Native Modules vs Turbo Modules

Edit

If you've worked with React Native, you may be familiar with the concept of Native Modules, which allow JavaScript and platform-native code to communicate over the React Native "bridge", which handles cross-platform serialization via JSON.

TurboModules are the next iteration of Native Modules that provide a few extra benefits, in particular these modules use JSI, a JavaScript interface for native code, which allows for more efficient communication between native and JavaScript code than the bridge.

How to migrate to TurboModules

Modules running as TurboModules will be available in the JS from `TurboModuleRegistry.get('<modulename>')` instead of `NativeModules.<modulename>`. So your JavaScript will have to be updated before switching. Ideally while you are at it, you should switch your modules to use Spec files. This will make your modules compatible with codegen.

> Note: `TurboModuleRegistry` will fallback to returning a native module instead of a turbo module if there is a native module registered from the native code. So you can update your JavaScript before updating your native code.

Starting in version 0.71, JS/TS spec files can codegen C++ spec files that can verify that the native implementation matches the definition in JS. In addition modules can now be run as

```
AddAttributedModules(packageBuilder, true);
```

Alternatively if you are registering modules more manually by calling `IReactPackageBuilder.AddModule`, you can call `IReactPackageBuilder.AddTurboModule` instead.

Additional differences running as Native Module vs TurboModule

After creating a spec in JS, you should see JS type errors showing that constants should be accessed using `MyModule.getConstants().myconst` instead of `MyModule.myconst`. If you fail to update you accesses of `myconst` the field will continue to work when the module is running as a Native Module, since Native Modules promote all the constants to fields on the module. This behavior does not happen with TurboModules, so the `myconst` field will be undefined. Calls using `getConstants().myconst` will work both for Native Modules and TurboModules.

Web Debugging Behavior

TurboModules cannot run when using Remote Debugging / Web Debugging. React-Native-Windows will attempt to run a TurboModule as a native module when running in that mode, but if the module is using JSI directly, that fallback may not work.

‹ Native Module Setup

Using Community Native Modules ›