POLITECNICO DI TORINO

Master's degree
in Communications And Computer Networks Engineering

Master's Degree Thesis

Review and testing of plugins in Flutter for Android and IOS

Supervisor
prof. Guido ALBERTENGO

Candidate
Yusi WANG

March 2022

Summary

With the advent of the 5G era, the development and design of mobile applications have become an important development direction of widespread concern in various fields of the Internet, and the demand for mobile application development has also grown rapidly. Among the various development cross-platform frameworks for mobile development, flutter is the best; it is currently the most popular and easiest to use cross-platform development framework. In addition, selecting appropriate plugins in developing mobile APP functions can effectively shorten the development cycle. Therefore, in order to better develop and improve development efficiency, the review and testing of plugins in Flutter for Android and iOS have become particularly important.

To explore this process, The thesis uses the Flutter cross-platform framework to develop and design a smart application. After understanding the background of Flutter and analyzing and studying its related technical principles, according to the most common and representative native functions of current mobile phones. The functions of mobile phone photo taking, photo album, calling, sending a short message, QR code scanning recognition, QR code picture recognition, and RFID tag NFC recognition are respectively realized.

The application interface is simple, clear, and beautiful, and the operation is simple and convenient. The overall UI framework uses Dart UI for layout. Through to the selection of various functional plugins, the addition of plugin dependencies, and the reference of plugins, the perfect combination of plugins and UI functional interface is finally realized, the exploration of various native function application plugins is realized, and a good display is obtained in the mobile phone test. The effect perfectly meets the expected requirement.

Keywords

Flutter, plugin, cross-platform, mobile app.

Acknowledgements

Time flies, my studies at the Politecnico di Torino are coming to an end, and I am entering the next phase of my life, starting work. Looking back on these two years, the most important and correct decision I made was to choose Italy. In such a historic and beautiful country. At Politecnico di Torino, I completed my master's degree in a friendly academic atmosphere. In addition, the time at the Politecnico di Torino helped me a lot in my studies and life. This is the first time I've faced and solved life's problems by myself, living thousands of miles away in another country. This experience has greatly exercised my ability and enriched my learning style and thinking.

There is a Chinese proverb: It takes ten years to plant trees and one hundred years to raise people. I want to thank my supervisor, Professor Albertengo. Whenever I ask him for help with questions, he will reply quickly and help me solve my confusion; Professor Albertengo's serious and responsible work attitude and rigorous academic style are the most worthy of my study. These excellent qualities will become a valuable asset in my life. Secondly, I would like to thank Professor Garello for recommending a professor to me based on my situation when I didn't know what topic to choose as a research topic. And help me set goals. I would also like to thank the CCNE students, Ali, Chang Yushuo, Enrico Analolo, Wang Keyu, Li Dawei are a group of lovely and hard-working people; they have given me a lot of encouragement in study and study life. Help me move forward. Finally, I would like to thank my family, who gave me strong backing and let me move forward fearlessly!

Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Research background of the subject

In today's society, people's demand for mobile Internet is increasing day by day [1]. According to statistics, in China, each mobile phone user has downloaded more than 60 apps on average, and the average usage time of apps is as high as 5 hours. There are as many as 3 million apps in the app store, covering all walks of life and different types. Also, developers release new apps on the app store every day. According to media reports, Apple's app market value in China is close to $260 billion. Therefore, more and more developers are devoting themselves to developing and designing mobile applications. The first step in developing a new application is to design the front-end interface. A beautiful and high-quality user interface will increase users and play a key role in improving the user experience. In addition, in the release of the app, since our existing mobile phone market has two systems, namely iOS and Android, two sets of codes are often required to adapt to different system environments in the development of the app, which undoubtedly greatly improves the app development cycle—increased developer workload. In addition, APP also needs to configure many functions for users to improve the user experience. Therefore, developers will use different plugins to achieve the above requirements, so how to choose a suitable plugin, whether the plugin is easy to use, and whether the functions encapsulated in the plugin can be used normally. All need to be reviewed and tested by us one by one.

Therefore, in order to solve these common problems and meet the needs of developers, Google's technical team has developed a new technical framework called flutter, which developers can easily and conveniently use to build high-quality and beautiful user interfaces. A set of code can run on both iOS and Android systems, which greatly improves the work efficiency of developers and reduces the maintenance cost of the code. On top of that, it also has a rich plugin marketplace.

Open source projects related to developing flutter plugins on GitHub are very active, which can help us choose and test plugins. In short, to develop an app, the most popular and best technical framework is flutter, and developing various functions for the mobile terminal in flutter is essentially the selection and application of different plugins. Therefore, exploring review and testing of plugins in Flutter for Android and IOS is selected and designed in the context of such a topic

1.2 Research on flutter and related native plugins

Flutter provides an SDK to easily compile source code to code for Android or iOS [2], and a library of widgets, functions, and packages to customize the graphical interface [3].

Flutter is a cross-platform mobile application development framework launched by the Google R&D team [4].Flutter provides an SDK to easily compile source code to code for Android or iOS [2], and a library of widgets, functions, and packages to customize the graphical interface [3].This way, developers can quickly build high-quality native user interfaces on iOS and Android. And greatly reduce the workload of developers for code maintenance [5]. Since the release of Flutter beta1 in February 2018, Flutter has been updated and iterated hundreds of versions in just four years. The latest Flutter version is 2.10.2 released on February 19th [6], which is enough to show that Google attaches great importance to this mobile development framework.

At the same time, the development speed of Flutter on GitHub is also quite amazing, with more than 127k stars [7]. At present, the popularity and attention of Flutter have surpassed the well-known cross-platform framework React Native, and it has become the most popular technical framework. In the industry, many Internet companies choose to use Flutter as a development tool, such as eBay, Google, ByteDance, Tencent, BWM [8]. And Flutter has many successful cases, such as the Beike app, Xianyu app, Google Pay, Abbey Road Studios, and so on. Therefore, more and more engineers use Flutter as a development tool, and the income of Flutter technology practitioners is also very considerable. This mechanism forms a good positive feedback mechanism between Flutter and practitioners. That is to say, the more practitioners, the more perfect the technology, to optimize the flutter framework and make Flutter easier to use and more powerful. Internet companies have also increased the demand for talent from developers who master Flutter technology. In order to win the talent competition, the company has once again increased the income and treatment of Flutter developers. Also, the future of Flutter is very bright. The technical teams of various Internet companies have carried out experimental research on Web desktop and embedded platforms. In

addition, Google also plans to apply Flutter technology in Fuchsia in the future.

Also, with the rapid development of flutter is the plugin market that relies on flutter. Pub.dev provides developers with many plugins, including more than 20kpacket plugins; developers can find most of the functions you want to design for the application and then use the functions packaged in the plugin to implement the application. For plugins, this A concept was first proposed by H.Simon in the 1960s. He believes that plugin modules are special frameworks that are dynamically balanced in the evolution of complex systems [9]. The plugin development method is an architectural model rather than a conventional general technical standard in an application development method. In the development process of an application, "module" is just a logical concept. The entire application is divided into independent host applications that are independent of each other, and multiple functional modules outside the application are plugins [10]. The basic principle of plugin implementation is to identify itself by implementing an extension contract (usually an interface) specified by the main program and receive event responses from the main program. This process of interacting with the main program is realized by mobilizing the services provided by the main program.

1.3 The content and significance of the research topic

This thesis mainly analyzes, researches, and tests how to implement mobile phone functions on the mobile terminal of smartphones through plug-ins in applications with Flutter as the main technical framework. The main feature of the application is that it is easy to operate, and it can provide various basic functions such as taking pictures, texting, calling, photo albums, etc., and can scan and identify QR codes and RFID tags. It is a smartphone application that effectively provides various functions to facilitate our lives. The application adopts the Flutter cross-platform solution as the basic framework for using various plug-ins, solving user needs, and implementing business application scenarios. Flutter has significant advantages over other cross-platform frameworks. The Flutter framework is very different from traditional native development. It can be implemented as a set of code to run on Android and iOS devices and uses self-rendering to maximize performance. In order to develop with Flutter better, you need to have a good understanding of Flutter. Its rendering principle, asynchronous loading mechanism, and communication principle with native are all essential in the application implementation process.

The overall architecture of the application adopts layered architecture design and module development. The function is divided into six modules with a hierarchical design. Realize projects at a glance, effectively discover and test problems,

and speed up development. After the system infrastructure is completed, business layer development is relatively easy. Low correlation and independence between business layers. They are mainly done by flutter components and depend on the underlying implementation. The implementation of each module of the business layer will design the component structure according to its business scenario and interface layout, minimize the layout level, and improve the smooth application experience. In the application development process, the compatibility, stability, and scalability of the system will be fully considered. In order to ensure performance , the problems encountered in the development process will be continuously adjusted and optimized, and the test will be carried out after completion to ensure that its functions can be perfectly reproduced on the mobile phone.This thesis's main significance is to explore the principles of Flutter-related technologies, the realization of Flutter framework application functions, and the entire process of reviewing, testing, and implementing plug-in applications. To provide technical support and theoretical understanding for developers who will browse the paper in the future. It is hoped that developers can have a basic and clear understanding of Flutter plug-in development and testing after browsing the thesis.

1.4 Thesis structure

This article is divided into six chapters, from the background of the topic selection, the original intention of the development, the technical principle to the design implementation, and the final test. It gradually introduces the whole process of project completion. The chapter summaries are as follows:

Chapter 1, Introduction: This chapter mainly introduces the background of the topic selection, analyzes the research status of flutter and flutter plug-ins, and expounds the research content and significance of this topic.

Chapter 2, Flutter-related theory and technology research: This chapter focuses on the Flutter architecture that implements the app, and focuses on the analysis and introduction of several main basic knowledge points of the system architecture. Including the architecture and advantages of Flutter, the rendering principle of Flutter, the asynchronous and threading of Flutter, and the communication principle of Flutter and Native.

Chapter 3, Flutter-based native plug-ins explore the overall design of the application: This chapter mainly introduces the preparations before development, including analysis of business and non-business requirements, overall system design, and project directory structure design

Chapter 4, the key technology application implementation of flutter-based native plug-ins in the mobile terminal: This chapter mainly introduces the implementation of each module of the flutter client and conducts an in-depth analysis

of the technical difficulties.

Chapter 5, Plug-in-based review and testing of research effects: This chapter mainly introduces how to conduct effective testing after development is completed to ensure the stability of the application and the final online deployment.

Chapter 6, Summary and Outlook: This chapter mainly summarizes the projects and articles and gives a good outlook on the follow-up research content and the development of Flutter.

Chapter 2

Flutter related theory technology and research

At present, the big front-end technology is constantly breaking through, and various technologies also appear in our field of vision. Flutter is one of them. It can not only quickly build beautiful application interfaces on Android and iOS systems but also make Flutter closer to native applications in terms of performance by using self-rendering methods. This chapter will mainly introduce the relevant theoretical technologies applied by Flutter in the mobile terminal and provide theoretical support for subsequent project design and application difficulties.

2.1 Flutter system

Flutter is a mobile application SDK that can generate applications with superior performance and support running on multiple platforms such as iOS and Android [11]. Since the first beta version of Flutter was released at Mobile World Congress in early 2018, its purpose has been to quickly build high-quality and beautiful native user interfaces on iOS and Android while maintaining high frame rates for continuous rendering. Flutter's version updates are very fast. Through the vigorous promotion of the framework by the Google R&D team, more and more developers have begun to try to use Flutter as the basic development framework. The Flutter community is also very active. GitHub-related projects developed by Flutter are also getting more and more attention. Due to its obvious advantages, development engineers from many large Chinese Internet companies such as ByteDance, Alibaba, and Tencent are also actively researching and using Flutter. BeikeApp, XianyuApp has also achieved good results, greatly saving the development cycle and maintenance costs.

2.1.1 Flutter Architecture

Flutter is developed using Dart [12]as the main programming language. Its experience effect is close to native applications, but strictly speaking, it is not a real native application framework. It has to be built in the native system. In fact, cross-platform is to embed Flutter's engine and superstructure into each platform system. Flutter's layered architecture consists of three layers. Each layer is independent of the other and depends on the structure of the next layer. Each part of the framework is optional, or replaceable [13]. The system architecture diagram is shown in Figure 2.1.

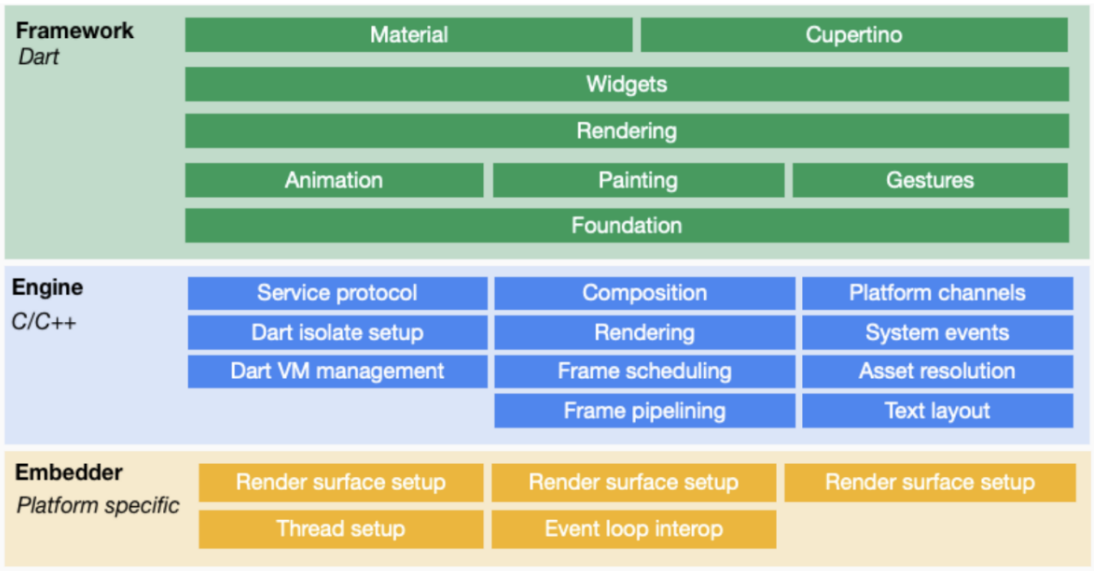

Figure 2.1. Flutter system architecture diagram

The first is the Framework layer, the top layer in the Flutter architecture. It is implemented by the Dart language and implements some basic components , such as UI, text, pictures, buttons and other components, rendering, animation, gestures, etc.

The second is the Engine layer in the middle, which is the core of the Flutter. It is mainly an SDK implemented in C++, which provides API calls for all UI libraries in the framework layer above, mainly including a 2D graphics rendering and text layout and Dart runtime engine, and is also a bridge system between the framework and the application.

The bottom layer is the Emberred layer. Flutter mainly embeds itself into other platforms to achieve cross-platform solutions through this layer. Other platforms don't need to be responsible for rendering. Just provide a canvas. The engine

provided by Flutter does the rendering logic. In addition, thread settings, Surface settings, and plug-ins are also the main work that Flutter does in this layer.

2.2 Advantages of Flutter

By comparing other technical solutions, choosing Flutter as the technical framework for mobile applications is mainly based on the following four aspects.

(1) Flutter can easily achieve cross-platform operation; only one set of code can run on Android and iOS systems and will support running on the Web and PC in the future. This can not only maintain the consistency of work, but also greatly reduce development costs, save testing time, and improve work efficiency.

(2) The performance of Flutter is much better than other frameworks, mainly in two aspects. First of all, because the coding uses the dart language, Flutter will choose the JIT mode during the development phase, which can effectively avoid recompilation for each change and the running speed is not slower than JavaScript [14]. Secondly, in the release stage, Flutter can generate efficient ARM code through AOT to ensure running performance [15], which is very helpful for view data calculation at a high frame rate. Second, Flutter's rendering work is done by its own engine, so there is no need for native rendering through an intermediate layer like React Native. Reduced performance loss, enabling high frame rate rendering of UI, making Flutter performance and experience closer to the native effect.

(3) Flutter is easier to learn, the layout components are displayed in a tree structure, and Able to achieve complex gesture animation click effects with simple principles. Developers can flexibly apply the components that come with Flutter and quickly build beautiful user interfaces by combining them. In addition, the Flutter topic of the technical forum is very active, and it is easy to find peers to learn together. In addition, there are rich open-source projects and learning documents to help developers refer to the learning process.

(4) Comparison between Futter and other cross-platform technologies as shown in Table 2.1.

2.3 Flutter rendering principle

At present, the mobile phone market's mainstream mobile phone refresh rate is 60Hz, which means that the mobile phone screen will be refreshed 60 times per second. During refresh, the monitor sends a Vsync signal every time a frame is drawn. Based on the aforementioned 60Hz calculation, the monitor will emit a total of 60 Vsync signals. The data transmitted by these signals are sent to the CPU for centralized integration processing, and display content is obtained

Table 2.1. Flutter vs. other cross-platform technology table

Type of technology	**React Native**	**Weex**	**Flutter**
R & D company	Facebook R&D	Alibaba R&D	Google R&D
Core language	React	Vue.js	Dart
Features	Developing overall App	single page development	Developing overall App
difficulty	Normal	Easy	Normal
Design Patterns	React Design Patterns	Vue Design Patterns	Responsive Design patterns
performance	Normal	Normal	High
Development efficiency	High	High	Extremely high performance
Engine	JSCore	JSV8 Engine	Flutter Engine
Ui rendering method	Native control rendering	Native control rendering	self-rendering
Support platform	Android/iOS	Android/iOS/Web	Android/iOSWeb
Degree of framework	Heavy	Light	Heavy

through calculation, and then this part of the content is transmitted from the CPU to the GPU, which is then rendered by the GPU and then transmitted to the display. Until we observe, Flutter's rendering process is shown in Figure 2.2.

First, the GPU will send the Vsync signal to the UI thread in this process. After the UI thread is synchronized to the signal, the dart language will process the signal at the application layer to construct abstract view structure data. This part of the data will then be sent to the GPU thread, where layer synthesis is performed, and after synthesis, it is provided to skia for rendering to form GPU identifiable data. Finally, the whole rendering process is completed by providing it to the GPU through OpenGL or Vulkan.

The most important link in the rendering process is to construct the view structure data. In constructing the view structure data, three tree structure elements play an indispensable role in it. They are the Widget tree, the Element tree, and the RenderObject tree. The relationship between these three is shown in Figure 2.3.

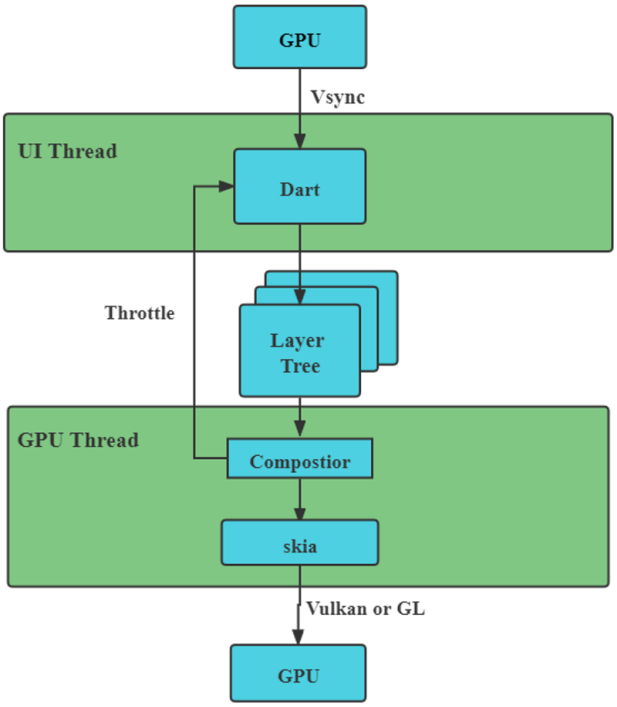

Figure 2.2. Flutter rendering flow chart

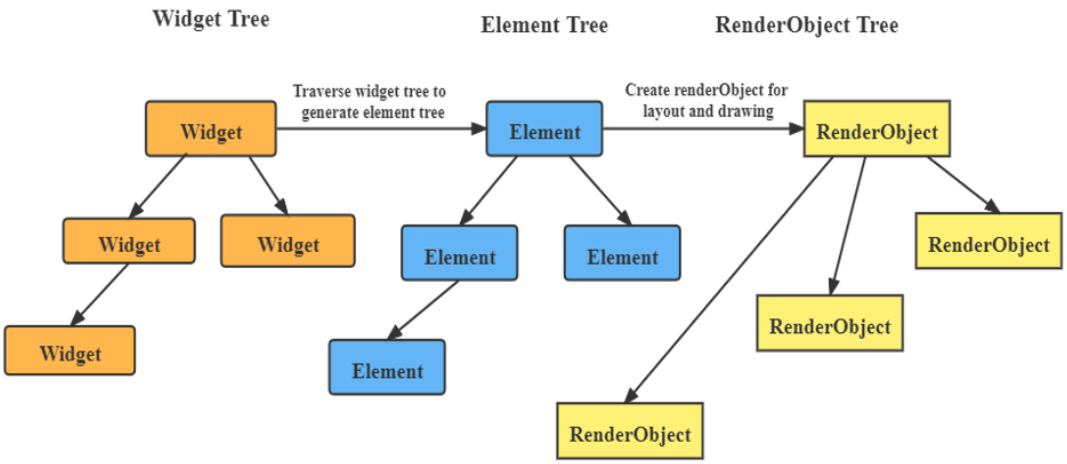

Figure 2.3. Widget, Element, RenderObject Relationship Diagram

Widgets are the heart of Flutter. It stores a lot of view configuration information, such as properties, layout, etc. It is a configuration file and does not directly participate in the drawing, so when developers frequently conFigure or delete it, it will not affect performance. There are two forms of Widget as a whole, one is StatelessWidget, and the other is StatefullWidget. The difference between them is that a stateful widget can refresh the state after calling the setstate function, while a statelesswidget cannot. In fact, their essential characteristics are the same, they are immutable components, and each frame change needs to be reset. Just a statefulwidget can store variable state in it by using the state.setstate() function. Whenever the widget tree is refreshed and changed, it only needs to call the function to achieve the new effect. The life cycle diagram of the state is shown in Figure 2.4.

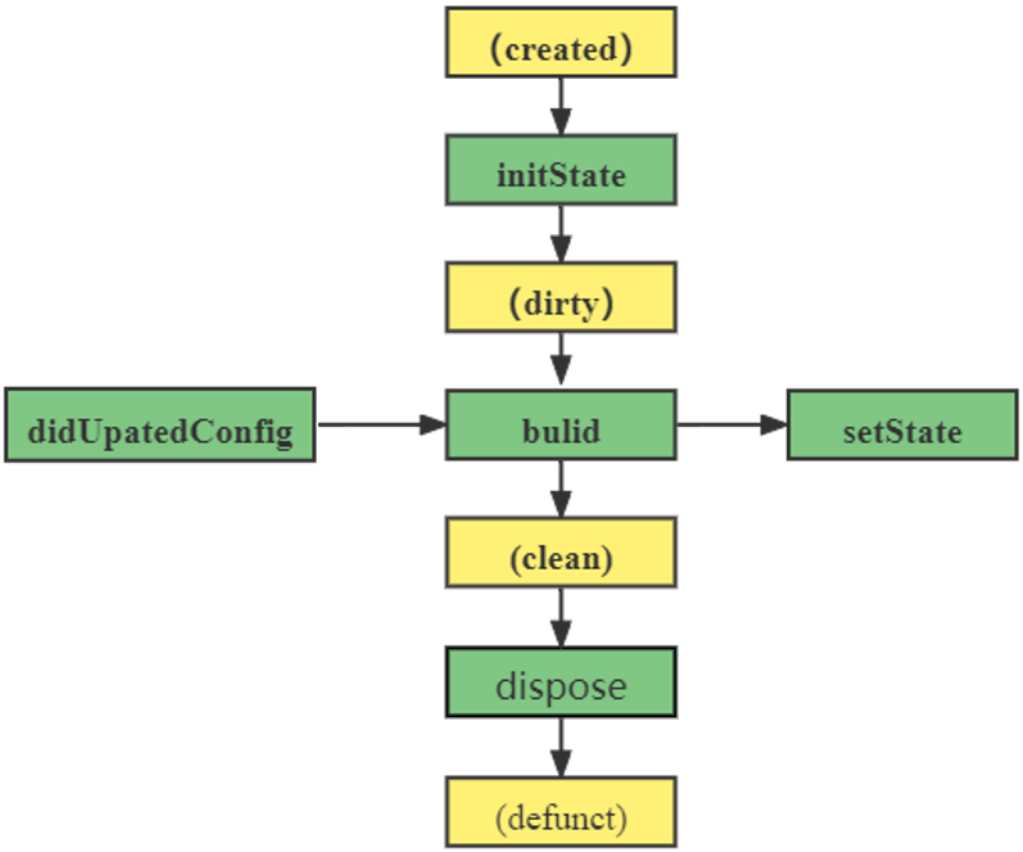

Figure 2.4. State Lifecycle Diagram

The element tree is generated by the widget tree [16], which acts as a bridge connecting the widget tree and the RenderObject tree, and plays the role of management and scheduling. Because widgets are very unstable, they may rebuild repeatedly. If you render directly, it will consume a lot of performance. So Flutter creates an element for the widget. Each widget corresponds to an element, and the element will have a unique key. When the widget needs to be updated, there is no need to re-render the entire widget; just synchronize the corresponding part of the element to the real RenderObject tree, which greatly reduces the modification of the real rendering and improves the rendering efficiency.

RenderObject is responsible for rendering and layout in Flutter. Unlike the one-to-one relationship between widget and element, only the widget that needs to be rendered will have a corresponding RenderObject node. Whenever the widget is modified or changed, the RenderObject will compare the difference between the new widget and the element to see if the reserved type and key are the same as the previous ones. The properties will be updated if they are the same, which greatly saves rendering overhead.

Four important properties and methods in RenderObject:

- constraints: constraints passed from parent.
- parentData: This carries the data used by the parent when rendering the child.
- performLayout(): This method is used to layout all children.
- paint(): This method is used to paint itself or child [17].

Next, will introduce the three sub-steps of Flutter Widget rendering: layout, drawing, and synthesis. Flutter optimizes each step. In the layout process, by setting layout boundaries for nodes, when the component is rearranged, it will not affect the outside world. For the drawing process, the redrawn boundary is also set. To better explain the principle, simply draw a schematic diagram 2.5 here.

As shown in the Figure, when the 2-node part needs to be redrawn and affects the 6-node, the 6-node will be switched to a new red layer. Only the former will be drawn to avoid redrawing affecting 6 -nodes and improve the Redraw efficiency.

For layer synthesis, Flutter adopts a dynamic texture scheme (a layer is automatically cached as a texture after being redrawn three times). Flutter thinks that this layer is very likely if a layer is drawn three times. It will be drawn for the fourth time or more. At this time, it will generate a texture for this layer. When it is drawn again, it only needs to project the texture on the screen.

The whole rendering process is summarized below. Widgets are first written to form a widget tree in the development process. Each widget will generate a corresponding element, and the element also obtains a unique key. When we use a

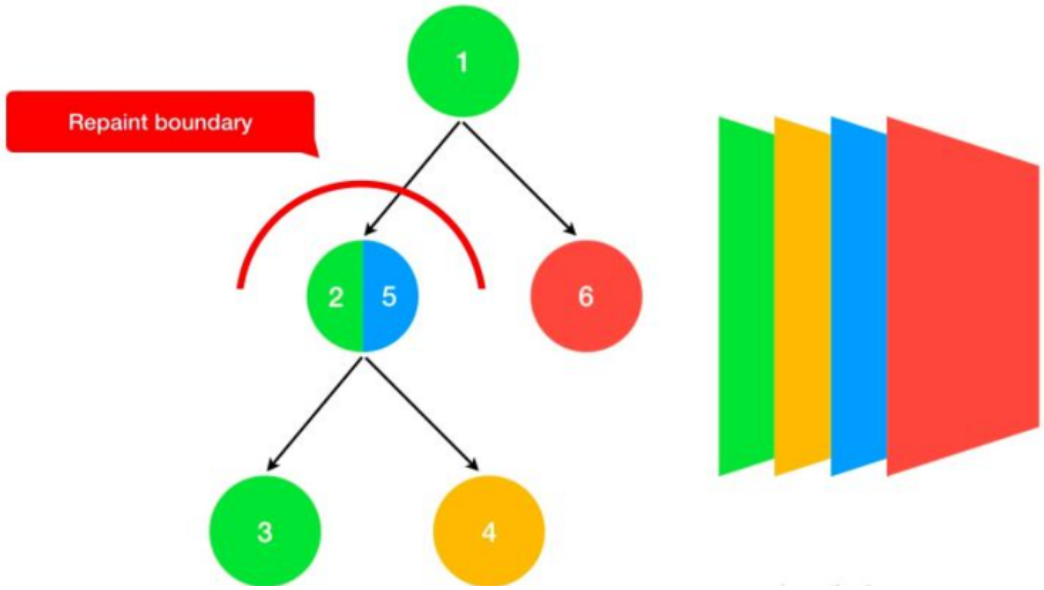

Figure 2.5. Performance optimization (drawing) diagram

statefulwidget, A state() function will be generated, and it will also be passed to the state in the element. Then the element that needs to be rendered will generate a RenderObject. When the widget changes, the corresponding element structure will also send changes. The node that changes will be marked as dirty, and the node will trigger an update in the next cycle. At this time, the latest widget will be associated with the corresponding RenderObject,Layout and drawing happen here. Skia then takes over the rest and renders the image to the GPU for display.

2.4 Asynchronous and threading of Flutter

Asynchronous functions were developed to utilize better the power of computers, which are very fast compared to human operations. It's a lot of waste if the computer is blocked, so async functions help your program do other work while waiting for your instructions to do the next thing. Greatly improved work efficiency.

Dart is the main programming language in Flutter. Like JavaScripts, it doesn't have a multithreading model. It needs to use the concept of Coroutine to achieve asynchrony. Unlike threads, the dart can use isolates to achieve multi-threaded effects; each isolate has its own memory space event loop and event queue. And isolates are independent of each other, do not share the memory, and only pass information through ports. Since there is no need to concurrently request access locks like threads, there is no deadlock phenomenon. Isolate provides a solution

for Dart applications to take better advantage of multi-core hardware.

Flutter's asynchronous application is mainly accomplished by maintaining an event loop and two event queues (microtask queue and event queue) in dart. The specific flow chart is shown in Figure 2.6.

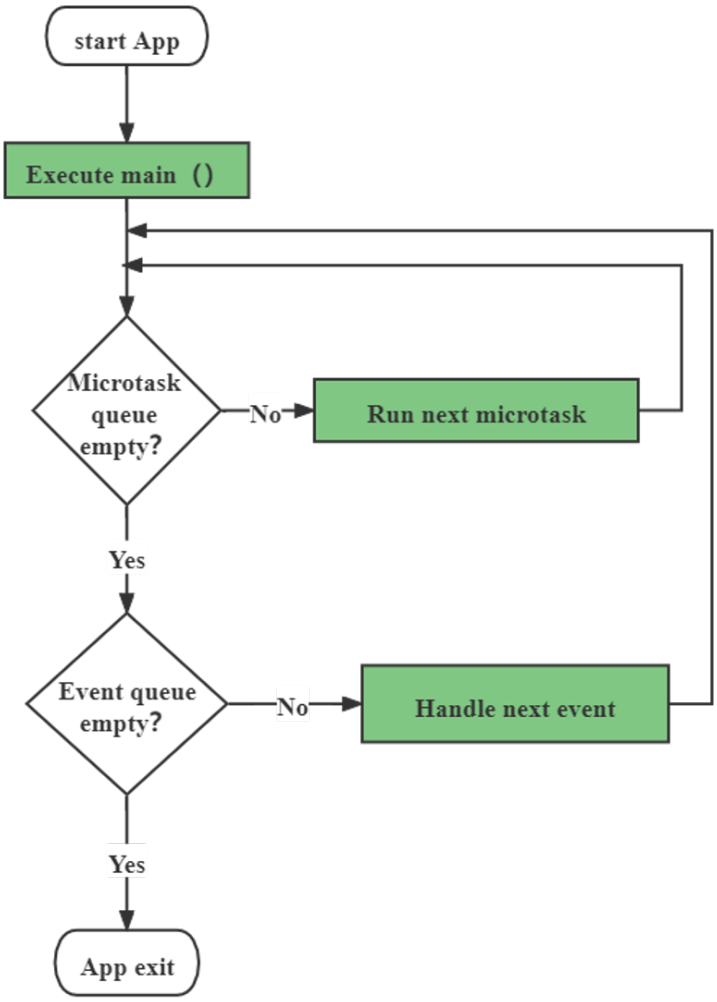

Figure 2.6. dart asynchronous loading flow chart

The order of execution is:

1. Execute the task in the main function.
2. Check if there is a task in the microtask queue. If there are tasks to be executed first, this provides a solution for dart tasks to cut the queue. It is worth noting that if there are too many micro-tasks, the event task will be blocked, resulting in the app being unable to perform UI rendering click events, etc.
3. When the microtask queue is idle, check the event queue, and if there is one, execute it in sequence.

2.5 Communication between Flutter and native

Developers develop some system functions on the mobile APP, such as calls, text messages, photos, Bluetooth, etc., which need to be obtained from native methods. Although there are many plug-ins in the plug-in market that encapsulate these functions, App functions can be implemented by accessing plug-ins. But in essence, the internal implementation principle of these plug-ins is completed through the communication between Flutter and native, so learning the communication principle between Flutter and native is particularly important for developing projects and optimizing performance.

Communication between Flutter and Native relies on platform channel. There are three types of channels, as shown below:

- BasicMessageChannel: Mainly used to pass strings and semi-structured data.
- MethodChannel: It is more common to pass method calls.
- EventChannel: Used for data stream transmission, with monitoring function. If there is a need to send data from the native back to the Flutter side, this channel is the most suitable.

Specifically, Flutter uses channel and native to pass information through MethodChannel. When we initialize a channel, a MessageHandler is generated. A unique key identifying the corresponding channel string is stored on the HashMap, so each MessageHandler corresponds to channels. When Flutter sends a message, the BinaryMessenger will find the corresponding message channel, process it through different Codecs, and send it to the MessageHandler. After the MessageHandler has processed the data information, it will return in the same way to complete the communication between messages. Two-way transmission. Its communication flow is shown in Figure 2.7.

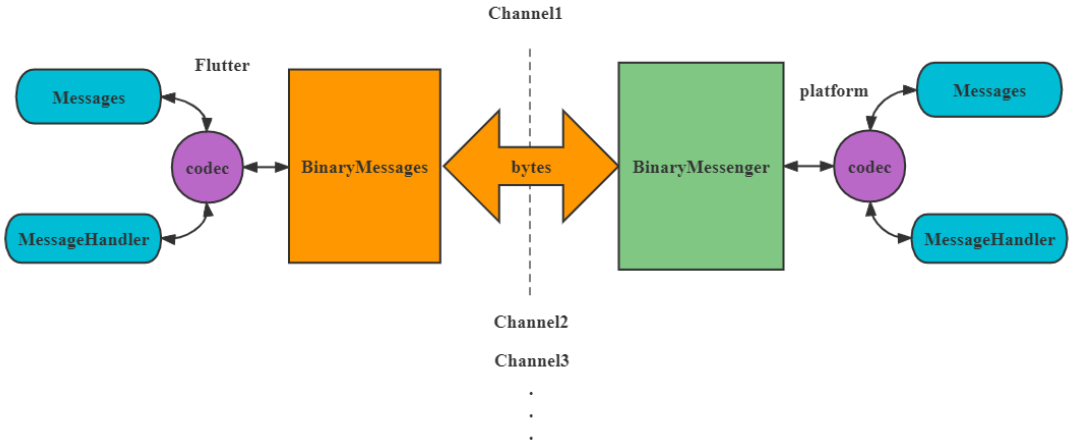

Figure 2.7. Flow chart of communication between Flutter and native

2.6 Chapter Summary

This chapter first introduces the Flutter system, followed by a technical analysis of the Flutter framework, expounding the advantages of choosing it as a project and focusing on the key technologies that will be applied in the project. In the development of Flutter, it is essential to understand the rendering principle, which will greatly improve the rendering effect. In addition, the asynchronous operation mechanism and isolate of the single-threaded model of the dart language are also very different from the general multi-threaded concurrency. In-depth research on this part of the principle will effectively improve our development efficiency and optimize the performance of our project. Of course, the review and testing of plug-ins are also inseparable from the communication between Flutter and native. Understanding these contents will lay a solid foundation for the subsequent development work.

Chapter 3

Discuss the overall design of the application based on the native plugin of Flutter

This chapter will start with Flutter's native plug-ins, explore application requirements and overall conceptual design, sort out the functional and non-functional requirements of each module, analyze the overall project structure and directory structure in detail, and ensure that the application can efficiently complete various functional requirements so that Flutter native plug-in application can successfully pass the expected test.

3.1 Demand Analysis

Based on the cross-platform features of Flutter, Flutter and the corresponding native plugins can be used more efficiently in the future development process. It is necessary to be familiar with its essential principles and deeply explore the application of native plugins. According to the functions often used in daily life, the corresponding native plug-ins can be integrated and called to realize the corresponding function design and complete the debugging and testing work. Design the corresponding requirements.

3.1.1 Business functional requirements

In order to explore the application of Flutter native plug-ins, the following functional requirements are proposed according to the typicality and universality of the required plug-ins:

(1) Camera, photo album plug-in application: In the context of the rapid development of the mobile Internet and the blessing of 5G networks, most smartphones are equipped with high-definition cameras. Therefore, many APPs can upload user avatars and store pictures. Users can call the camera to take beautiful photos or open the album to select favorite images to upload. Cameras and photo albums have almost become necessary skills for APP development.

(2) Calling, SMS plug-in application: Calling and texting are essential functions in everyday life. It has important applications in multiple scenarios such as app new user registration, member login, password retrieval, payment confirmation, system notification, marketing campaign access dial-up, user authentication, and member notification reminders. Therefore, we must explore and develop these two functions.

(3) QR code scanning plug-in application:With the increasing number of mobile Internet users, QR code, as brand-new information storage, transmission, and identification technology, is used in many aspects such as information acquisition, website redirection, advertisement push, anti-counterfeiting traceability, mobile payment, account login, etc. Therefore, the exploration of Flutter QR code-related function plug-ins is also representative.

(4) NFC plug-in application : NFC is a powerful builder of a digital and intelligent society. With the development of smart cities and smart transportation, NFC technology devices can exchange data when they are close to each other. Through point-to-point, card reader, analog card, and other modes, complete mobile payment, electronic ticketing, access control, mobile identification, anti-counterfeiting, and other applications. Therefore, it is necessary to explore the NFC plug-in.[18, 19, 20].

3.1.2 Non-functional requirements

Flutter's native plug-in application exploration mainly selects several typical requirements and functions such as camera, photo album, phone call, SMS, QR code, NFC, etc., for development and elaboration. The plugins developed by Flutter based on native development are hosted in the plugin application market pub.dev. Each developer can upload and host plugins that comply with the specification. Therefore, in the development process, non-functional requirements such as convenience, efficiency, compatibility, security, scalability, and maintainability of plug-ins will be involved.

(1) Convenient and efficient: Everyone has their way of thinking about problems. Different people may have different solutions to the same problem, resulting in differences. In the vast plug-in market, a plug-in with a specific function may also have plug-ins with different solutions, so the convenience and efficiency of plug-ins also require our attention.

(2) Compatible security: Flutter is an efficient and popular language, and it is also in the process of rapid development and iteration. Therefore, different versions have different compatibility. Depending on the developer's development environment, different plug-in versions are required. If the version does not match, it will directly cause the program to report an error and fail to run. Therefore, the compatibility and security of plugins are more worthy of our attention.

(3) Scalable and maintainable: The development of a project needs to go through long-term maintenance iterations, so it has high requirements for its stability. The stability of the software refers to the abnormal phenomenon that will not be affected by user operations, network conditions, and the increase in the number of users [21]. Therefore, it will involve the iteration of different functional requirements versions. Good scalability and maintainability can produce efficient work results in subsequent continuous iterations, and it is also convenient for us to respond to different business needs quickly.

3.2 Overall system design

Based on the Flutter native, plug-in exploration application, the overall functional layout is constructed with pure Flutter Widget widgets, involving the home page function display, including the design of several functional areas such as photography, photo album, dialing, SMS, QR code, and NFC.

The application mainly adopts a general hierarchical organization model, and the whole consists of three layers, namely the business layer, the component layer, and the basic system layer. Each layer is independent and mutually dependent.

Business Layer

It mainly includes visual UI functions that directly interact with users, mainly implemented in the Futter programming language Dart, including the icons, titles, backgrounds, colors, shadows, rounded corners, gradient implementations of each function button, and the jump pop-up window of each pop-up page window. Interaction logic.

component layer

According to convenience, efficiency, compatibility, security, scalability, and maintainability, select the exploration target plug-ins corresponding to dial-up text messages, photo albums, QR codes, and NFC-related functions. This layer mainly depends on the underlying native environment. The difference between platforms is to achieve corresponding functional consistency through native communication between Flutter and native. After encapsulating the reference, call the functional business layer directly.

Base system layer

The basic system layer is the underlying architecture layer of Flutter, and it is the basic support for the program to run across platforms. Mainly implemented by native, relying on native system services. Through this layer of conversion, the consistent conversion of compilation and running differences of different system platform characteristics is realized. The Flutter native plug-in explores the application layered architecture as shown in 3.1.

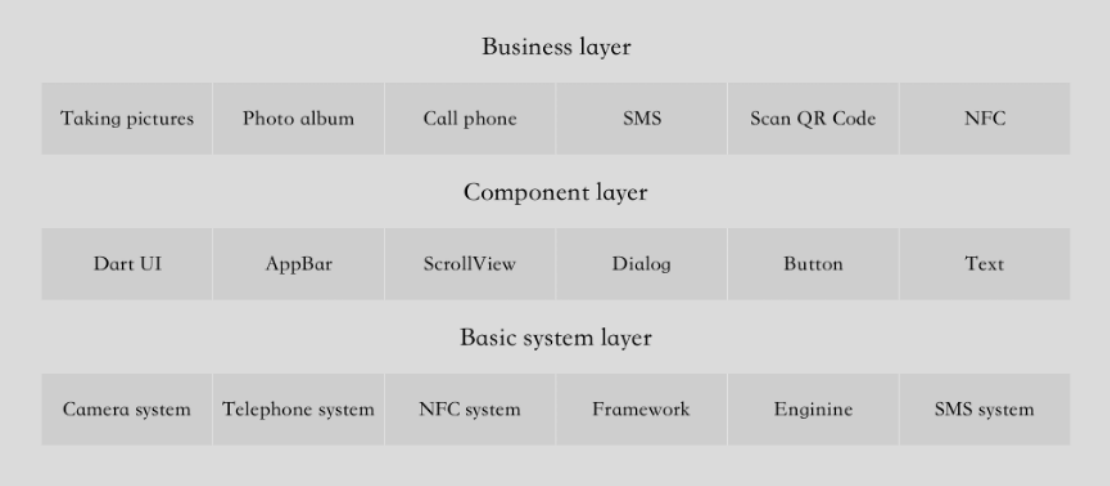

Figure 3.1. Layered architecture of Flutter native plug-in application

Flutter is divided into two parts, and C++ implements the underlying Engine. Dart implements the upper framework.

Engine provides a complete operating environment for the Framework. Framework is divided into four layers, from bottom to top, respectively.

1. Foundation,
2. Rendering,
3. Widgets,
4. Material.

The Foundation layer is implemented by dart:ui. dart:ui provides the most basic functions that the Framework can run, such as the raw information of events such as drawing, interface refresh, touch screen, and mouse.

The rendering layer consists of several submodules: animation, painting, gestures. Flutter provides RenderObject to implement complete layout and drawing functions at this layer.

The Widgets layer is the layer that developers touch the most. Widget is the encapsulation of RenderObject. At the Widget layer, Flutter implements a responsive development framework.

Material+Cupertino layer, in this layer, Flutter provides a series of Widgets, of which Material Widget implements Material Design. Cupertino provides a set of iOS-style controls [22, 23].

3.3 Project Directory Structure Design

As a cross-platform development language, Flutter's project directory structure also has specific cross-platform features. The project contains not only its own dart language-related code but also native platform-related Android and iOS and Web-related code. The directory structure of the Flutter program can help us get started quickly, understand the cross-platform features of Flutter, and is also conducive to daily development and debugging. The Flutter project directory is shown in Figure 3.2.

The Flutter project directory structure description is shown in the following Table 3.1.

The more important folders and files are Android, iOS, lib, test, pubspec.yaml:

- lib: The dart language code we develop daily is placed here; it can be said to be our "core working folder".
- iOS: This contains the configuration and files related to the iOS project. When our project needs to be packaged and launched, we need to open the Runner.xcworkspace file in this file for compilation and packaging.
- android: The same as the iOS folder. When the android project needs to be packaged and put on the shelves, the files in this folder need to be used. Similarly, if we need native code support, native code is also placed here.
- test: Here, we store our test code in the process of project development. Good testing habits are a necessary means to ensure code quality!
- pubspec.yaml: The core configuration file of the Flutter project, including the project name, version number, project description, homepage, development documentation, and most importantly, the project dependency plugin configuration.

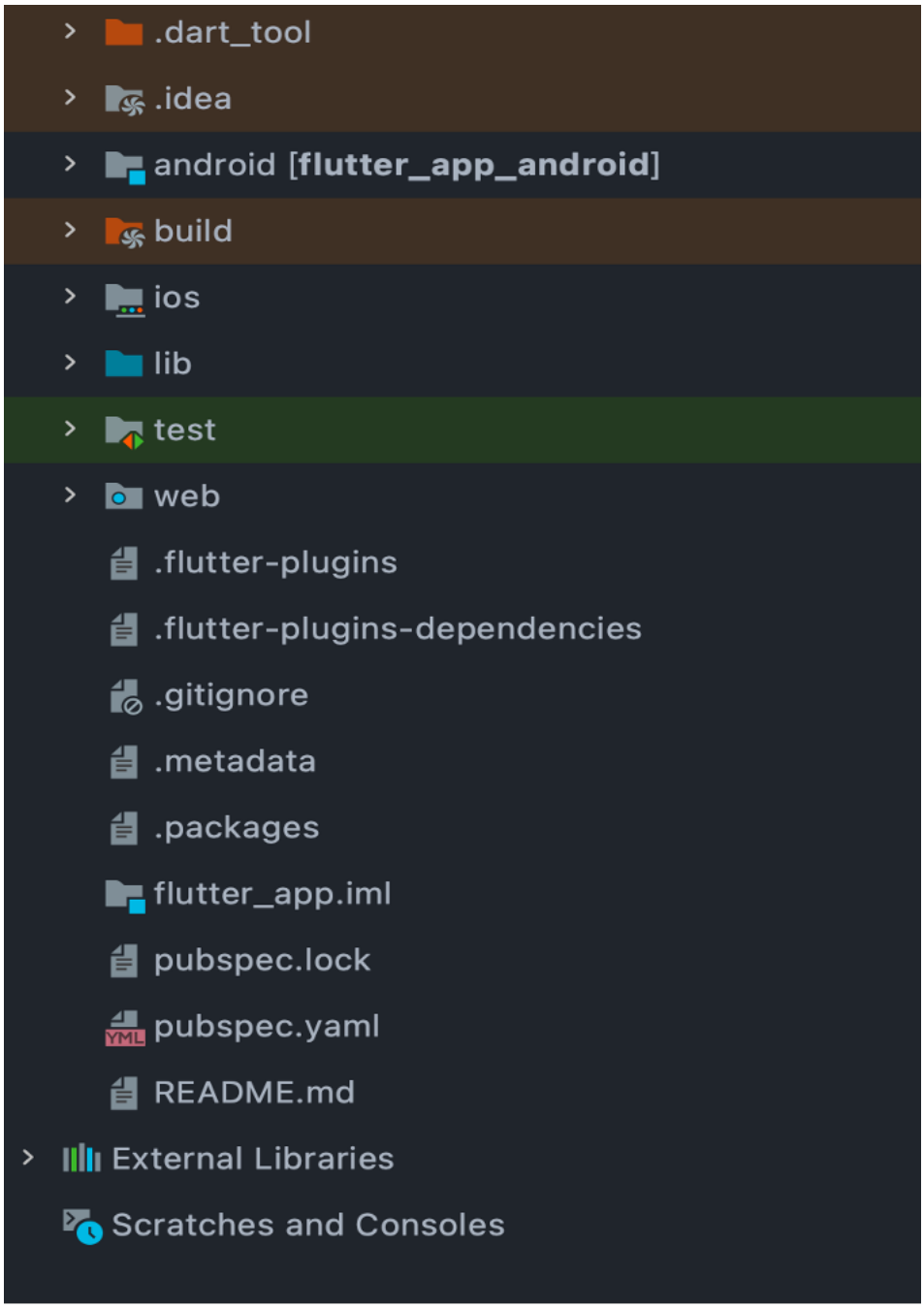

Figure 3.2. Schematic diagram of the Flutter project directory

Table 3.1. Flutter project directory structure description

file/directory	**role**
dart_tool	Dart tool development kit
.idea	development environment configuration
android	Android native project file
iOS	iOS native project file
build	Compile or run the product
lib	contains project related files ending in .dart
test	contains project test files ending in .dart
web	web native project file
.gitignore	git commit repository ignore files
.metadata	A configuration record for the current workspace
.packages	Absolute paths to files ending in lib
pubspec.lock	file generated before project dependencies
pubspec.yaml	dependency configuration
README.md	readme project information (html tag)
External Libraries	Android rack package and resource files, Dart SDK files, project development dependency plug-in API
Scratches and Consoles	List of temporary files and buffers created

3.4 Chapter Summary

This chapter starts by exploring the requirements of applications based on Flutter's native plug-ins, and introduces the functional requirements and non-functional requirements of the system. Demand is the original intention of every APP. We can better design products that meet the requirements by truly understanding the demand. A robust program requires a good set of architectural patterns. This application is also divided into the business, component, and base system layers based on a general layered architecture pattern, which can be well extended and decoupled. The key to mastering a language is to understand the project directory structure. This chapter starts with the basic structure of the Flutter project and succinctly sorts out the entire directory structure and description

Chapter 4

The key technology application implementation of Flutter-based native plug-ins on the mobile terminal

The application research of Flutter native plugins mainly focuses on the widely used and representative native plugins, involving cameras, photo albums, dialing, SMS, QR codes, NFC, etc. Starting from using Flutter basic components, build the main application framework. According to the basic principles of convenience, efficiency, compatibility, security, extensibility, and maintainability, the plugin is selected and studied, the plugin's reference is determined, and the combination of the plugin components and the framework is realized. Complete the entire process of using and exploring the plugin. This chapter mainly introduces the home page,phone call, camera, QR code, NFC, photo album, SMS, and other modules from multiple dimensions such as design selection, interaction logic, principle exploration, and code implementation.

4.1 Homepage Design and Implementation

As the main framework of the program entry, the home page is an indispensable function carrier for any App. The home page provides an exploration entry for various functional plug-ins in the flutter application. The UI design of the camera function, photo album function, dial function, SMS function, QR code function, and NFC function is displayed on the home page. Figure 4.1 is the functional business architecture diagram of the home page.

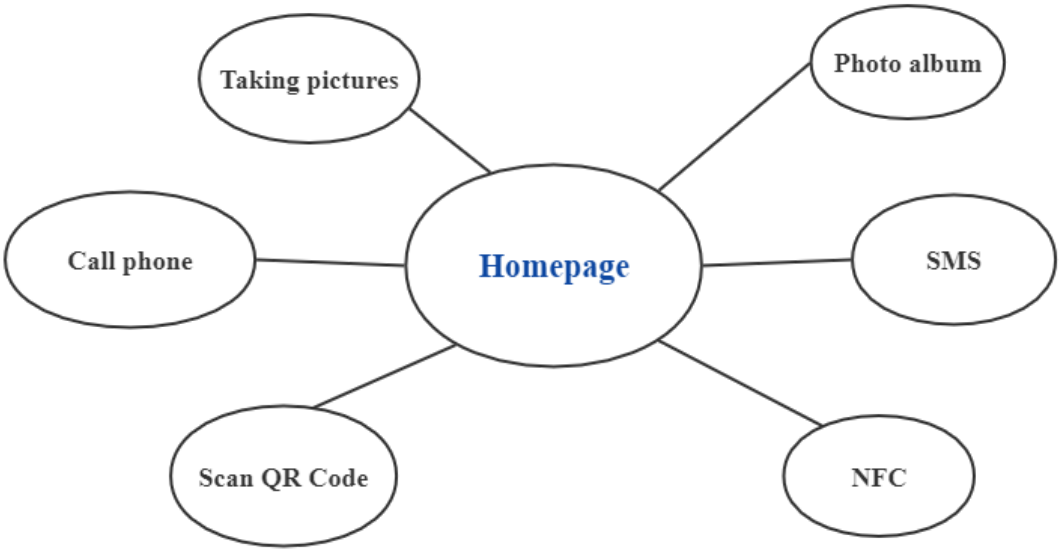

Figure 4.1. Home Function Business Architecture Diagram

After the user clicks the Flutter application icon and starts the APP, he will enter the home page directly. The overall framework of the home page is constructed by combining StatefulWidget and StatelessWidget. Everything in Flutter is a Widget. Widget is the package for building visual effects and the carrier of the UI interface. The build is the implementation method for the Flutter framework to build the UI interface.In the build method, customizing the UI is usually achieved by configuring the corresponding UI for the basic Widget or combining various basic Widgets. StatefulWidget and StatelessWidget are two subclasses of Widget.

StatefulWidget

When creating a new Flutter project, the system will generate a default sample program, which is a StatefulWidget at this time. The system chooses StatefulWidget because in the sample code, we will click the button, and the data displayed

on the interface will change; at this time, we need a variable to record the current state, and then display this variable on a Text Widget; and Every time the variable changes, the content displayed on our corresponding Text also changes. And StarefullWidget can do just that.

The data defined in the Widget is immutable and needs to be defined as final because: at the beginning of the design, Flutter decided that once the data displayed in the Widget changes, the entire Widget needs to be rebuilt; so Flutter needs to pass some mechanisms to qualify member variables defined in Widget as final;

```
@immutable
abstract class Widget extends DiagnosticableTree {
...omitting the code
}
```

Through the above Widget source code, Flutter realizes that the data defined by the Widget during the development process is final.There is a very critical part here, @immutable, which is an annotation involving Dart's metaprogramming.The official description of @immutable: the class or subclass marked by the @immutable annotation must be immutable.[24].

But StatefulWidget needs stateful changes (which can be understood as variables), so to meet this requirement, flutter designs StatefulWidget into two classes, that is to say, two classes must be created when creating StatefulWidget: one class inherits from StatefulWidget, as Part of the Widget tree; a class that inherits from State, is used to record the State that StatefulWidget will change, and build a new Widget according to the change of State, thus realizing StatefulWidget to store variable State.

Mutable state, in turn, introduces a new conceptual lifecycle. The program's life cycle refers to: in iOS development, you need to understand the whole process of UIViewController from creation to destruction, while in Android development,In Android development, you need to know the entire Activity process from creation to destruction. so that different actions are done in different lifecycle methods; Also in front-end development: Vue and React's components also have their life cycles, and developers can do different operations in different life cycles;

Flutter widgets also have a lifecycle. StatelessWidget can directly pass values from the parent widget and call the build method. The whole process is very simple; StatefulWidget needs to manage its data through State and also determines whether to rebuild the entire Widget by monitoring the changes of State; The lifecycle callback of StatefulWidget is shown in Figure 4-3 below. The content in the gray part is operated internally by Flutter. We do not need to set it manually; the white part represents methods that we can manually monitor or call; we know that StatefulWidget itself consists of two classes: so StatefulWidget and State

need to be analyzed separately.

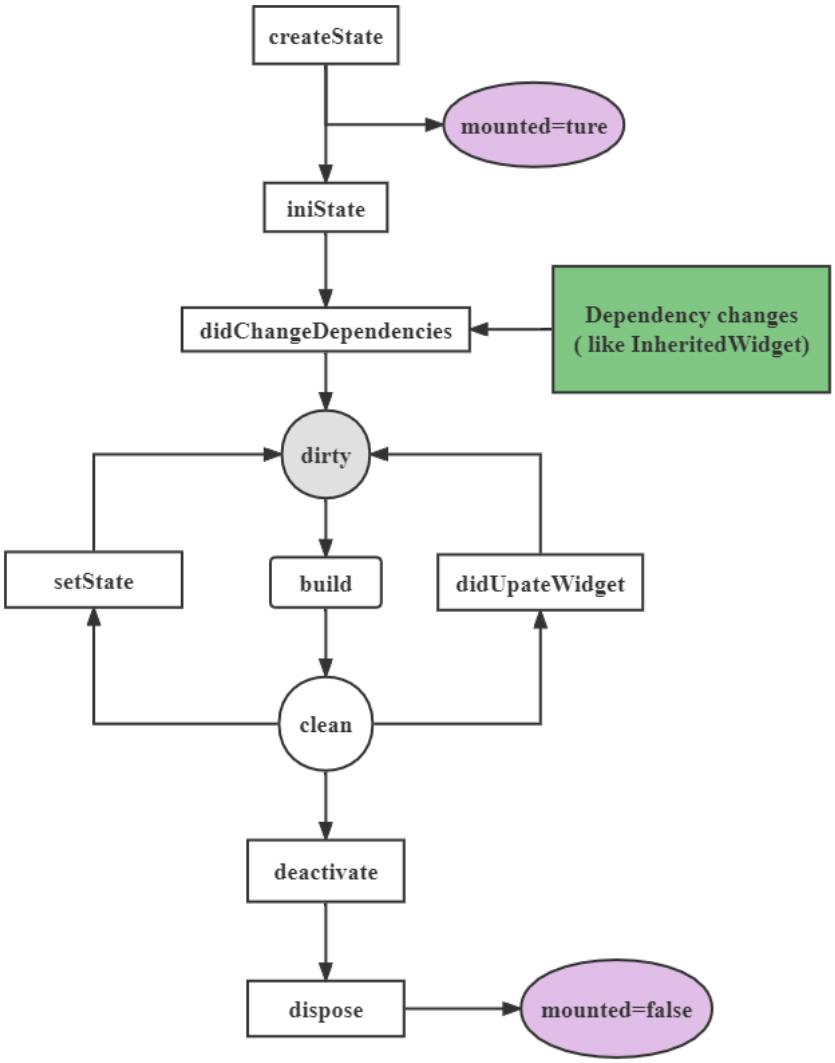

Figure 4.2. Lifecycle Flowchart

First, execute the relevant methods in StatefulWidget:

1. Execute the Constructor of StatefulWidget to create a StatefulWidget.

2. Execute the createState method of StatefulWidget to create a State object that maintains StatefulWidget.

Secondly, Execute the relevant methods of the State class when calling createState:

1. Execute the Constructor of the State class to create a State object.
2. performs initState, usually to perform some data initialization operations, or possibly to send network requests.
3. executes the didChangeDependencies method. This method will be called in the following two situations.

   Case 1: It will be called by calling initState;

   Case 2: When some data in a dependency changes
4. Flutter executes the build method to see which widgets need to be rendered in our current code.
5. When the current Widget is no longer used, it will call dispose to destroy it.
6. Manually calling the setState method will re-call the build method to build the corresponding Widget based on the latest state (data). The
7. didUpdateWidget method is executed when the parent Widget triggers a rebuild, and the system calls the didUpdateWidget method.

StatelessWidget

The StatelessWidget in Flutter is a widget that does not need to change the state; that is, the StatelessWidget has no internal state that needs to be managed and is a stateless widget. The basic components of StatelessWidget are shown in Figure 4.3 below.

Container

Widgets for drawing, positioning, and resizing. We can usually use it as a container view and then carry out a specific layout through the internal subviews We can control the view's width and height, background color, shadow rounded corners, etc., through it. If there are no child widgets, no width, height, and constraints are set, and the parent widget has no unbounded constraints set, it will adjust itself to be small enough. If there is no child widget, alignment, but width, height or constraints are provided, then the Container will adjust itself to be small enough according to its constraints and the parent node. If there are no child widgets, width, height, constraints, and alignment, but the parent widget provides bounded constraints, the Container will adjust itself to be large enough according to the parent widget's constraints. If there is alignment and the parent widget provides unbounded constraints, the Container resizes to wrap the child widget; If there is alignment and the parent Widget provides bounded restrictions, within the

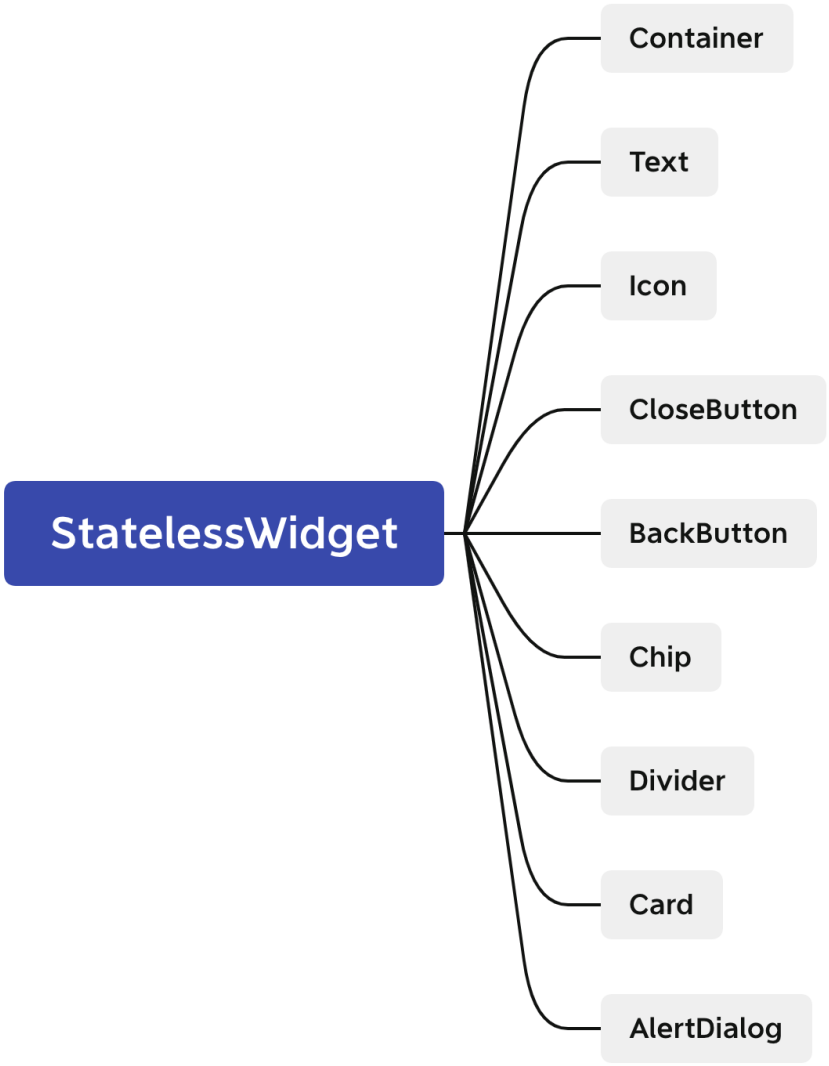

Figure 4.3. StatelessWidget basic component diagram

scope of the parent Widget, the Container will adjust itself to be large enough, and then adjust the position of the child widget according to the alignment; If there is a child widget but no width, height, constraints, and alignment, the Container will pass the constraints of the parent widget to the child widget and adjust itself according to the child widget. The Container has the following properties:

- margin: padding, related to child widgets.

- padding: margins, related to the parent widget.
- child: child widget.
- color: background color, if foregroundDecoration is set, it may cover the color effect.
- constraints: Boundary constraints [25].

Text

Text in a single format. Usually used for plain text display

Icon

Mainly used for some vector icons, it has the Size attribute is primarily used to control the size, and the color attribute controls the displayed theme color [26]. TextDirection is the setting of the text direction.

Divider line

The dividing line is often used in actual projects, such as between list elements, etc. The height parameter in Divider refers to the height of the container, not the height of the line. If you want to change the height of the line, you can only customize the component. thickness, the line width of the dividing line, the dividing line is in the center of the Divider, the left spacing of the indent dividing line, the right spacing of the endIndent dividing line, the color of the dividing line.

Card

Cards with rounded corners, shadows, borders, etc., are often used for functional block layouts that require rounded shadows. You can pass the card background color (color), shadow height (elevation), BorderShape, borderOnForeground, margin, clipBehavior, child controls Attributes such as control the display style.

AlertDialog

The pop-up box is a commonly used component and is often used for reminders. We also use this component in many places in this project, such as displaying picture results, mobile phone number input, QR code scanning results, and other functions. It mainly has attributes such as title, titlePadding, titleTextStyle, content, contentPadding, contentTextStyle, actions, backgroundColor, elevation, and shape..

Based on the above basic principles and basic components, the overall functional design of the home page from top to bottom is the navigation bar area, camera, photo album, dial-up, SMS, QR code, NFC. The overall framework layout code of the program is as follows, The StatelessWidget wraps the overall root view, sets the program's theme Colors.blue with ThemeData, and MyHomePage implements our home page function specifically.

```
void main() {
runApp(MyApp());
}
class MyApp extends StatelessWidget {
@override
Widget build(BuildContext context) {
return MaterialApp(
title: 'Flutter Demo Home Page ',
theme: ThemeData(
primarySwatch: Colors.blue,
),
debugShowCheckedModeBanner: false,
home: MyHomePage(title: 'Flutter Demo Home Page'),
);
}
}
```

The following is the specific implementation of the MyHomePage homepage program framework. This widget is the home page of the application. It is the stateful component page of StatefulWidget, and the main body includes two parts, StatefulWidget and State. StatefulWidget is a UI layout component. The state is part of state management. The title is a final attribute that can configure the corresponding title for our homepage navigation bar. Widget build (BuildContext context) is how the UI layout is rendered. Each call to setState triggers a rerun of the method and refreshes the layout rendering. The advantage of the Flutter framework is that the rerun build method is fast and efficient so that anything that needs to be updated can be rebuilt instead of changing the Widget instance individually.

- title: Widget - the main content in the Toolbar, usually displayed as the title text of the current interface.
- PopupMenuButton to display as three dots, click to pop up a secondary menu.
- brightness : Brightness - the brightness of the Appbar, with white and black themes, the default value is ThemeData.primaryColorBrightness.
- textTheme : TextTheme - Text style on the Appbar.

The navigation bar down is the main body of the home page, including photo albums, dialing text messages, QR codes, and NFC function modules. The main structure is based on the SingleChildScrollView view, which is better compatibility. SingleChildScrollView is a sliding. The component is convenient to be compatible with the area beyond the screen and swipe to browse.

- scrollDirection = Axis.vertical: scroll direction.
- reverse = false: whether to reverse the order.
- primary: Whether to support scrolling when the content is not enough to scroll.
- physics: controls the interaction of the user's scroll view.
- controller: sliding controller.

SingleChildScrollView subview is built with Center as the base component and Column Center displays its child widgets in the Center of itself. Common properties are as follows.

- widthFactor: width factor.
- heightFactor: height factor.
- child: child view.

Column

A container for storing other Widgets, which can arrange its sub-components in the vertical direction. Usually, the arrangement of sub-Widgets on the vertical axis is controlled by mainAxisAlignment. The space occupied by Column is set to be the largest by mainAxisSize, and the alignment of all sub-widgets is realized by crossAxisAlignment. Only when the crossAxisAlignment is start and end, the textDirection setting the left and right display orientation of the child widget will work. Children load a group of child widgets and set the UI layout through these properties.

In summary, the overall program architecture design of the home page is implemented as follows.

MyHomePage is a stateful component page that inherits from StatefulWidget. It provides an external title parameter. You can configure the homepage navigation bar through the title parameter when calling externally. _MyHomePageState part is a Widget that inherits from the State state. By overriding the Widget build(BuildContext context) method, the layout rendering of the home page is realized. The Widget build(BuildContext context) method internally returns a Scaffold component. Scaffold defines a UI framework, including the header navigation bar, body, floating button in the lower right corner, bottom navigation bar, etc. Through the appBar property of the Scaffold component, we define an AppBar component to set the navigation bar by setting Scaffold's body to combine SingleChildScrollView, Center, Column, Row, Icon, Text, GestureDetector, Inwell, and other components to complete the layout settings of the home page.The home page function page design function is finally realized, as shown in Figure 4-4 below.

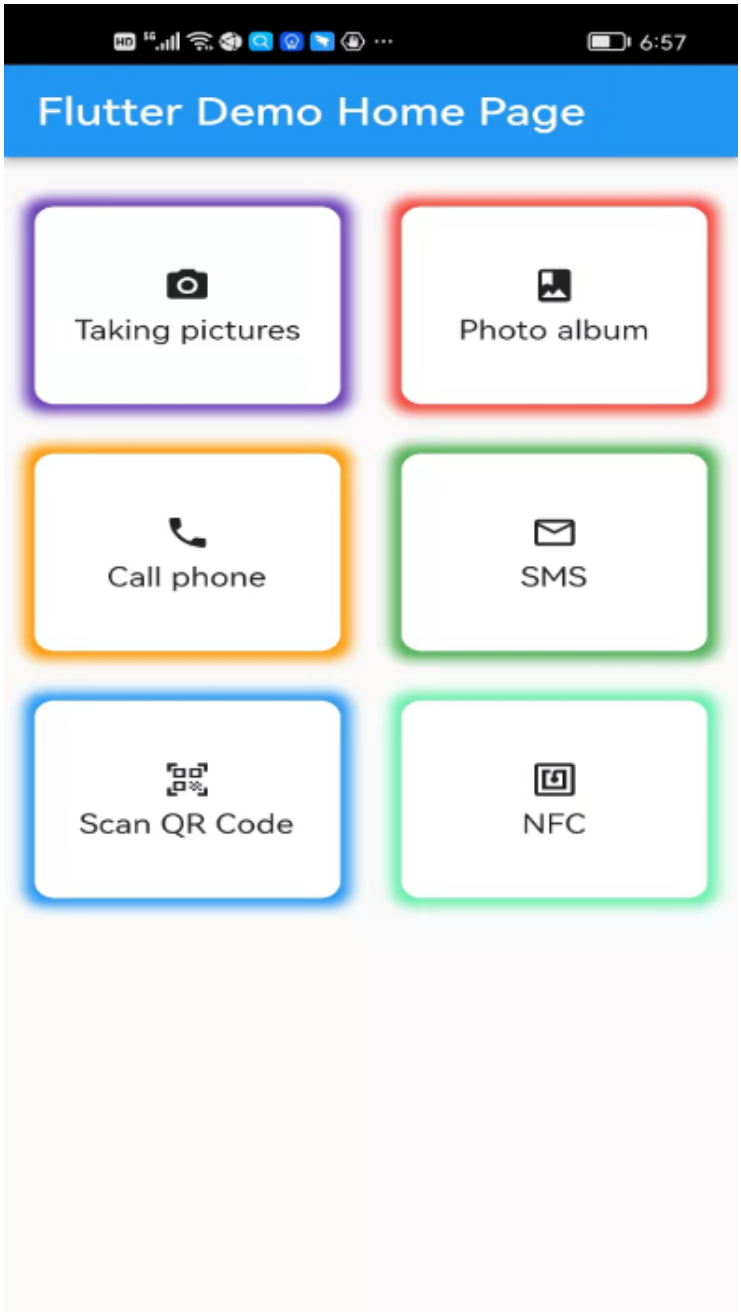

Figure 4.4. Home page

4.2 Design and implementation of taking picture and photo albums

According to the functional layout design of the home page, below the navigation bar is the photo and photo album functions. So let's start by exploring the plugins related to camera and photo album. The UI layout is based on the principle of minimalist design to provide a good visual experience while being convenient to use.

As a whole, the horizontal layout component Row is used as the functional area framework, the camera component layout implementation method setupTakingPicturesView(), and the camera component layout implementation method setupAlbumView() are separately extracted. The camera and photo album function buttons are wrapped by the Row component and displayed in equal divisions in the horizontal direction.

ROW

Row is mainly used for horizontal layout, making a group of widgets arranged horizontally. Generally, children pass in the list of children and then control the alignment of the horizontal and vertical axes through mainAxisAlignment and crossAxisAlignment, respectively.

The overall functional component layout of the camera and album is similar. GestureDetector wraps the overall view to facilitate adding click events. The camera view adds the _getCameraImage() method. When the camera view button is clicked, this function method will be executed, and the corresponding plug-in will be called. Album view Adding the _getAlbumImage() method will call the system album. The functional area is laid out in a Container component with a width of 150 and a height of 120. The whole is a functional block with a white background. The entire area can be clicked, and a rounded border is added to the functional area through the decoration property. The rounded corners are set to 5, the shadow extension degree and the shadow blur degree are set to 5, and the shadow color functional area is set through the blurRadius and spreadRadius of BoxShadow, respectively. The photo is a purple gradient background shadow, and the photo album is a red gradient shadow. The core of the overall function block is composed of Column The component is wrapped, with an Icon and a title Text component up and down, respectively. The whole is relatively beautiful and generous, The effect is shown in Figure 4.5 below.

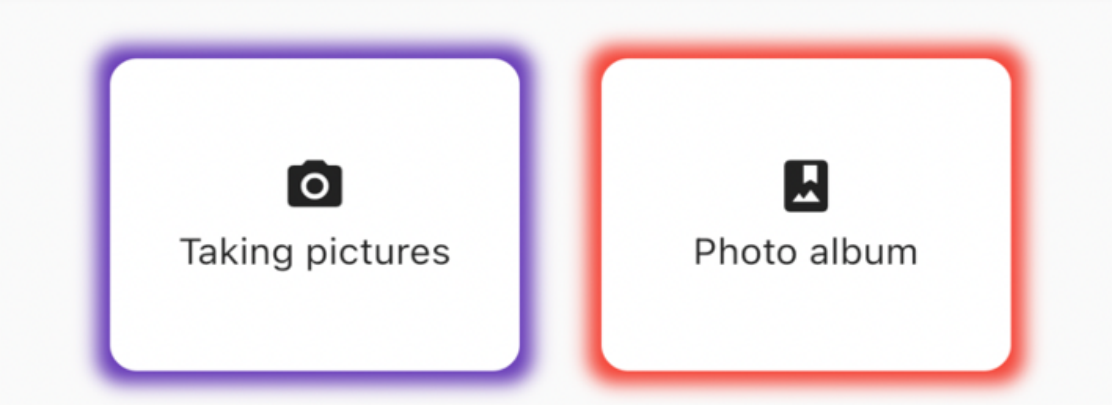

Figure 4.5. Camera and album Ui design figure

When clicking the camera function area or the album function area, the _getCameraImage() and _getAlbumImage()methods will be called respectively, and the mobile phone camera will be called to take pictures or access the mobile phone album. These two functions mainly depend on the basic hardware of the mobile phone, which is the interactive process of entering the native plugin. Based on the previous requirements and basic principles, the image_picker plugin is finally determined as the object of exploration after the layer-by-layer screening.

First, need to add the image_picker: ^0.6.7+22 plugin to the pubspec.yaml plugin management file. After adding the plugin, you need to execute the flutter pub get command to update the plugin dependencies to the project. After adding the plugin, you need to import the plugin before using the plugin

```
import 'package:image_picker/image_picker.dart';
```

When performing a photo or photo album task, call the await picker.getImage(source: ImageSource.gallery) method of the image_picker plugin to view the album; and the await picker.getImage(source: ImageSource.camera) method calls the camera to take a photo, and when the plugin executes the picker.getImage(source: ImageSource.camera), the plug-in will encapsulate and call the native method of calling the camera to take pictures according to the different native platforms of android and iOS.After the picture is taken, the photo results will be returned, and the results will be displayed through the corresponding AlertDialog component.

async/await

They are keywords of the Dart language; they are the paradigm of asynchronous programming in Flutter, which allows you to write asynchronous code in the form of synchronous code. In daily usage scenarios, we usually use async and await to asynchronously process time-consuming operations such as IO, network requests,

and Platform channels communication in Flutter.

The basic process of the camera and album plug-in function is shown in Figure 4.6 below.

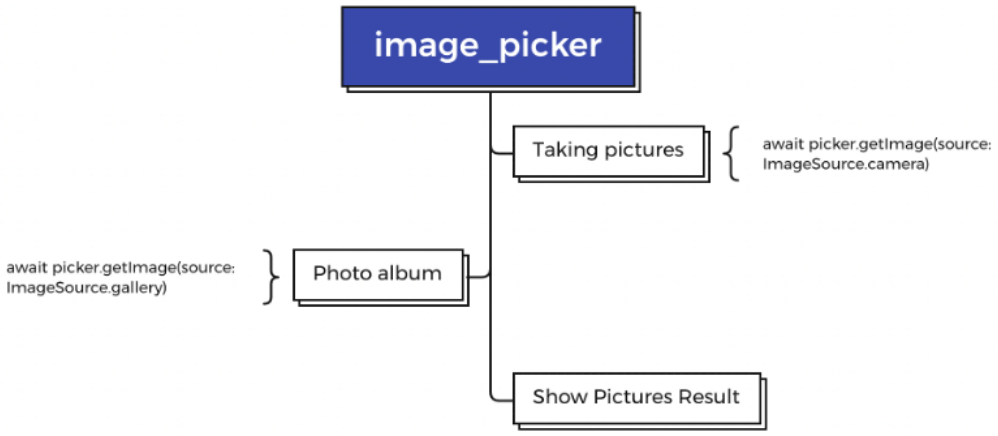

Figure 4.6. Photo album and camera plug-in function flow chart

4.3 Design and Implementation of Call and SMS

Under the camera and photo album, the function module is the call and text function area. The overall functional component layout is also based on minimalism and consistency. Wrap the overall view through InkWell to add click time, execute the showDialog() method to pop up the phone number input dialog box, and the dialog box completes the layout of the function through the custom component CallPhoneDialogContent, which provides a title parameter to configure the title of the pop-up window, and a TextEditingController text input controller, and provides okBtnTap confirmation button click event callback, cancelBtnTap cancel button click event callback, when the OK button is clicked, the callback method of the OK button will obtain the corresponding input text through TextEditing-Controller, and then call the dial-in plug-in to complete the function call . The functional area is laid out in a Container component with a width of 150 and a height of 120. The entire area is a functional block with a white background. The entire area is clickable. Add a rounded border to the functional area through the decoration attribute. The overall rounded corner is 5, the degree of shadow

extension and shadow blur is set to 5, and the shadow color ribbon is set by the blurRadius and spreadRadius of BoxShadow respectively. The call phone is an orange-yellow gradient background shadow, and the SMS is a green gradient background shadow .ow. The core of the overall function block is composed of Column The component is wrapped, with an Icon and a title Text component up and down, respectively. The whole is relatively beautiful. The display result is shown in the following figure 4.7.

Figure 4.7. Call phone and SMS Ui design figure

When the dial function area or SMS function area is clicked, the showDialog component will be called. The pop-up window component will pop up the input page component CallPhoneDialogContent. When the number input is completed and click the dial or send SMS button, the entered mobile phone number will be called back,At the same time, it will call the system's mobile phone dialing and sending SMS functions. Based on this, we chose the url_launcher plugin as the object of exploration.

First, need to add the url_launcher: ^5.7.10 plugin to the pubspec.yaml plugin management file. After adding the plugin, you need to execute the flutter pub get command to update the plugin dependencies in the project. After adding the plugin, need to import the plugin before using the plugin:

```
import 'package:url_launcher/url_launcher.dart';
```

When the dialing task is executed, the following code will be called to dial. The basic grammar rule of dialing is tel: + mobile phone number. First, it will judge whether the current environment can dial, and if so, call the plug-in to dial.

```
var url = 'tel:${_vc.text}';
if (await canLaunch(url)) {
```

```
await launch(url);
} else {
throw 'Could not launch $url';
}
```

When the SMS task is executed, the following code will be called to dial. The basic grammar rule of dialing is sms: + mobile phone number. First, it will judge whether the current environment can send SMS, and if so, call the plug-in to execute SMS.

```
var url = 'sms:${_vc.text}';
if (await canLaunch(url)) {
await launch(url);
} else {
throw 'Could not launch $url';
}
```

The basic flow of the Call phone SMS plug-in function is shown in Figure 4.8 below.

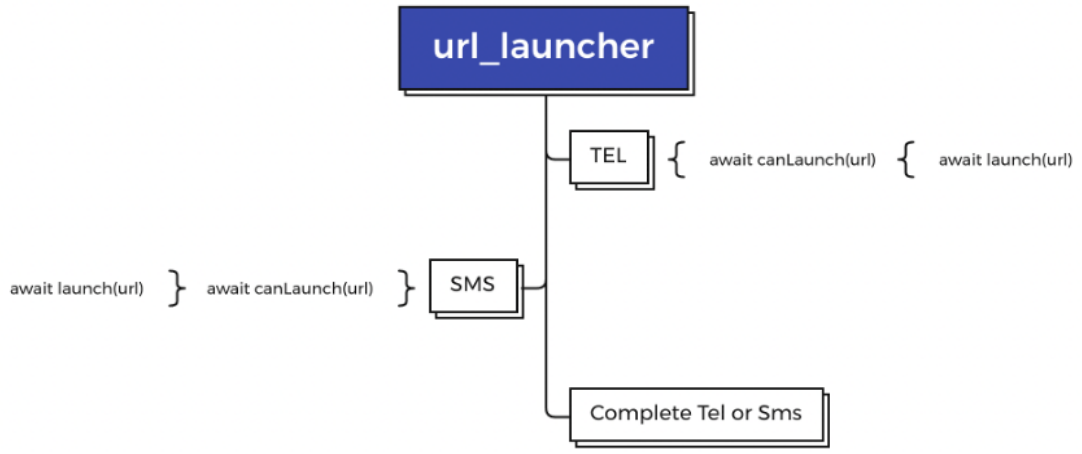

Figure 4.8. Call and SMS plug-in function flow chart

4.4 QR code design and implementation

The QR code functional area wraps the overall view with gesture GestureDetector to facilitate adding click events.The layout of the functional area is a Container component with a width of 150 and a height of 120. The entire area is a functional

block with a white background. The entire area can be clicked. The decoration attribute adds a rounded border to the functional area. The overall rounded corner is set to 5. The shadow extension and shadow blur are set to 5. The blurRadius and spreadRadius of BoxShadow set the shadow color function area. The whole is a blue gradient background shadow . The core of the entire function block is wrapped by the Column component, with an Icon and a title Text component. The displayed result is shown in the Figure 4.9 below.

Figure 4.9. Scan QR Code Ui design figure

For the exploration of the QR code plugin, we chose the scan plugin as the object of exploration. At the same time, it needs to be supplemented by the permission_handler permission to obtain the plug-in:

```
pub.dev/packages/permission_handler
```

Add the scan: $\hat{1}$.5.0 plugin and the permission_handler: $\hat{5}$.1.0+2 plugin to the pubspec.yaml plugin management file according to the process. After adding the plugin, you need to execute the flutter pub get command to update the plugin dependencies to the project. After adding the plugin, you need to import the plugin before using the plugin:

```
import 'package:permission_handler/permission_handler.dart';
import 'package:scan/scan.dart';
```

When you click on the QR code function area, the following code will be executed, and the camera permission will be obtained through Permission.camera.request.

If you have permission, it will jump to the custom packaged component scanning page ScanPage.

```
var status = await Permission.camera.request();
if(status.isGranted){
Navigator.push(context, new MaterialPageRoute
(builder: (context) => ScanPage()));
}
```

ScanPage inherits from StatelessWidget and is a stateless component page. The page defines an IconData type attribute lightIcon to control the flash. ScanController type attribute _controller controls the scan animation, wraps a MaterialButton button through StatefulBuilder, and add a The onPressed button updates the flash status. The current page will turn on the camera. A top-down scanning animation will appear directly. When the relevant QR code is recognized, the result of the callback recognition will be parsed, and the recognition result will be displayed through the AlertDialog.

There are two ways to realize QR code recognition: one is to use the scan view that comes with the ScanView plugin to scan through the camera, and the scan result will be returned in the onCapture method; the other is to execute await ImagePicker().getImage(in combination with the album plugin. source: ImageSource.gallery) method to obtain the QR code image, and then execute the Scan. Parse (pickerImages.path) method through the QR code recognition plugin Scan to parse the QR code image data to obtain the result and then display the result in a pop-up window. The basic flow of the QR code function is shown in Figure 4.10 below.

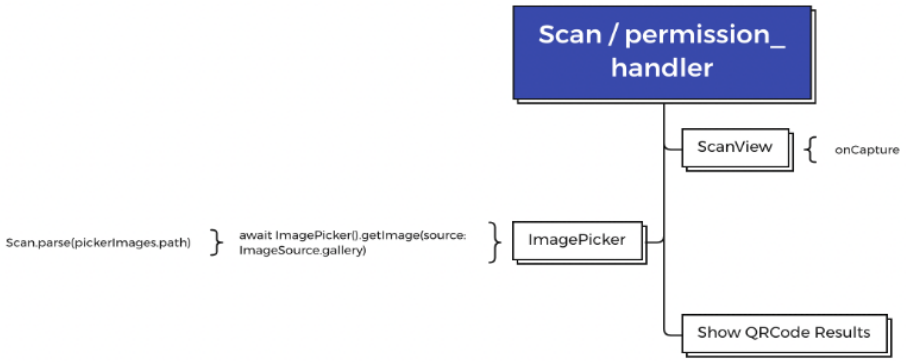

Figure 4.10. QR code function basic flow chart

4.5 NFC design and implementation

To facilitate adding click events, the NFC ribbon wraps the overall view with a gesture GestureDetector. The layout of the functional area is a Container component with a width of 150 and a height of 120. The entire area is a functional block with a white background. The entire area can be clicked, and a rounded border is added to the functional area through the decoration property. The overall rounded border is 5, the shadow expansion and shadow blur are both set to 5, and the shadow color functional area is set by the blurRadius and spreadRadius of BoxShadow, respectively. The whole is a green gradient shaded functional area, and the Column component wraps the core of the entire functional block, which is an Icon and a title Text component. The display results are as follows Figure 4.11.

Figure 4.11. NFC Ui design figure

For the exploration of NFC function plug-ins, we chose the NFC_manager plug-in as the research object. Add the NFC_manager: ^3.1.0 plugin to the pubspec.yaml plugin management file according to the process. After adding the plugin, you need to execute the flutter pub get command to update the plugin dependencies in the project. After adding the plug-in, you need to import the plug-in before using the plug-in.

pub.dev/packages/nfc_manager

When you click on the QR code function area, the following code will be executed, jumping to the NFC page of the custom packaged component.

```
Navigator.push(context, new MaterialPageRoute
(builder: (context) => NFCPage()));
```

The NFC page mainly includes a result display area and three function buttons: Tag Read, Ndef Write, and Ndef Write Lock. This module builds a Widget through FutureBuilder. The asynchronous model in Flutter builds its widgets based on the latest snapshot of interactions with Futures. FutureBuilder contains Future and builder. Future acquires data through asynchronous operations, and the builder passes in the context and asynchronous snapshot AsyncSnapshot. AsyncSnapshot contains ConnectionState, data, etc. The builder returns different widgets depending on the state of the snapshot. At the same time, the buttons are laid out through the Jiugongge component GridView. GridView is encapsulated based on ScrollView, which is a scrolling multi-column list. Its common properties are

- scrollDirection: scrolling direction, there are vertical and horizontal, the default is vertical direction (Axis.vertical).
- reverse: The default is to scroll from top or left to bottom or right, this property controls whether to reverse, the default value is false, no reverse scrolling.
- controller: controls the position of the child when scrolling.
- primary: Whether it is the primary scroll view associated with the parent's PrimaryScrollController.

When jumping to the current function page, the program will judge whether the current device can meet the NFC execution environment through NfcManager.instance.isAvailable(). When the_tagRead button is clicked, the NfcManager.instance.startSession(onDiscovered: (NfcTag tag) method will be executed to return the result and stop data scanning. Ndef is determined when the_ndefWrite method is executed, and the NFC write will be locked when the_ndefWriteLock method is executed. The basic process of NFC function is shown in Figure 4.12.

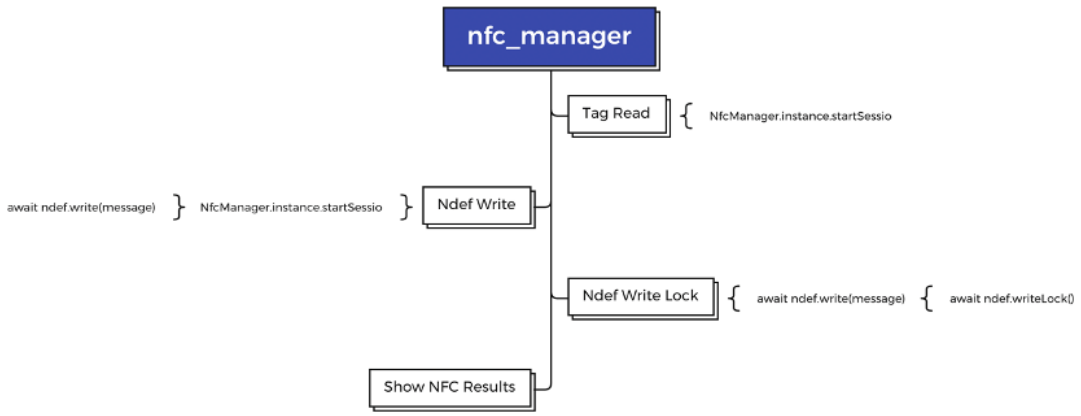

Figure 4.12. NFC function basic flow chart

4.6 Chapter Summary

This chapter mainly introduces the design and implementation of each functional module of the Flutter-based native plug-in application, including the overall program structure, theme color, navigation bar, photo album, call, SMS, QR code, NFC specific design and implementation, and business call logic And the usage process and principle of the corresponding plug-in. It focuses on the minimalist design concept of functional modules and the detailed explanation of the components used in the implementation process to better understand the actual application of Flutter and the essence of plug-in interaction.In addition, in realizing project functions, factors such as compatibility, stability, and scalability also need to be considered.

Chapter 5

Plug-in-based research effect review and testing

Testing after application development is an essential step and an intuitive process for checking the results of our research. Only rigorous testing can make our development research more meaningful and reliable. Exploring applications based on Flutter's native plug-ins mainly explores representative plug-ins that are strongly dependent on the native hardware system of mobile phones. Therefore, during the test process, we need to coordinate the use of the real mobile phone environment, run the program on the mobile phone, and conduct actual debugging tests.

5.1 Photographic test

The camera function is implemented according to the requirements and design mentioned above, and the test process is as follows. First, click the camera button on the homepage of the program, which will trigger await picker.getImage(source: ImageSource.camera) method of the ImagePicker plugin, which will call up the camera interface of the phone. In the actual test process, we followed these steps and successfully called the camera function of the mobile phone, as shown in the following figure 5.1.

After clicking the camera button, you can obtain the specific image file through the imagePath = File(cameraImages.path);method, and then display the image results we intercepted in the form of a showAlertDialog pop-up window, as shown in the following figure 5.2.

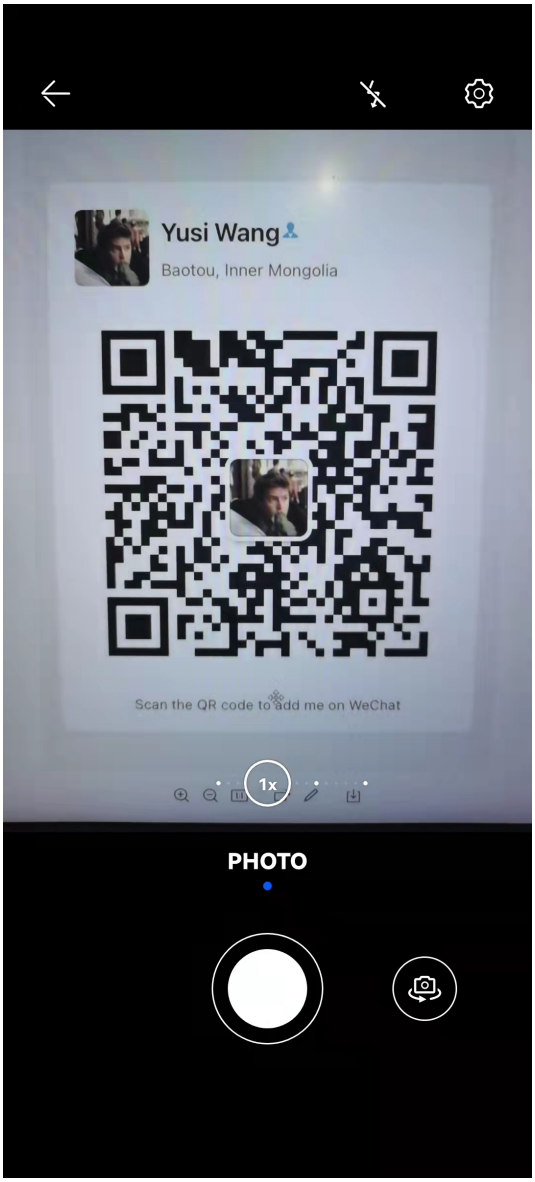

Figure 5.1. Mobile phone camera interface

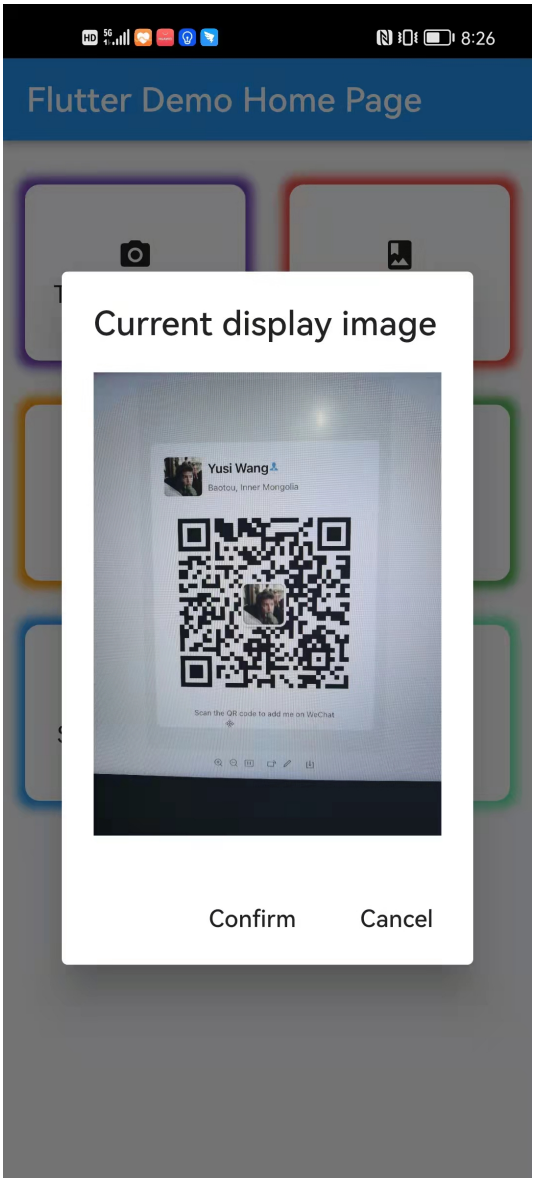

Figure 5.2. Camera function Click the result picture

So far, the debugging process of our camera plug-in has been completed, and it has indeed met our initial design requirements as expected. The functions of taking pictures and obtaining pictures have been realized conveniently and efficiently.

5.2 Album test

The photo album function is as follows according to the debugging test process. When the album button is clicked on the homepage of the program, await picker.getImage(source: ImageSource.gallery) of the ImagePicker plugin will be triggered; At this time, the photo album system interface of the mobile phone will be called . In the actual test process, we successfully called the mobile phone photo album function according to this step, as shown in Figure 5.3.

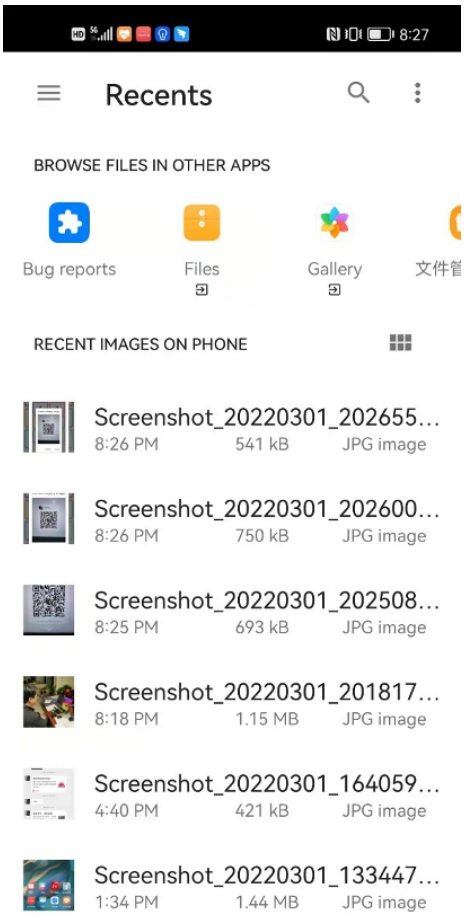

Figure 5.3. Mobile photo album system page

After clicking and selecting any picture in the album, you can obtain the specific picture file through imagePath = File(cameraImages.path);. Then display the result of the image we took in the form of a showAlertDialog pop-up window as shown in the following Figure 5.4.

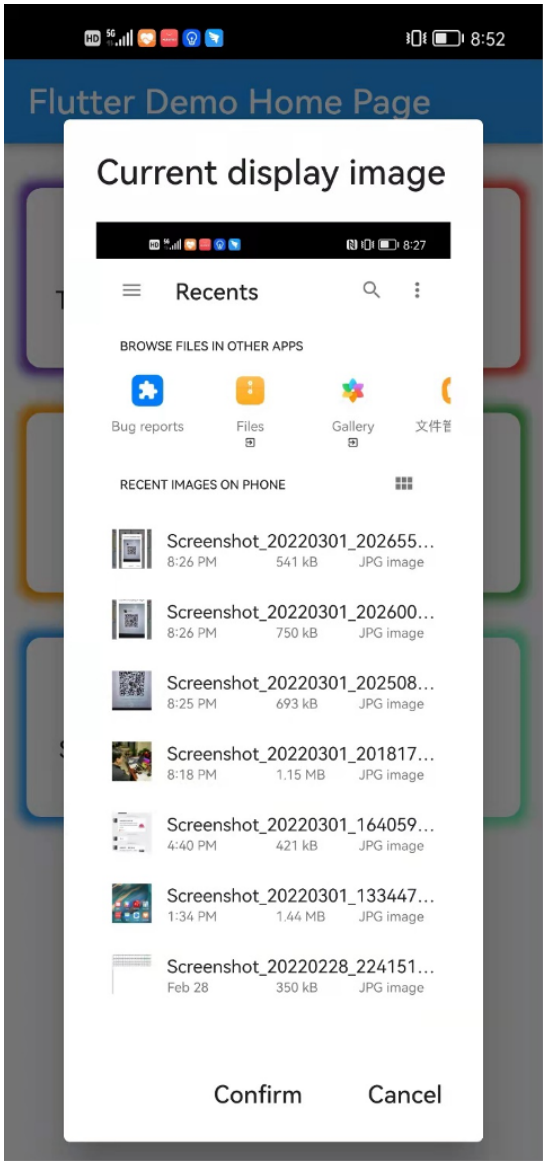

Figure 5.4. Album picture click result picture

In the actual process, according to the plug-in calling process, it has indeed reached our expected design requirements, and the acquisition and successful display of album pictures can be conveniently and efficiently realized.

5.3 Call and SMS test

The debugging and testing process of the dialing function is as follows. We can click the dial button on the home page of the program, and showDialog will be called at this time, and the mobile phone number input box will pop up, as shown in the figure 5.5.

After entering the mobile phone number, click dial by tel: + mobile phone number to trigger the launch method of the url_launcher plugin, and the mobile phone dialing system interface will be called up. In the actual test process, we follow this step to successfully call up the phone dialing system page, as shown in the following figure 5.6.

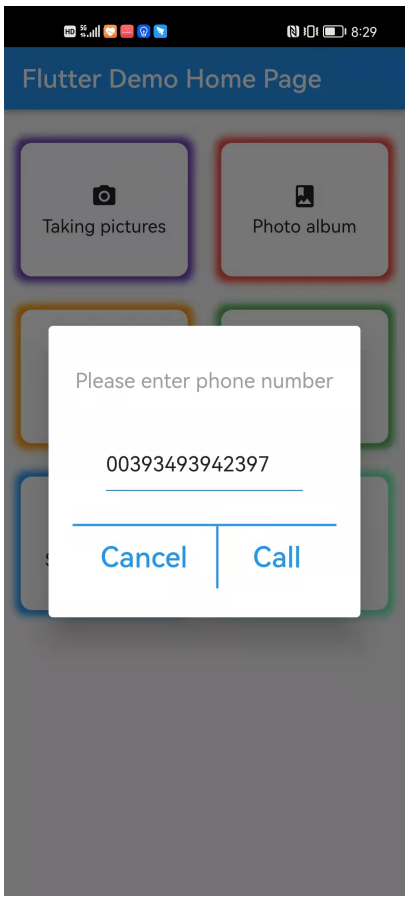

Figure 5.5. Call function interface

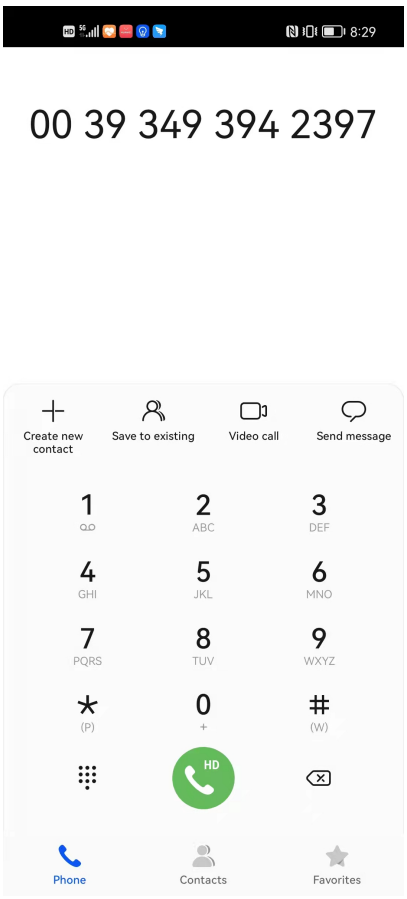

Figure 5.6. Phone's dial system page

The short message function debugging test process is as follows. We can click on the SMS button on the homepage of the program. At this time, showDialog will be called, and the mobile phone number input box will pop up, as shown in

the following figure5.7.

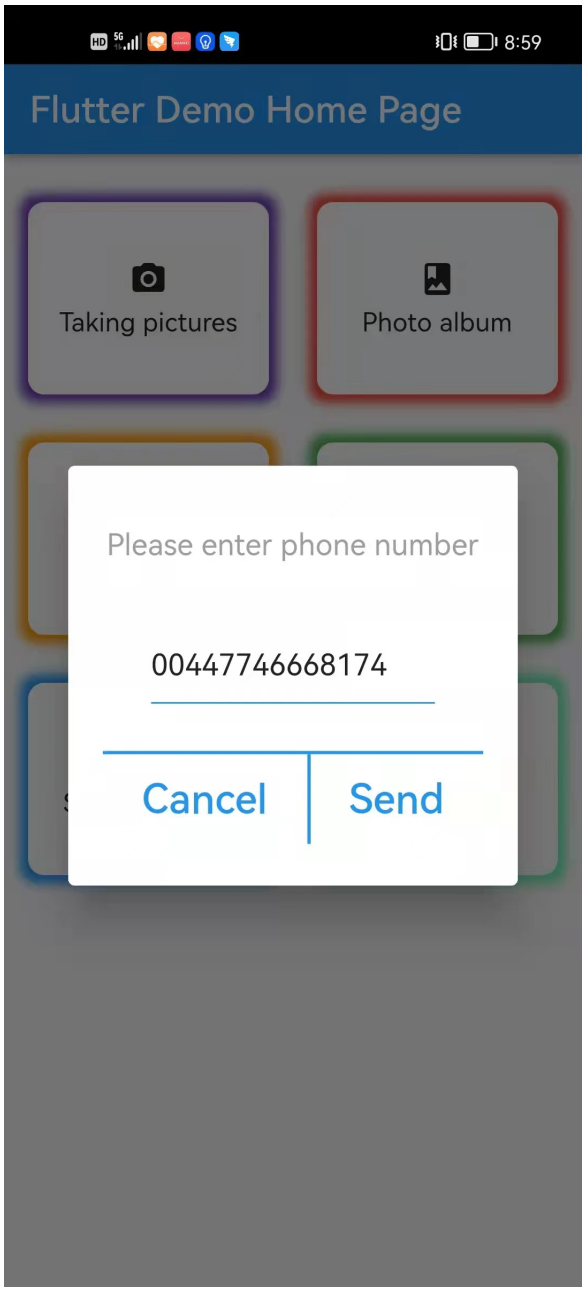

Figure 5.7. SMS function interface

After entering the mobile phone number, click to dial through sms: + mobile phone number will trigger the launch method of the url_launcher plug-in, which

will call up the SMS system interface of the mobile phone. During the actual test, we followed this step and successfully called up the SMS system page of the mobile phone, as shown in Figure 5.8.

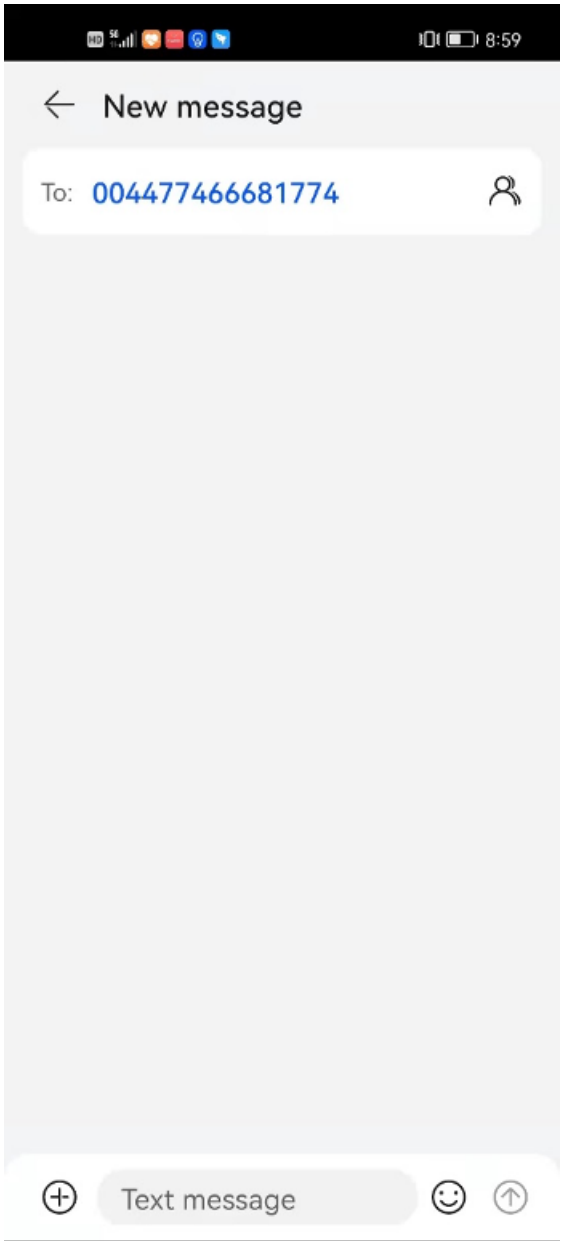

Figure 5.8. The mobile phone's SMS system page

In summary, we rely on the plugin url_launcher and reference to implement the requirements debugging of dial-up and SMS-related functions successfully.

5.4 QR code test

To debug the QR code function, we need to click the QR code function button on the home page of the program, and at the same time, the Navigator.push(context, new MaterialPageRoute(builder: (context) ⇒ ScanPage())); method will be triggered to navigate the page to the Define Scans page. The page behavior is shown in Figure 5.9 below.

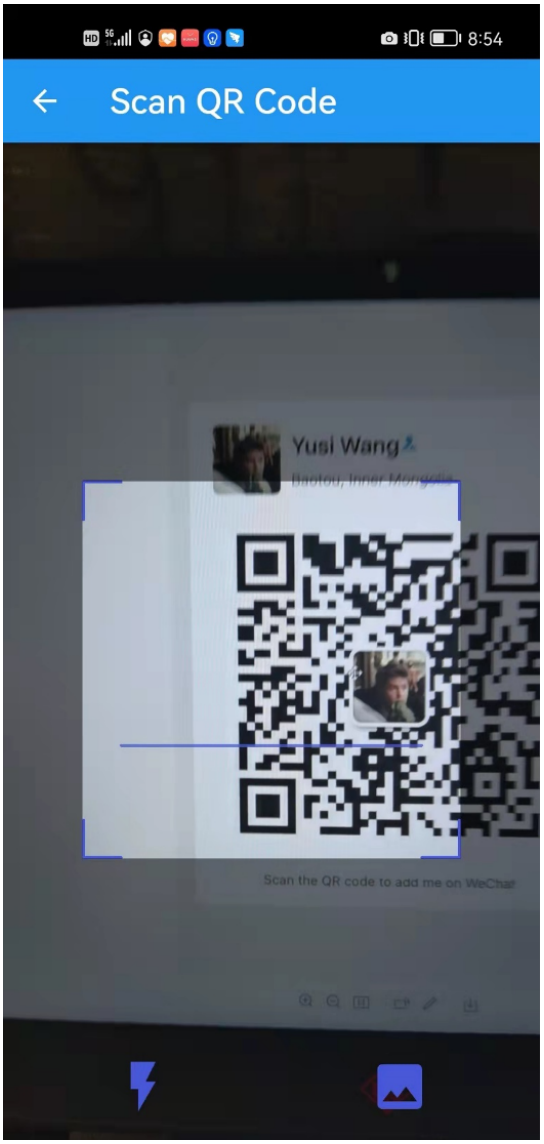

Figure 5.9. Scan QR code fuction interface

The entire page consists of an automatic scanning area, a flash button, and a photo album picture reading button. When the QR code scan recognizes a corresponding QR code image in the area, the onCapture:(data) callback of ScanView will be triggered, and the recognition result of the QR code image will be passed over. At this time, automatic scanning will be suspended, and the recognition result will be passed. The showAlertDialog(data, context) method is displayed in the form of a pop-up window. The display result is shown in the following figure 5.10.

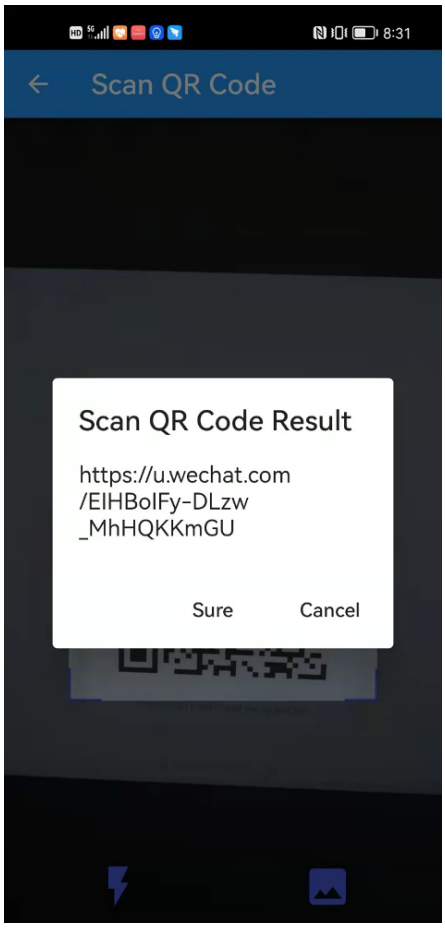

Figure 5.10. QR code scan result picture

We have successfully identified and processed personal WeChat QR code information and displayed it. Through this plug-in, we can easily use two methods: automatically scan the QR code picture or click the album button to read the information from the QR code picture saved in the album. Through testing, we found that the application is indeed as expected, realizing the reading of QR code

information and meeting the expected requirements.

5.5 NFC test

Finally, the debugging of the NFC function. When we click the NFC function button on the homepage of the program, the Navigator.push(context, new MaterialPageRoute(builder: (context) ⇒ NFCPage())); method will be triggered to navigate the page to the custom NFC action page. The NFC page will judge whether the current device environment is suitable for NFC-related operations according to NFCManager.instance.isAvailable(), and display different pages accordingly. When it does not conform to the NFC operating environment, the page is shown in Figure 5.11, and we will prompt that the current environment is invalid, which is convenient for users to enable relevant function permissions.

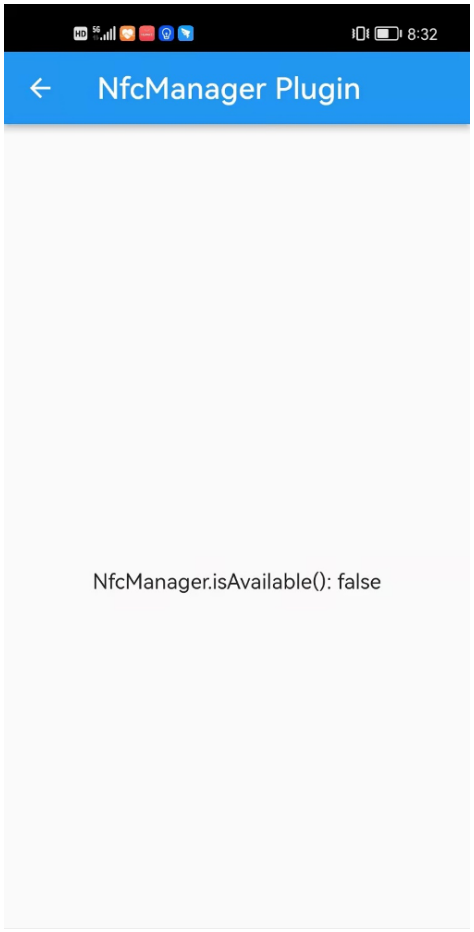

Figure 5.11. The NFC function interface when the mobile phone NFC function is turned off

When the conditions meet the NFC-related operating environment, the NFC will display the following page, mainly the result display area and three common function operation buttons: Tag Read, Ndef Write, and Ndef Write Lock. The page is shown in Figure 5.12.

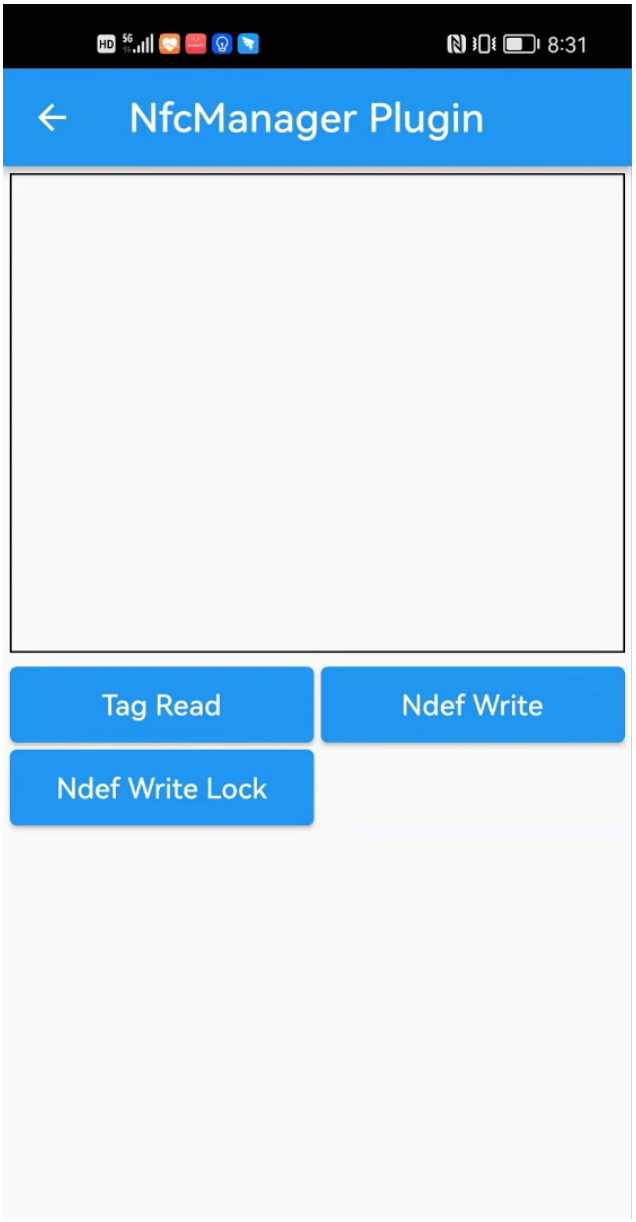

Figure 5.12. The NFC function interface when the mobile phone NFC function is turned on

Move the mobile phone close to the NFC target device (such as a cell access control card with an RFID tag) and click the Tag Read button to start NfcManager.instance.startSession(onDiscovered: (NfcTag tag) to read the relevant data. In the actual formal test process, We read the community's access control information through the mobile phone's NFC function and successfully obtained the

relevant information. The result is shown in the following Figure 5.13.

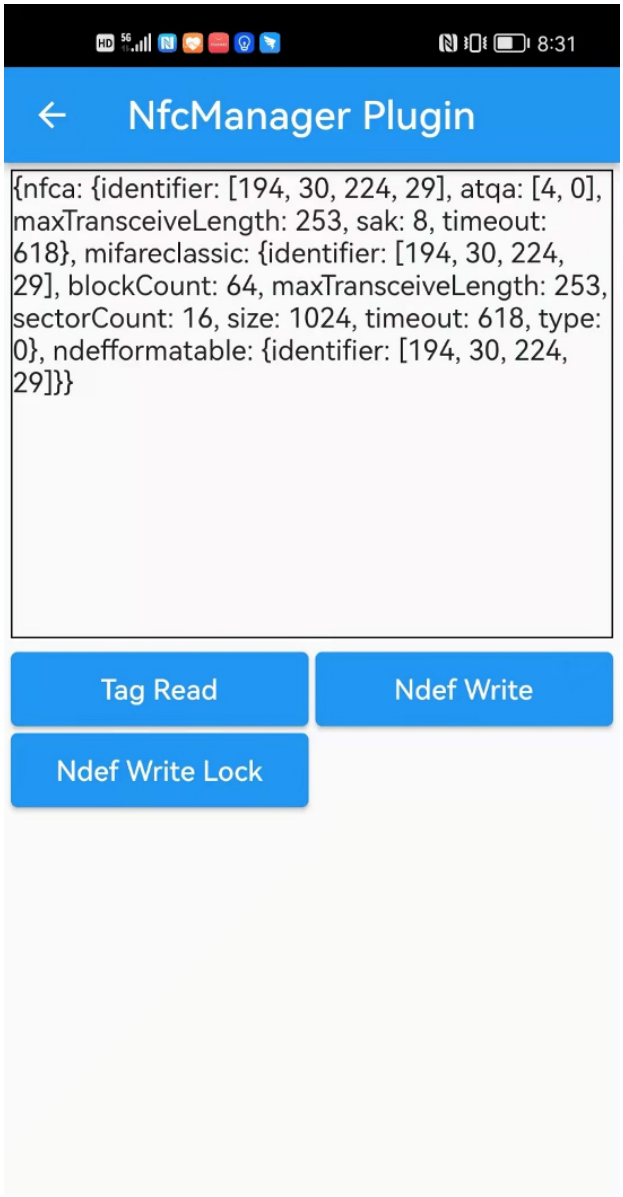

Figure 5.13. NFC information reading page

In summary, we successfully read the RFID tag information through the NFC-related plug-in. The access control information of the daily community was obtained, which met our initial design needs.

5.6 Chapter Summary

This chapter mainly explores and debugs the functional requirements of the original design by running the Flutter native plug-in in the real environment of the mobile phone and successfully realizes the functions of mobile phone photography and mobile photo albums by calling the image_picker plug-in. By calling the url_launcher native plug-in, the mobile phone dialing and SMS sending functions are successfully implemented. By scanning the native QR code plug-in, the two functions of custom scanning of QR code and recognition and reading of QR code pictures in the album are realized. The access control information with RFID tags is successfully read through the NFC plug-in NFC_manager. completing the NFC function test, with the completion of the test process, we also successfully completed the whole process of Flutter native plug-in exploration application from design to implementation to completion of the test.

Chapter 6

Summary and Outlook

6.1 Summary of the paper

Based on the rapid development of mobile Internet, the growing demand for mobile Internet in today's society, and the desire for high performance, high user experience, and efficient cross-platform development language, this paper discusses the basic principles of the most popular cross-platform development technology Flutter. And the corresponding native plug-in application process. Combined with the latest research and the most common and representative functional requirements of current mobile phones, designed and implemented: camera function, photo album function, mobile phone calls, SMS sending, QR code scanning recognition, NFC identification function. The development of the entire project starts with a brief understanding of the background and principles of Flutter and then an in-depth study of various technical points of Flutter. After getting familiar with Flutter's overall framework principles and programming foundation, follow business and non-business requirements. The main direction and main frame of the application of the whole program are determined. The development process realizes various business functions according to the requirements, reasonably grasps the performance of multiple aspects, and finally realizes the intelligent mobile application with good maintainability, high stability, and robust scalability.

The application interface is simple, clear, and beautiful, and the operation is simple and convenient. The overall UI framework adopts the Dart UI layout. Completed the exploration of various native function application plug-ins. Flutter is currently the hottest and most promising cross-platform development framework. Its ecology is also in the process of rapid development and improvement. Good technology also needs more practice accumulation and promotion. Exploring the application implementation based on Flutter's native plugin is a perfect practice of Flutter technology. While familiar with the basic technical framework of Flutter,

it has also explored the entire principle process of Flutter's native plug-ins.

6.2 Future Outlook

With the development of the mobile Internet and the exponential value-added of mobile terminal users, two problems of dynamism and growth cost are mainly faced in daily pure native development. Some cross-platform dynamic frameworks have been born in response to these two problems. At the same time, the industry has been working hard to find a good solution. Today, there are many cross-platform frameworks, which are mainly divided into three categories:

- H5 + native (Cordova, Ionic, WeChat applet).
- JavaScript development + native rendering (React Native, Weex).
- Self-drawn UI + native (Qt for mobile, Flutter).

It is precise because cross-platform technology does not fundamentally solve application performance problems such as UI consistency; the rise and fall of cross-platform technology continue. However, the emergence of Flutter technology has allowed us to see the light of day. Flutter is a framework released by Google for creating cross-platform, high-performance mobile applications. Unlike other cross-platform technologies, Flutter does not use native controls. Instead, it implements a self-drawing engine and uses its layout and drawing system. From an ecological point of view: the Flutter ecosystem is developing rapidly, and the community is very active. Both the number of developers and third-party components are already very impressive. From the technical support point of view: Google is now vigorously promoting Flutter. Many of Flutter's authors are from the Chromium team, and they are very active on Github. From another perspective, from the birth of Flutter to the present, frequent version releases can also indicate that Google has invested a lot of resources in Flutter. From development efficiency: a set of code running on multiple terminals; during the development process, Flutter's hot reload can help developers test, build UI, add features, and fix bugs faster. Hot reload in milliseconds on iOS and Android emulators or real devices without losing state.From the Google I/O conference in 2017, from Google's first release of Flutter to late February 2022, the number of stars on Github is very high, exceeding 127K stars. After more than four years, the Flutter ecosystem has grown rapidly. Many successful cases are based on Flutter at home and abroad, and Internet companies have dedicated Flutter teams. Flutter has received extensive attention and recognition in the industry and warmly welcomed developers. It has become one of the most popular frameworks in mobile cross-end development.

The Flutter-based native plug-in exploration application is also an active development attempt of Flutter technology, which is quite good overall development experience and efficiency. When choosing future development technology solutions, I believe that Flutter will also become one of the indispensable solutions. Developers will happily choose the Flutter framework to try out their newly developed applications to meet their needs. For existing Android applications and iOS natively developed applications, you can also consider importing Flutter into the project in the form of modules in a mixed form to improve development efficiency. The development process can also be combined with a rich plug-in market to quickly and efficiently implement our functions. No matter what technology is used, it needs to serve the product. Choosing the most cost-effective and most suitable solution is necessary according to the project's actual situation. No matter which generation of development technology it is, only a few issues need to be solved: performance, development efficiency, and hot updates. The first two Flutter solved it almost perfectly. As for hot updates, I believe that there will be more and more excellent solutions with the continuous update and iteration. Believe that Flutter's cross-platform development technology should be the future. As for whether Flutter will be the ultimate winner, no one is sure there may be better development frameworks in the future, but this kind of self-painting should be the trend. After this review, design, implementation, and final testing based on Flutter's native plug-in application, I also have a new and comprehensive understanding of Flutter's technology. I have added more interest and expectations to Flutter's technology. In the future, I will continue to explore and practice in-depth, pay attention to the latest developments in Flutter and cross-platform technologies, and strive to contribute to the technical community.

Bibliography

[1] Kinza Shafique, Bilal A Khawaja, Farah Sabir, Sameer Qazi, and Muhammad Mustaqim. Internet of things (iot) for next-generation smart systems: A review of current challenges, future trends and prospects for emerging 5g-iot scenarios. *Ieee Access*, 8:23022–23040, 2020.

[2] Nikita Kuzmin, Konstantin Ignatiev, and Denis Grafov. Experience of developing a mobile application using flutter. In *Information Science and Applications*, pages 571–575. Springer, 2020.

[3] URL: https://docs.flutter.dev/resources/architectural-overview.

[4] Eric Windmill. *Flutter in action*. Simon and Schuster, 2020.

[5] Sebastian Faust. Using google s flutter framework for the development of a large-scale reference application. 2020.

[6] URL: https://docs.flutter.dev/development/tools/sdk/releases?tab=windows.

[7] URL: https://flutter.dev/community.

[8] URL: https://flutter.dev/showcase.

[9] Lars Rodseth. From bachelor threat to fraternal security: Male associations and modular organization in human societies. *International Journal of Primatology*, 33(5):1194–1214, 2012.

[10] Carliss Young Baldwin, Kim B Clark, Kim B Clark, et al. *Design rules: The power of modularity*, volume 1. MIT press, 2000.

[11] Priyanka Tyagi. *Pragmatic Flutter: Building Cross-Platform Mobile Apps for Android, iOS, Web, & Desktop*. CRC Press, 2021.

[12] Bruno Guigas. Specpad: device-independent nmr data visualization and processing based on the novel dart programming language and html5 web technology. *Magnetic Resonance in Chemistry*, 55(9):821–827, 2017.

[13] URL: https://docs.flutter.dev/resources/architectural-overview.

[14] Bonnie Eisenman. *Learning react native: Building native mobile apps with JavaScript*. " O'Reilly Media, Inc.", 2015.

[15] URL: https://dart.dev/platforms#optimized-production-code-dart-aot.

[16] URL: https://docs.flutter.dev/resources/architectural-overview#build-from-widget-to-element.

[17] URL: https://api.flutter.dev/flutter/rendering/RenderObject-class.html.

[18] Vedat Coskun, Busra Ozdenizci, and Kerem Ok. A survey on near field communication (nfc) technology. *Wireless personal communications*, 71(3):2259–2294, 2013.

[19] Vedat Coskun, Kerem Ok, and Busra Ozdenizci. *Near field communication (NFC): From theory to practice*. John Wiley & Sons, 2011.

[20] Gerald Madlmayr, Josef Langer, Christian Kantner, and Josef Scharinger. Nfc devices: Security and privacy. In *2008 Third International Conference on Availability, Reliability and Security*, pages 642–647. IEEE, 2008.

[21] Jawad Javed Akbar Baig, Sajjad Mahmood, Mohammad Alshayeb, and Mahmood Niazi. Package-level stability evaluation of object-oriented systems. *Information and Software Technology*, 116:106172, 2019.

[22] Frank Zammetti. *Practical Flutter*. Springer, 2019.

[23] Prajyot Mainkar and Salvatore Giordano. *Google Flutter Mobile Development Quick Start Guide: Get Up and Running with IOS and Android Mobile App Development*. Packt Publishing Ltd, 2019.

[24] URL: https://api.flutter.dev/flutter/meta/immutable-constant.html.

[25] URL: https://api.flutter.dev/flutter/widgets/Container-class.html.

[26] URL: https://api.flutter.dev/flutter/widgets/Icon/Icon.html.