Fabric

Fabric is React Native's new rendering system, a conceptual evolution of the legacy render system. The core principles are to unify more render logic in C++, improve interoperability with host platforms, and to unlock new capabilities for React Native. Development began in 2018 and in 2021, React Native in the Facebook app is backed by the new renderer.

This documentation provides an overview of the new renderer and its concepts. It avoids platform specifics and doesn't contain any code snippets or pointers. This documentation covers key concepts, motivation, benefits, and an overview of the render pipeline in different scenarios.

Motivations and Benefits of the new renderer

The render architecture was created to unlock better user experiences that weren't possible with the legacy architecture. Some examples include:

- With improved interoperability between host views and React views, the renderer is able to measure and render React surfaces synchronously. In the legacy architecture, React Native layout was asynchronous which led to a layout "jump" issue when embedding a React Native rendered view in a *host view*.
- With support of multi-priority and synchronous events, the renderer can prioritize certain user interactions to ensure they are handled in a timely manner.
- Integration with React Suspense which allows for more intuitive design of data fetching in React apps.
- Enable React Concurrent Features on React Native.
- Easier to implement server side rendering for React Native.

The new architecture also provides benefits in code quality, performance, and extensibility:

- **Type safety:** code generation to ensure type safety across the JS and host platforms. The code generation uses JavaScript component declarations as source of truth to generate C++ structs to hold the props. Mismatch between JavaScript and host component props triggers a build error.
- **Shared C++ core**: the renderer is implemented in C++ and the core is shared among platforms. This increases consistency and makes it easier to adopt React Native on new platforms.
- **Better Host Platform Interoperability**: Synchronous and thread-safe layout calculation improves user experiences when embedding host components into React Native, which means easier integration with host platform frameworks that require synchronous APIs.
- **Improved Performance**: With the new cross-platform implementation of the renderer system, every platform benefits from performance improvements that may have been motivated by limitations of one platform. For example, view flattening was originally a performance solution for Android and is now provided by default on both Android and iOS.
- **Consistency**: The new render system is cross-platform, it is easier to keep consistency among different platforms.
- **Faster Startup**: Host components are lazily initialized by default.
- **Less serialization of data between JS and host platform**: React used to transfer data between JavaScript and *host platform* as serialized JSON. The new renderer improves the transfer of data by accessing JavaScript values directly using JavaScript Interfaces (JSI).

Is this page useful?

Edit this page

Last updated on ***Mar 10, 2022***