By Minasie L.

Introduction to Flutter

Understanding Flutter

Flutter's architecture is designed to provide a high-performance, expressive, and flexible framework for building cross-platform applications.

At its core, Flutter follows a reactive and declarative programming paradigm.

Let's take a closer look at the overview of Flutter's architecture and how it renders UI:

Dart Language

Flutter is built using the Dart programming language.

Dart is a modern, object-oriented language with features like strong typing, hot reload, and a rich standard library.

Dart is compiled both to native code and JavaScript, enabling Flutter to run on multiple platforms.

Widget Tree

- In Flutter, everything is a widget.
- Widgets are the building blocks of the user interface, representing elements from simple components (such as buttons and text) to complex layouts.
- Flutter's widget tree is a hierarchical structure where each widget encapsulates part of the UI.
- Widgets are either stateful or stateless, and their composition forms the complete UI.

Reactive Framework

- Flutter follows a reactive programming model.
- When the state of a widget changes, Flutter automatically rebuilds the affected part of the widget tree.
- This reactive approach simplifies UI development, as developers only need to specify how the UI should look based on its current state.

Declarative UI

Flutter adopts a declarative approach to UI development.

Instead of imperatively describing how to transition from one state to another, developers declare the desired state of the UI.

Flutter takes care of efficiently updating the UI to reflect the declared state.

Flutter Engine

The Flutter engine is written in C++ and provides a low-level rendering framework.

It manages tasks such as graphics rendering, input, and event handling.

The engine communicates with the Dart runtime to execute Flutter applications.

Skia Graphics Engine

Flutter utilizes the Skia graphics engine to achieve high-performance rendering.

Skia is an open-source 2D graphics library that provides the drawing and rendering capabilities necessary for Flutter's UI.

How Flutter Renders UI?

Widget Building

Developers create a widget tree using Flutter's extensive set of pre-built widgets.

Each widget represents a part of the UI, from the smallest components to entire screens.

Element Tree

Flutter converts the widget tree into an element tree, where each widget is associated with a corresponding element.

Elements are the runtime representations of widgets and are responsible for rendering.

Render Objects

Render objects are lower-level counterparts to widgets.

They describe the layout and painting of UI elements on the screen.

Render objects are created based on the element tree.

Layout and Painting

Flutter's rendering engine performs layout and painting by traversing the render tree.

During the layout phase, Flutter determines the size and position of each render object.

The painting phase involves rendering visual elements onto the screen.

Hot Reload

Flutter's hot reload feature allows developers to make changes to the code and see the results instantly without restarting the entire application.

This iterative development process significantly speeds up the development cycle.

Conclusion

- By combining a reactive framework, a declarative UI approach, and a high-performance rendering engine, Flutter provides a powerful and efficient platform for developing cross-platform applications with a visually appealing and consistent user experience.
- The architecture's flexibility allows Flutter to target various platforms, including iOS, Android, web, and desktop.