' to make managing database easier and faster. Realm is an open source project living in GitHub and it is being maintained and developed by the core developers as well as quite a few external contributors. In this chapter Realm is analyzed in the following aspects: stakeholder analysis, context view, development view, technical debts, revolution perspective and conclusion. The chapter as a whole can serve as a helpful introduction to prospective developers looking to better understand the architecture to which they might contribute. (Lastly, we observe that Realm is a well-engineered project without glaring technical holes or major debt.)

Table of Contents

1. Introduction

2. Stakeholder Analysis

3. Context View

4. Development View

5. Deployment View

6.Functional View

7. Evolutional Perspective

8. Conclusion

[9. References](#)

Introduction

With the increasing use of the mobile devices such as phones, wearables and so on, the amount of data transferring between mobile devices grows exponentially every year, and the need for a database handling the data on mobile devices more efficiently is getting more intense. In 2014, the entrepreneurial team behind Y Combinator created Realm--The first database designed specifically for mobile platforms.

Realm is a cross-platform mobile database for iOS and Android, of which the core data engine is built in C++ instead of an ORM based on SQLite, so Realm is faster, better, and easier to use and accomplish database operations with less code than SQLite and CoreData. Besides, rather than simply encapsulating CoreData, Realm uses its own set of persistent storage engines. Furthermore, Realm is completely free, which makes it both more popular and more accessible to developers.

This chapter aims to give users a quick understanding of how Realm is organized, developed and maintained. Firstly, the Stakeholders and Context view are analyzed to provide insights into the organization of the project. Furthermore, detailed analysis of the architecture is presented in order to understand its structure. In addition, technical debt in terms of code, testing and documentation was also found throughout the analysis of the system and is presented in the chapter. This is followed by the evolution of the project explaining how Realm emerged from C++ and discussing the Realm which is considered as the alternative to CoreData and SQLite. Finally, the chapter is closed with the conclusion of the whole analysis.

Stakeholders

As a MVCC (Multi-Version Concurrency Control) database which runs on mobile phone, pad and wearable devices, hundreds of developer was engaged in its development on GitHub. Nowadays, it interacts with lots of stakeholders.

Stakeholders	Description	Quality Attributes Concerned
Acquirers	Realm is created by an entrepreneurial team of Y-Combinator, who are authorized funding for products and system development.	Usability,Reliability
Assessor	Rules and regulations that apply to the Realm product portfolio are handled by the General Management team.	Reliability,Usability,
Communicators	The Community Management team explains the system with documentation and training materials.	Portability
Users	Realm is used by students,hobbyist as well as professions who want an excellent database in the embedded system development.	Usability,Reliability,Portability
Testers	All bugs in the software can be posted on the Realm forum, emailed or issued on GitHub. Bugs and problems are picked up by the software and hardware staff employees of Realm or the contributors on GitHub.	Testability
Developers	There are both software and hardware teams working on the code for the end products.	Modifiability
Support Staff	The website realm.io plays a central role in staying in touch with users that report bugs as well as positive explanations about nice projects that can be built with the products.	Usability,Reliability,Portability
Suppliers	Realm is created by core data engine C++ and provide its own data engine. We can work with data objects directly and create database more efficiently.	Reliability
Maintainers	Software solutions are assessed by the Software development team in order to maintain the programming standards and architectural choices.	Maintainability

Fig.1 : Summary most important stakeholders

On April 24th,2019 MongoDB announced that Realm had signed a final agreement for the acquisition, which is expected to be completed by January 31st, 2020. Dev Ittycheria, President and CEO of MongoDB, said: "Realm is very popular with mobile developers because it allows mobile developers to use data to accelerate innovation easily, which is very consistent with our philosophy."

The power-interest grid is as followed:

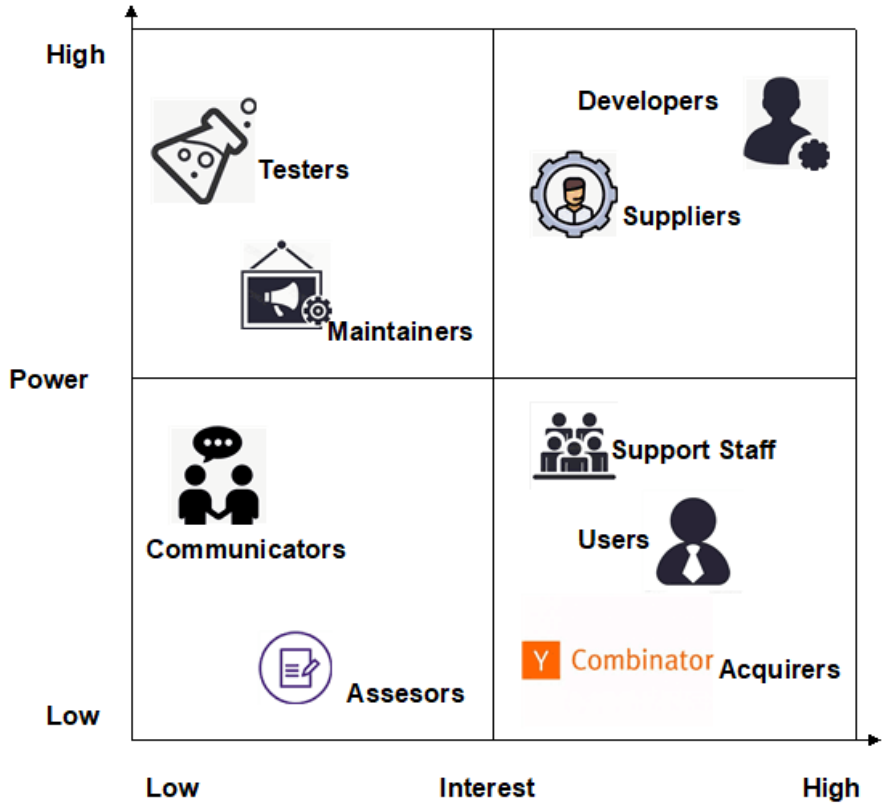


Fig.2 : Power Interest Grid of Realm

Fig.3 : Customers of Realm

The acquisition of Realm will deepen MongoDB's relationship with the developer community focused on mobile and Server-Free development. Active developers using Realm have exceeded 100,000, and the solution has been downloaded more than 2 billion times. Realm currently has Realm database and Realm platform, which can help users deploy quickly and achieve seamless cloud data synchronization.

Fig.4 :Wide usage of Realm

Now Realm is still trying its best to communicate with the latest and hot technical aspects, such as 5G, IBM Cloud Functions, Microsoft Azure and so on. The group of stakeholders will become more and more powerful, as well as its bright future.

Context View

The context view shows the different relations, dependencies and interactions Realm has with its environment. Important for the context are the people, systems and external entities with which the system interacts.

System Scope & Responsibilities

According to its website, Realm Database is defined as a "**fast, easy to use, and open source alternative to SQLite and Core Data for IOS and Android developers today**". The scope of the software is clearly defined here. It is a database engine utilizing many new features (e.g. offline functionality, data synchronization, etc.) on mobile platforms, which are IOS and Android.

And the responsibilities define what the system should do in order to fulfill its objective. These include the following:

- Employing features of offline-first functionality, fast queries and safe threading.
- Providing powerful encryption that is strong enough for global financial institutions.
- Providing seamless data synchronization with the aid of Realm Platform.
- Enabling users to build reactive apps with live objects, which means they always have the latest data.
- Open source on GitHub, allowing add-ons from open community.

External Entities

Realm Java is a widely-used mobile database out of Y Combinator Inc. As one can imagine, a software project like this cannot be developed without external libraries, tools and frameworks. On the other, many companies cannot develop their software without Realm Java. These external relations are examined in this section.

Fig.5 :External Entities

In order to understand the context model of Realm Java, the role of the external entities with respect to Realm Java is explained.

- **Programming Languages & Supported Platforms & IDE**

  Realm Java is now available only on the Android platform. Given that, Android Studio is the one IDE that users working with Realm Java should work on. Accordingly, no programming language but Java is allowed. The software of Android Studio can be installed on multiple platforms: Linux, Mac OS and Windows.

- **Developers**

  The very first version of Realm was written in C++ by a development team From San Francisco. Soon, new versions of Realm written in other mainstream languages are developed with the help of open source community. Realm Java, for example, is now maintained by the combined effort from Realm team and individual developers on GitHub.

- **Development Tools**

  As is mentioned above, Realm Java requires specific development tools such as Java SDK and Android Studio. Also, there are tools developed by Realm team to operate on Realm database. Realm Studio helps users open and edit local and synced Realms, and administer any RealmServer instance. Realm Browser, which is available only on Mac OS, shares the same abilities.

- **Users**

  Thanks to its powerful features, Realm Java is widely used in all kinds of trades. For example, Cartasite built an offline-first app for heavy industries that seamlessly syncs field data with backend system, and Tread Learning uses Realm Platform to help special education teams coordinate care by syncing data in realtime across multiple devices. After all, Realm is built into apps used by hundreds of millions of people every day.

- **Version control and Issue Tracker**

  Realm Java uses GitHub and Git for version control to help developers collaborate and track issues related to the software. Contributions in this repository come from the Realm Java development team, as well as from open source community.

- **Competitors**

  Main competitors in the field of mobile app database is SQLite and Core Data, which have considerably large user base. According to Realm website, however, Realm database outweighs its competitors in many aspects.

- **Communication Tools**

  In order to stay in contact with developers and users, the following communication tools are used: the Realm Website, the Realm Forum. Also, using Git, individual developers can communicate with Realm team on a technical level. What's more, Realm's community managers also need to maintain a good relationship with magazines like ADT Magazine who writes about Realm products and competitors.

Quality Attributes

Runtime System Qualities

Functionality

- Employing features of offline-first functionality, fast queries and safe threading.
- Providing powerful encryption that is strong enough for global financial institutions.
- Providing seamless data synchronization with the aid of Realm Platform.
- Enabling users to build reactive apps with live objects, which means they always have the latest data.

Performance

Scenario: Realistic limits of storing declared by Realm Doc are violated.

Source of Stimulus: Users

Stimulus: Inappropriate ways of storing information

Artifact: System

Environment: Under normal operations

Response: Presenting warnings and errors

Security

Scenario: Hackers attack the database and attempt to steal information stored.

Source of Stimulus: Hackers

Stimulus: Attempts to steal stored information

Artifact: Data stored in the database

Environment: Whether users are online or offline

Response: Verify visitors' identification through a symmetric key (AES)

Availability

Scenario: Possible mistakes made by users when building app with Realm database.

Source of Stimulus: Users

Source of Stimulus: Users

Stimulus: System crash due to users' mistakes

Artifact: Component that dealing with crashes

Environment: Under normal operations

Response: Record the crash information and inform the parties concerned

Usability

Scenario: Users are unfamiliar with the configuration steps and want to learn them.

Source of Stimulus: Users

Stimulus: Attempt to learn about configuration steps and system features

Artifact: System

Environment: During configuration

Response: Provide links to help documents

Non-Runtime Qualities

Modifiability

Scenario: Developers in open source community have better ideas to help improve the Realm Java Project.

Source of Stimulus: Developers in open source community

Stimulus: Commitments of modifications of Realm Java source code.

Artifact: System

Environment: During designing

Response: Test the modifications and deploy them.

Integrability

Scenario: Users have demand of integrating Realm with other commonly used libraries for Android.

Source of Stimulus: Users

Stimulus: Importing of external libraries.

Artifact: System

Environment: During compiling

Response: Link with external libraries.

Testability

Scenario: Users need to test the apps developed in Realm Java unit by unit.

Source of Stimulus: Users

Stimulus: Completed unit

Artifact: Component being tested

Environment: During developing

Response: Prepare testing environment and provide results.

Architecture Qualities

Conceptual Integrity

- The whole point of Realm, or at least one of its very core ideas, is that it is objects all the way down.
- Try to minimize overhead.
- Try to keep as many operations as zero-copy as possible.

Development View

Module organization

Obviously, realm-java is composed of a large quantity of source files,which are logically organized into disparate modules. We have made a module organization for getting a much clearer understanding of the whole realm-java project.

A module organization describes how the organization of the source files classified into modules that contain related code. Such a structure provide an overview of the source code which guides developers to understand and navigate the codebase.

All main modules are described in the table below .

Module	Description
Test	All the test-related files.
Exceptions	Classes for exceptions and errors,stack trace removing logic,cleaning public APIs.
JUnit	Realm-java JUnit integration,rule and runners,and JUnit integration support classes.
Listener	Public classes related to the listener APIs.
Handler	Classes calling all listeners wanted for the realm-java, before delegating it to the parameterized handler.
Runners	JUnit runners,internal classes and utils for runners implementation.
Session	Realm-java session builder and implementation.
Verification	Verification checkers, implementations for dealing with matching arguments,verification logic and implementation.
Creation	Classes for realm object creation including its settings, instances.
Debugging	Everything which helps debugging failed testss
Util	All the static utils including reflection utilities, IO utils, and so on.
Invocation	Public API related to realm method invocations,invocation related classes, and implementations of method calls.
Plugins	Plug-in components realm use in the process of compiling.
Transaction	Encapsulate a realm transaction.
Annotations	Annotations realm use to mark the field or classes.
Progress	Class used to encapsulate progress notifications when either downloading or uploading Realm data.
Session	Realm session builder and implementation.
Migration	Classes used to perform the migration of one realm schema to another.
Schema	Class for interacting with the schema for a given `RealmObject` class.
UserStore	Interface for classes responsible for saving and retrieving Object Server users again.

Fig.6 :description of the main modules

All modules interact together to implement the function of Realm-java. In table 2, we can find out the modules into which the individual source files are collected and e dependency among these modules, and only the main part of the project is shown.

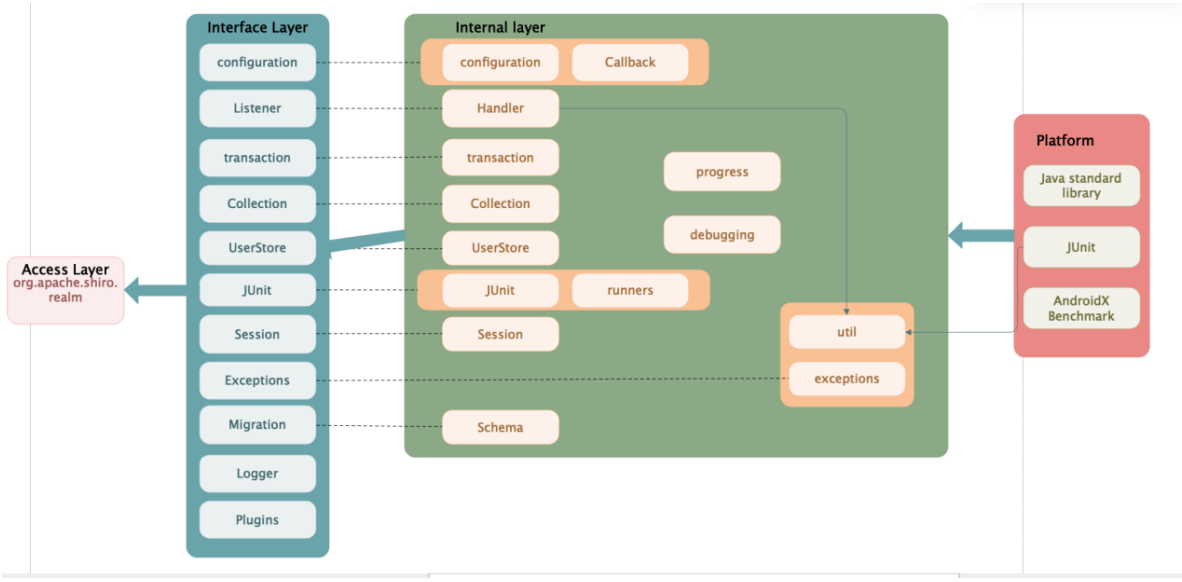


Fig.7 :main part of the project

Access layer - Provide external support of Realm-java. Through Gradle or Maven, users can access Realm-java as a library for project.

Interface layer - Interfaces for the internal layer,including exceptions interface, configuration interface, `UserStore` interface,session,migration and so on.

Internal layer - All the main classes in Realm-java are functioned, including `exceptions`, `configuration`,`userstore`, `transaction`, `session`, and so on.

Platform - All the external and basic libraries for internal layer,including Java standard library and JUnit.

Common Design Models

In this section, common designs that are used and standardized in the development of Realm are described.

Common Process Standardization

As a MVCC Database,Realm in Java has some common used models to follow. First of all,it must run on Android platform. Java other than Android is not currently supported. Meanwhile, the version of Android studio must be above 1.5.1 and the version of SDK must be updated. JDK should also be above Version 7.It also supports Android 2.3 and the later.

To configure the Realm,`RealmConfiguration` present with some methods:

- `Builder.name`: Specifies the name of the database. If not, it is default.
- `Builder.schemaVersion`: Specify the version number of the database.
- `Builder.encryptionKey`: Specify the key for the database.
- `Builder.migration`: Specify the migration class for the migration operation.
- `Builder.inMemory`: Declare database persistent only in memory.
- `build`: Finish configuration。

Design Standardization

For Realm is an open source platform, everyone is free to contribute to the repository on GitHub. Seeing as multiple contributors are influencing Realm, the core developers have standardized aspects of the design of the system to make it as maintainable, reliable and technically cohesive as possible. The most important aspects of contributing to Realm are discussed in the Realm

Platform Extensions License part on GitHub:

- Contributors are Licensors and any individual or Legal Entity on behalf of whom a contribution has been received by Licensor and subsequently incorporated within the work;
- Unless contributors explicitly state otherwise,any contribution intentionally submitted for inclusion to the Licensor shall be under the terms and conditions of the License, without any additional terms or conditions;
- When contributors reproduce and distribute copies of the work or derivative works thereof in any medium, with or without modification,they must present modified files to carry prominent notices stating that they changed the files.

Test Standardization

By standardizing the test approaches, technologies and conventions, the overall testing process remains consistent and has a higher pace. Realm has a standardized fold Tests which contains all kinds of tests needed in the development.

File `Configuration/Realm/Tests.xcconfig` is to configure the test code and locate to `Realm/Tests/RealmTests-Info.plist`,which contains test information such as `Query tests`, `property tests`, `object tests` and so on will execute to test the modified codes. Thus, the contribution can be merged into the process when build succeeds.

Codeline Model

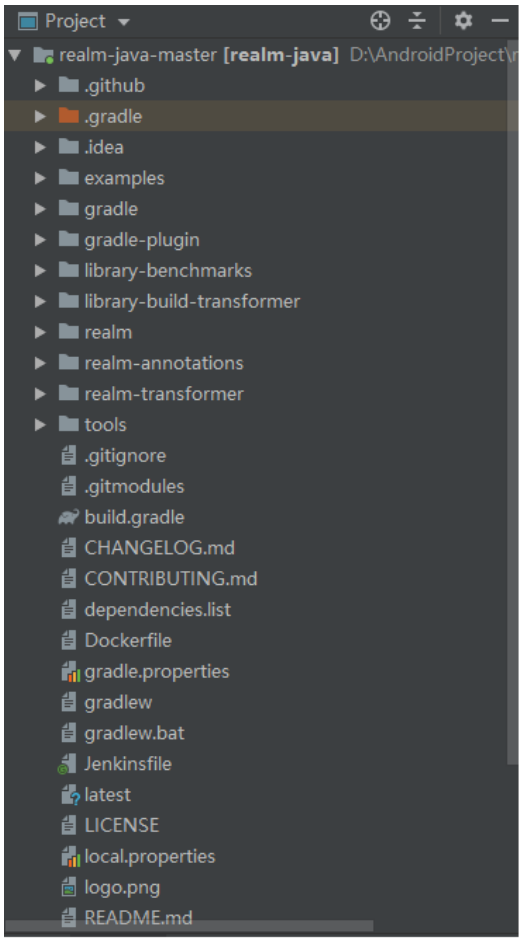


Fig.8 : file structure of Realm

The core of Realm-Java is partitioned into following parts, and we will discuss them in the following part :

- **realm** : contains the key code in Java and Kotlin.
- **realm-annotations** : contains the added annotations in Realm.
- **realm-transformer** : take part in compilation to generate code.

realm

Folders underneath the *realm* folder:

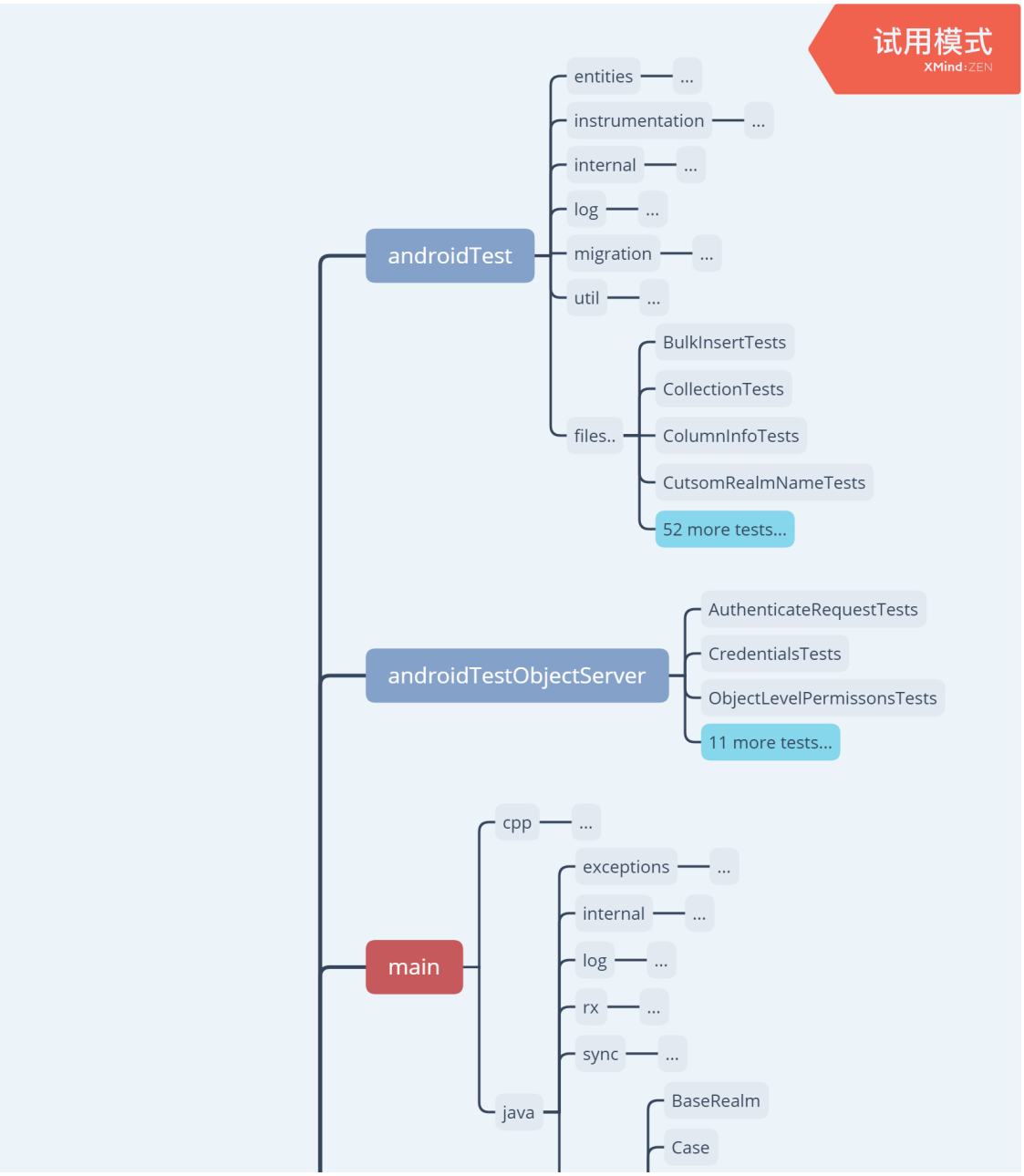
- **kotlin-extensions**

  This folder contains code in Kotlin that provides Realm functionality in a Kotlin program. For example, in the code of file `RealmModelExtension`, it presents many functions such as `deleteFromRealm()`, `isValid()`, `isLoaded()` etc. These functions can be called in a Kotlin program to create or edit Realm Object.

- **realm-annotations-processor**

  Code in this folder plays the role of annotation processor in Java. The Realm-annotations are explained in the following part. `annotations-processor` explain itself literally. Since the realm annotation processor can work at the compile stage of a Realm program, we can use it to do what we need to do during compilation. A common use of realm annotation processor is to get the relevant realm annotation data at compile time, and then dynamically generate Java source files (let the machine write the code for us), usually automatically generate some regular repetitive code, which solves the problem of manual repetitive code and greatly improves the efficiency of coding.

- **realm-library**



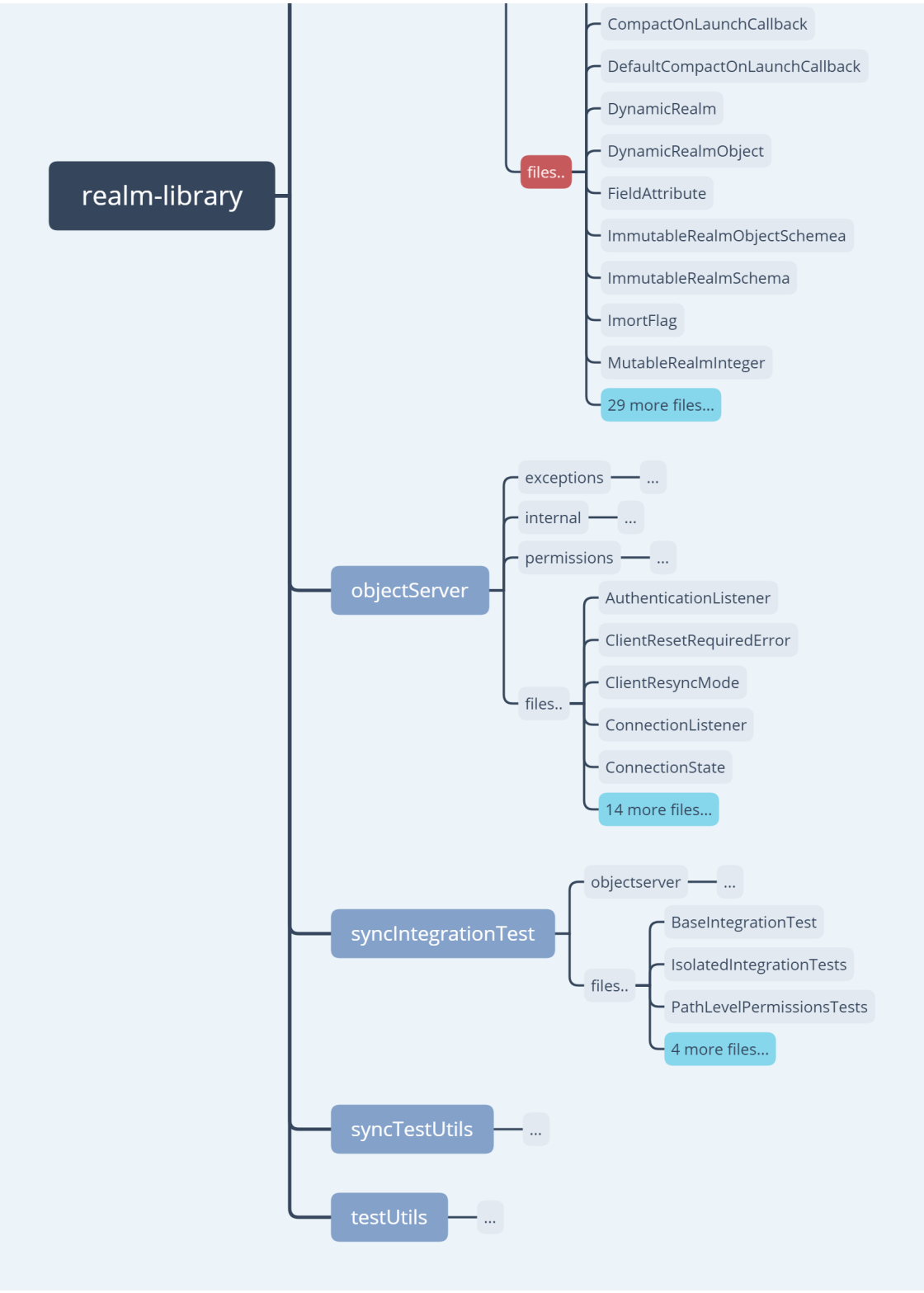


Fig.9 : file structure of realm-library

This is the core folder containing the functional code of Realm. `androidTest` folder contains file that make example of a given function in applying Realm. The purposes of each tests are revealed by its name literally. code in `androidTestObjectServer` test Realm `objectServer`. Deploying a server object allows programmers to run realm remotely. `testUtils` contains code that are for test as well. The core part of Realm is encapsulated in the `main` folder. Two folders are located underneath this folder: `cpp` and `java`. The `cpp` folder contains code mainly for compilation, and code in Java makes up the core of Realm, such as defining `RealmObject`, `Realm Error`, `Realm Collection` etc., and make base classes such as `BaseRealm`, `RealmList` etc.

- **other folders**

Auto generated folders are listed in this classification, such as `config`, `gradle`, `tools` etc.

realm-annotation

Code under this folder create annotations for realm, which usually work with annotation-processor we mentioned before. Several annotations are created here, such as: `RealmField`, `RealmClass`, `PrimaryKey` etc. Annotation are special tags in the proxy that can be read and processed at compile, loading class, and run-time starting with JDK5. By using annotations, developers can embed additional information in the source file without changing the original logic. This makes sense in Realm, since Realm provides many functions and attributes for developers so that developers don't need to do extra work for the object they have to declare manually. The reason why programmers benefit from the simplicity of such usage lies in the code underneath this folder. On the other hand, another use of this folder is to keep the code written by the programmers in accordance both logically and grammatically.

realm-transformer

This part has three key Java files: `ComputerIdentifierGenerator`, `RealmAnalytics`, `Utils`, which plays an important role in helping Realm generate code during compilation.

Deployment View

The deployment view is what looks into how the program is expected to operate in live operation. It will show what hidden dependencies realm-java has,realm-java's runtime environment and finally the specialist knowledge for realm-java.

A diagram of the deployment view is given in the Figure 14

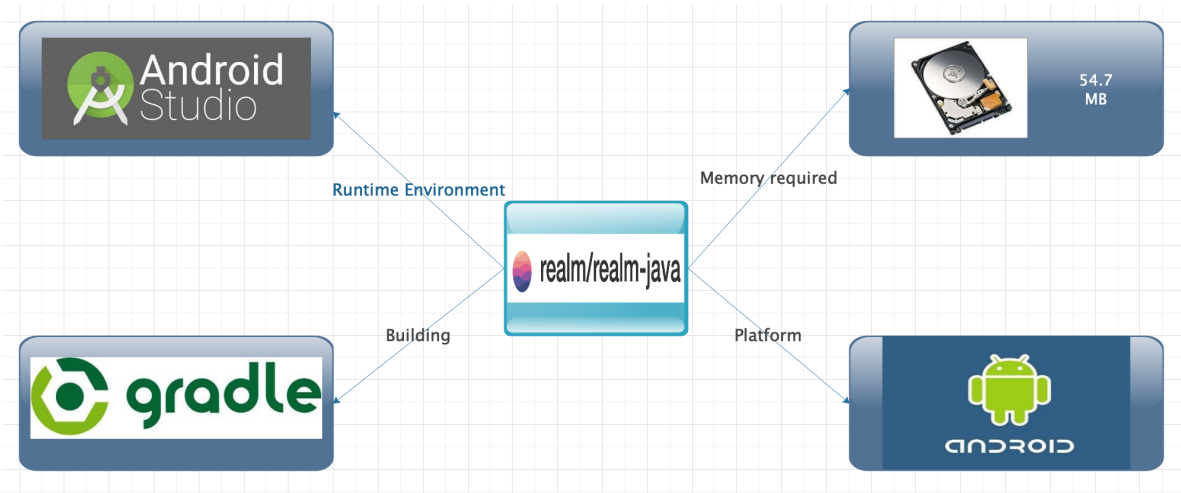


Fig.10 : Graphical Overview of the Deployment view

Dependencies

Running a realm-java instance requires several supporting libraries and programs.

These are, at the time of writing:

- Android Studio version 1.5.1 or higher
- JDK version 7.0 or higher
- A recent version of the Android SDK
- Android API Level 9 or higher (Android 2.3 and above)
- GSON: a library created by Google for deserializing and serializing JSON. GSON should work with Realm out of the box.

- Retrofit: makes it easy to work with a REST API in a typesafe manner
- Rxjava: extends the Observer pattern
- Lombok
- PowerMock

Realm-java requires Gradle to be built. Gradle also handles collecting all the external Java dependencies when building realm-java from source. Getting the dependencies by serching for them on Google is not only boring,but also some up-to-date versions of the dependencies cannot be found while looking on the internet.

Aside from the memory required to install realm-java, no other hardware requirement for using realm-java are mentioned anywhere. The required storage space is around 54.7MB.

Runtime Environment

As realm-java runs entirely within a single system,and on this system it runs inside Android Studio. Most of its runtime environment is either highly simplified or defined by its hosting programs.

- A computing system(computer or phone .etc) with support for Android Studio and a supporting processor architecture.
- The prerequisites mentioned in dependencies.

Nevertheless, in most cases,realm-java is used on rather large data sets,which will increase RAM usage.

Functional View

In this section we will discuss the most important functions and unique characteristics of Realm database compared with other lightweight database such as SQLite, CoreData etc. We will also address the architecture elements and choices that make these functions possible. Realm offers vast amount of unique functionality such as Multiversion Concurrency Control, Local linkage, String and Int optimazing etc. The detailed descriptions of these functionalities are as followed.

Function Capabilities

MVCC(Multiversion Concurrency Control)

The aim of this function is to support mobile apps with multiple threading access. Its design is just like the Git, since it contains fork, atomic commit etc. It means that you don't have to copy all the data but just working on a single branch. Now here is how it works: when starting a transaction, you get a copy of the whole database. In the mean time, hundreds or thousands of other threads might also want to start a transaction. Since what they get is just the copy of the database, worries of one thread may affect one another are completely not necessary. After the transaction is committed, the changes are all applied to the original database maintaining the stability of the system while little efforts are done to keep this function.

With this function, write transactions no longer need to block from read transactions. From what we discussed above, since a thread gets a copy of the database, read transaction could be regard as reading a constant, which won't interfere with the write transactions.

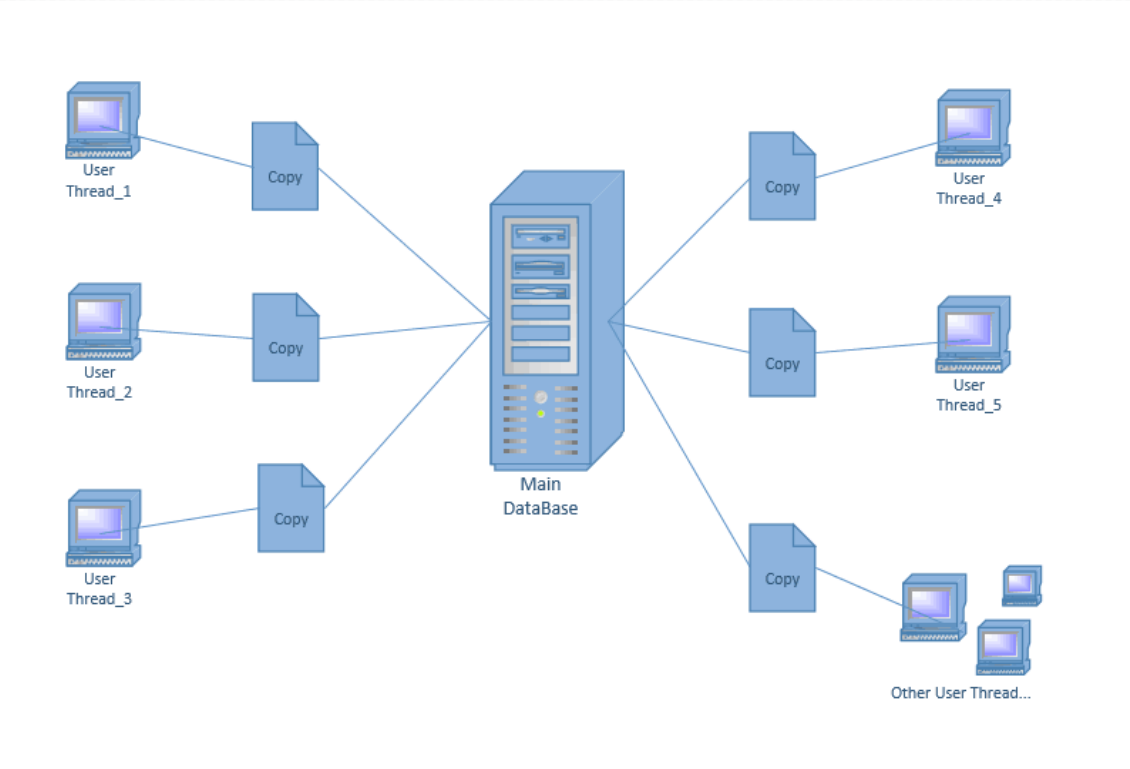


Fig.11 : MVCC Framework

The code block to realize MVCC lies in ***/realm/realm-library/src/main/java/io/realm/Realm.java*** . The main code are as followed:

```
/**
     * Executes a given transaction on the Realm. {@link #beginTransaction()}
and {@link #commitTransaction()} will be
     * called automatically. If any exception is thrown during the transaction
{@link #cancelTransaction()} will be
     * called instead of {@link #commitTransaction()}.
     *
     * @param transaction the {@link io.realm.Realm.Transaction} to execute.
     * @throws IllegalArgumentException if the {@code transaction} is {@code
null}.
     * @throws RealmMigrationNeededException if the latest version contains
incompatible schema changes.
     */
    public void executeTransaction(Transaction transaction) {
        //noinspection ConstantConditions
        if (transaction == null) {
            throw new IllegalArgumentException("Transaction should not be
null");
        }

        beginTransaction();
        try {
            transaction.execute(this);
            commitTransaction();
        } catch (Throwable e) {
            if (isInTransaction()) {
                cancelTransaction();
            } else {
```

```
                RealmLog.warn("Could not cancel transaction, not currently in a
transaction.");
            }
            throw e;
        }
    }
```

Local Linkage

The core of the file system is linkage. In Realm, execution of retrieving object mainly relies on 'B+ Tree'. Therefore, the speed of searching objects are much faster than other databases. Instead of creating dual abstractions for the relationship between ORM, programmers can create direct object links as part of the file transformation at the file system level. The same is true for retrieval, such as integer columns, relationships, one-to-many relationships, or even many-to-many relationships.

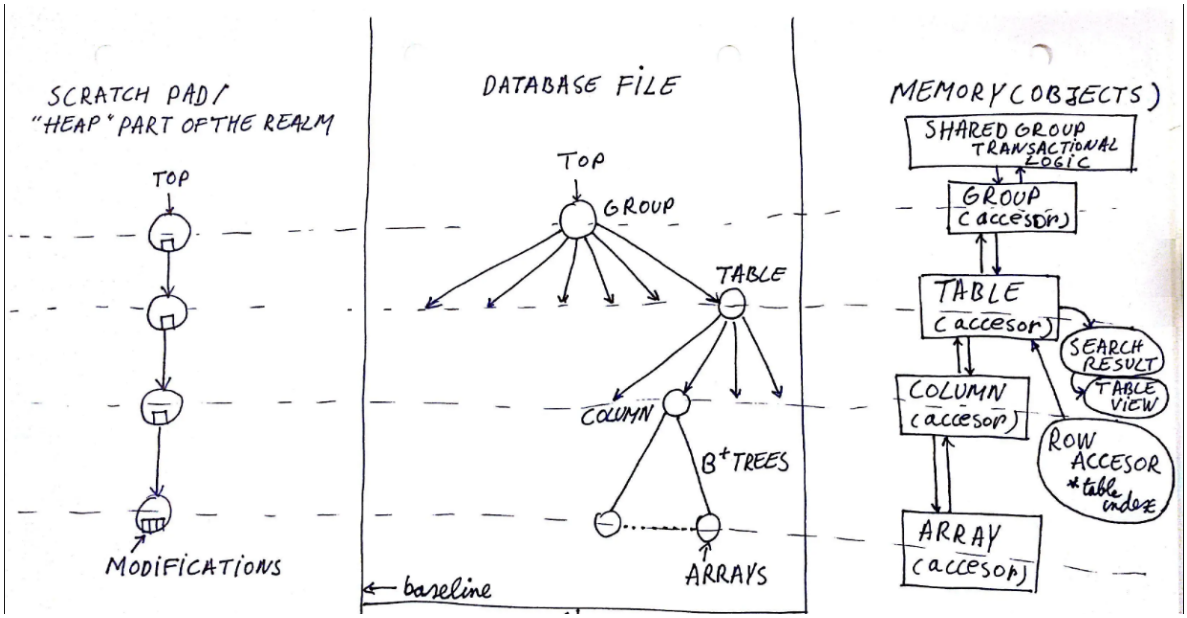


Fig.12 : B+ Tree Structure of Local Linkage

Here is how this function works: when realm object is created, it calls the method `createObject`, this method returns the function `createObjectInterval` which has a return value of a generic subclass object extends from `RealModel`, whose code block is listed below. The return value `configuration` is an initialized object of class `RealmConfiguration`, which calls the method `getSchemaMediator`. With the object of `RealmProxyMediator`, it calls the function `newInstance` finishing the process of creating a new Realm object and keeping it in a B+ tree structure.

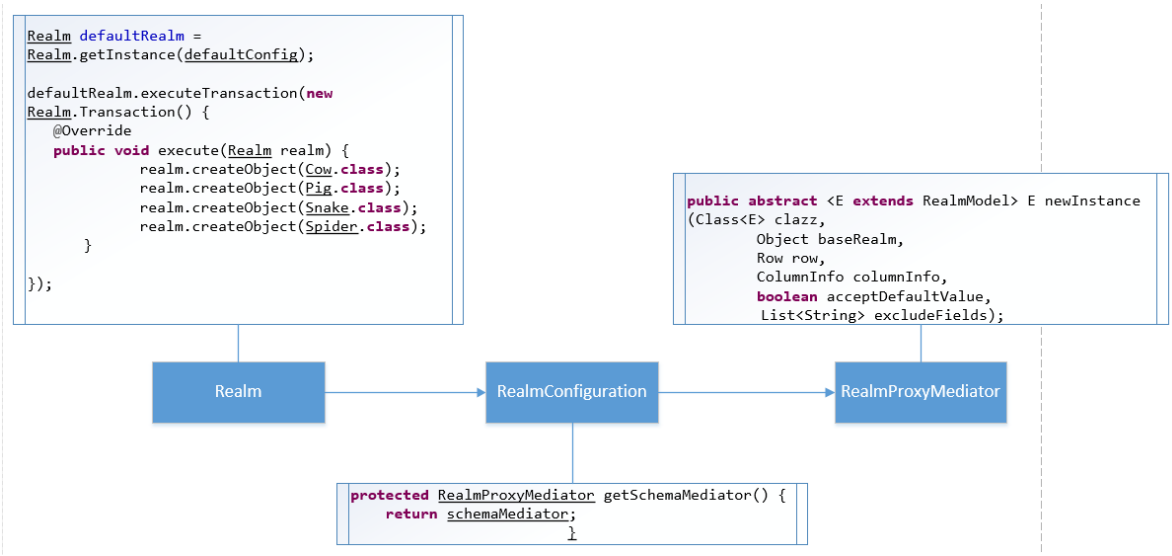


Fig.13 : Process from Creating Object to Local Linkage

The main code blocks lies in ***/realm/realm-library/src/main/java/io/realm/Realm.java*** .and the main code are listed below:

```
/**
     * Same as {@link #createObject(Class)} but this does not check the thread.
     *
     * @param clazz the Class of the object to create.
     * @param acceptDefaultValue if {@code true}, default value of the object
will be applied and
     * if {@code false}, it will be ignored.
     * @return the new object.
     * @throws RealmException if the primary key is defined in the model class
or an object cannot be created.
     */
    // Called from proxy classes.
    <E extends RealmModel> E createObjectInternal(
            Class<E> clazz,
            boolean acceptDefaultValue,
            List<String> excludeFields) {
        Table table = schema.getTable(clazz);
        // Checks and throws the exception earlier for a better exception
message.
        if (OsObjectStore.getPrimaryKeyForObject(
                sharedRealm,
configuration.getSchemaMediator().getSimpleClassName(clazz)) != null) {
            throw new RealmException(String.format(Locale.US, "'%s' has a
primary key, use" +
                    " 'createObject(Class<E>, Object)' instead.",
table.getClassName()));
        }
        return configuration.getSchemaMediator().newInstance(clazz, this,
                OsObject.create(table),
                schema.getColumnInfo(clazz),
                acceptDefaultValue, excludeFields);
    }
```

String and Integer Optimization

Realm optimize data, they perform operations such as transformations between variable types. For example, when you encounter a lot of repetitions of strings, Realm will automatically transform transverse them and convert them into enumerated type; Integer objects are casting to int type etc.

The code block to realize this function lies in ***realm/realm-library/src/main/java/io/realm/RealmFieldType.java***, and the main code are listed below:

```
/**
     * Converts the underlying value used by the storage engine to the proper
Realm type.
     *
     * @param value the value to convert
     * @return the corresponding Realm type.
     * @throws IllegalArgumentException if value isn't valid.
     */
    public static RealmFieldType fromNativeValue(int value) {
        if (0 <= value && value < basicTypes.length) {
            RealmFieldType e = basicTypes[value];
            if (e != null) {
                return e;
            }
        }
        if (LIST_OFFSET <= value) {
            final int elementValue = value - LIST_OFFSET;
            if (elementValue < listTypes.length) {
                RealmFieldType e = listTypes[elementValue];
                if (e != null) {
                    return e;
                }
            }
        }
        throw new IllegalArgumentException("Invalid native Realm type: " +
value);
    }
```

Crash Safety

This function relies on the B+ Tree structure we discussed before. When you commit a change, the write copy action is initiated, which means that you create a "B+ Tree" branch and do not modify the data of the original branch. So that if something goes wrong, the original data will not be corrupted. Fortunately, because of the good architectural design, the top-level pointer will always point to the tree structure that didn't crash, and your write transaction will happen somewhere else. When you finally decide to commit the changes, once Realm confirm that the data is safely synchronized to the hard drive, Realm will move the pointer to the new official version. This means that in the worst case, you will only lose the changes you are currently making, but not all the data.

Zero Copy and Lazy Loading

Originally when retrieving data from a normal database, a SQL statement should be created. After all the data are fetched, the data are then transferred to the hard disk. Before the data reaches the memory, it has to go through deserialization by CPU. Finally, there are still couple of manipulations to be made before the program can use the data it originally wants. However, things are much easier with Realm. Since Realm skips the whole copy process. First, files are always memory-mapped, and you can access any content of a file whether it's in memory or not.

The important thing about core file formats is that we make sure that the file formats on the hard disk are memory-readable so that Realm doesn't need to do any deserialization. As we can see, Realm skipped the whole steps. All you need to do is to calculate the offset in the file so that you can read the data in the memory-mapped memory, by reading the offset and the data between the length of the data, and then you can get the original value through property access. In a word, Realm skips most of the steps and make fetching data much more efficient. This is what Zero copy does, its name reveals its function literally.

Another merit of Realm in reading data is Lazy Loading. Instead of storing data horizontally normal database uses, Realm stores data vertically. A more straight forward saying is instead of doing a lot work to realize a certain manipulation to the data, Realm tries to modify the data property to reach the same effect. The most obvious advantage of Lazy Loading is that it avoids fetching data between hard disk and memory, which saves a lot resources and time.

Functional Interactions

Some of the core functions of Realm-Java are described above. In this section, we will focus on how these functions and functional modules interact with each other.

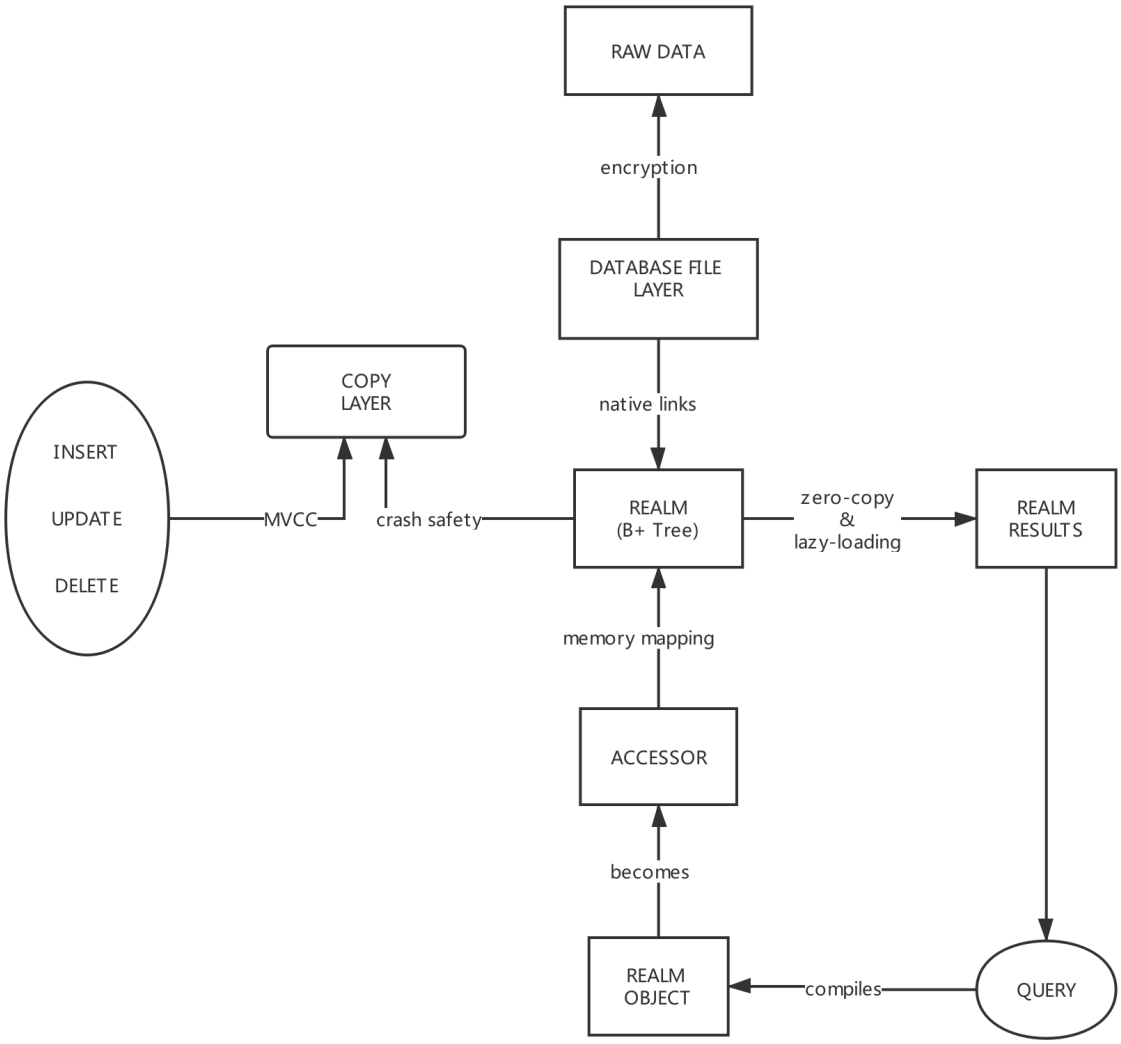


Fig.14 : Frame of Basic Manipulation in Realm

QUERY

- Firstly, Realm compiles the query object, transforming it to an accessor.

- Secondly, using data mapping, users may access the data in database directly trough accessors.
- Thirdly, Realm uses zero-copy mechanism so as to save itself from deserializing the returned data. And by keeping properties contiguously linked at the vertical level, lazy-loading makes it possible to access raw data more efficient.
- Last, the data of type REALM RESULT is returned and shown.

INSERT & UPDATE &DELETE

- MVCC helps to control concurrency when multiple writing operations occurs. It is always append-only and copy-on-write. You're always working in a fresh, isolated state for your write transaction. The way that it's built under the hood is that we basically get the concurrency at the database level for free, and what we need on top of that is to notify the other process whenever things happen. We just use a named pipe for that, which is pretty low overhead.
- As you're making changes, copy-on-write behavior is happening, meaning you're forking the B-tree and writing without modifying existing data, so if something goes wrong, that initial data is still there. Thankfully, by design, that top level pointer is still pointing to the non-corrupt tree; you're just writing elsewhere. What happens finally is that you've got this two phase commit concept, where once we've confirmed that everything is synced to disk and safe

DATA STORAGE

The whole point of the file structure is that everything is a link. We like saying that it's just B-trees all the way down, which is why queries on multiple links — relationships — are so fast in Realm. You don't have dual abstraction between ORM to relational, but raw links to the objects at a file system level in the file format. It's the same if you're querying, say, an integer column, or a relationship, or a one to many, or a many to many.

l Since Realm memory maps the whole file, it can protect parts of that memory. If anyone tries to read from this encrypted chunk, realm throws a file system violation that says someone is trying to access the encrypted data, and only after decryption can user access the chunk.

External Interfaces

The external interfaces provided by Realm-Java mainly concerns functionality to extension development possible. Here is a list of them and their description.

Realm Model

Helps to declare the Realm data models.

Transaction

Automatically executes the starting and committing of input transaction, and cancels that when error.

Parcelable

Automatically generates the boilerplate.

Evolutional Perspective

The evolution perspective deals with concerns related to evolution during the lifetime of a system, which is relevant to Realm for it need to be changing continuously along with the coding language and developing tools.

I've really tried to find some resources or source project of the previous versions of Realm, in order to draw the evolutional view of the whole system structure. However, since Realm is still at minority and references about it are still at a small amount, the previous versions of Realm are deprecated and are not available for download as well, in this section I will mainly introduce to you the main breakthroughs in each version and the comparison between.

Realm has changed significantly since its first draft in 2014-09-29. Its initial intension was to build a mobile database which is faster and safer than the existing one such as SQLite etc. However, five years from then, Realm has done much more than that. It supports not only Android, but Swift, JavaScript, .Net other platforms as well. It's an embedded database which provides developers a distinct way to treat modules and their service logics. All notable changes to the project are documented in the CHANGELOG.md.

The details of each update since Realm originally developed are listed below.

The first draft version is 0.71.0, released in 2014. After the first version, various changes have been made, which were either to fix bugs or to add new features. The first breaking release took place in 2015-10-08. This release announced that the current Realm files no longer support previous ones, and many deprecated functions, methods, and constructors from the Realm class were removed. The most noteworthy one is that boxed type such as Boolean, Byte, Short, Integer, Long, Float and Double were introduced to the Realm class. Additionally, from now on, Realm adds support for x86-64. Other bug fixing job has done in this version as well.

Fig.15 :Realm Version 0.88.0 prerequisite

Five months later, in 2016-03-10, Realm has released version 0.88.0. In this version Realm has to be installed as a Gradle plugin, which means it relies on Android Studio from now on, and do not support Java outside of Android. This means that Realm in Java has a tighter bonding relationship with Android Studio, which indicates it has a bigger potential to be applied in an Android Operating System mobile device.

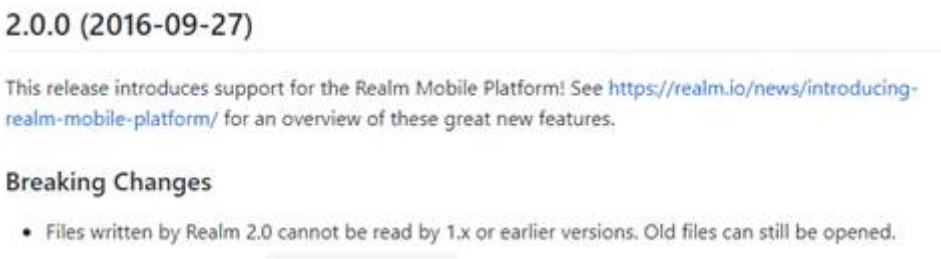


Fig.16 :Realm Version 2.0.0

In 2016-09-27, Realm released the version of 2.0.0. As we can see, within 2 years, Realm has released two upward compatible versions. On the one hand, we have to admit that, with the increasing amount of software engineers joining into the project, as well as the popularity of the development of 'light database', Realm has made great breakthroughs and provided convenience for mobile software development. However, on the other hand, frequent forward upgrades increase the instability of the whole system. The new release also added many features, such as added new `RealmFileException` etc. However, Realm 2.0.0 contained a serious bug 'Build error when using Java 7' which would cause serious problem for programmers for Java 7 is still the latest version back then, and was not finished until version 2.0.2.

Fig.17 :Realm Version 4.0.0

For almost two years, programmers have been using Realm from version 2.X.0 to 3.X.0. Realm has made a lot of changes during this period but not as noteworthy until the release of version 4.0.0 in 2017-10-16. In Realm 4.0.0, The internal file format has been upgraded, which means opening an older Realm will upgrade the file automatically, but older versions of Realm will no longer be able to read the file. Additionally, Realm upgraded a series of Internal components to adapt to the current version.

Fig.18 :Realm Version 5.0.0

Half a year later, Realm 5.0.0 was released in 2018-03-15. This release is compatible with the Realm Object Server 3.0.0-beta.3 or later.

The latest version is Realm 6.0.1, which was released in 2019-11-11, and the lead developers are still making progress for new version. The project is published on GitHub, which is still an active project. This is necessary, considering Realm is intended to replace SQLite as a faster and lighter mobile database.

Conclusion

This chapter summarized Realm in many architectural views and perspectives,helping the reader to be able to understand and contribute to the project. The analysis of the Realm project has led to the conclusion that Realm has an interesting, yet complex architecture.

Stakeholders in the project,company-wise as well as users and external developers are identified. Realm,which hundreds of software engineers and maintainers contribute to,is such a widely used database in embedded system development.

After that, in the *Context View*,we shows the different relations, dependencies and interactions Realm has with its environment. First,the scope of Realm is explained to be a database engine utilizing many new features on mobile platforms, which are IOS and Android. Second,the external entities are identified in the External Entities Figure,At last,we introduce programming languages and program platforms used in the development,as well as the users,the competitors and communication tools.

Then comes the *Quality Attributes*. Runtime system qualities,non-runtime system qualities and architecture qualities are all explained to dig deeper into the system architecture and its usages in certain scenarios.

Next,in the *Development View*,the principles that guide the development of Realm are given in *Module Organization* for getting a much clearer understanding of the whole realm-java project. The *Common Design Models* also explained some of the testing and design standards.

The *Evolution Perspective part deals* with concerns related to evolution during the lifetime of a system, which is relevant to Realm for it need to be changing continuously along with the coding language and developing tools.

Furthermore,in the *Deployment View*,the main components necessary for launching Realm were given. We have explained what hidden dependencies realm-java has,realm-java's runtime environment and finally the specialist knowledge for realm-java.

Finally,in the *Functionality* part,the most important functions and unique characteristics of Realm database compared with other lightweight database such as SQLite, CoreData etc. are present from the perspective of architecture elements and choices by showing code-line examples.

All in all,Realm always keeps its pace with the latest and hottest technical aspects, such as 5G, IBM Cloud Functions, Microsoft Azure and so on. Realm's architecture is still being modified for better and easier usages,which helps it to embrace a brighter future.

References

1. Nick Rozanski and Eoin Woods.2012. Software System Architecture. Working with Stakeholders Using Viewpoints and Perspectives. Addison-Wesley Professional.
2. The realm-java licenses on Github, https://github.com/realm/realm-java/blob/master/LICENSE, access date : 10-10-2019
3. Stack Overflow, http://stackoverflow.com, access date : 10-10-2019
4. Issues for realm-java on Github, https://github.com/realm/realm-java/issues, access data : 10-10-2019
5. Realm-java user guide,dependencies, https://github.com/realm/realm-java/blob/master/dependencies.list, access date : 10-10-2019
6. Gradle , https://gradle.org/, access date : 10-10-2019
7. Statistics about package use in Java projects on Github,Google,https://cloud.google.com/bigquery/public-data/github,access date : 10-10-2019
8. Pull Requests on the realm-java Github,https://github.com/realm/realm-java/pulls,access date : 10-10-2019
9. Realm-java homepage,https://realm.io/docs/java/latest,access date : 10-10-2019
10. Realm-java library-build-transformer, https://github.com/realm/realm-java/tree/master/library-build-transformer
11. Realm-java main classes,https://github.com/realm/realm-java/tree/master/realm, access date : 10-10-2019
12. PowerMock,https://www.ibm.com/developerworks/cn/java/j-lo-powermock/, access date : 20-10-2019
13. Realm-java libraries.io,https://libraries.io/github/realm/realm-java, access date : 20-10-2019

14. Brief introduction about software architecture,https://www.cnblogs.com/kexinxin/p/11518954.html, access date : 27-9-2019
15. Demo about the use of realm-java,https://github.com/realm/realm-java/tree/master/examples, access date : 20-10-2019