Module imports for InPlace

Joop Ringelberg 26-06-20 Version: 1

Introduction

InPlace (project perspectives-react-integrated-client) imports a number of other Perspectives projects. In this text we provide an overview and describe how these projects are built into modules by Webpack. We also describe how these projects *externalise* common dependencies and what has been done to make them available.

Static and dynamic modules

Except for the screen modules that are loaded dynamically (through the use of react-loadable (https://github.com/jamiebuilds/react-loadable), all dependent modules are packed by Webpack. It turns out that dynamic loading severely limits our options for externalisation: only a value on the global scope will work.

Module relations

Module import relations form a straightforward tree:

- InPlace
  - core
    - perspectives-proxy
  - perspectives-proxy
  - perspectives-react
    - perspectives-proxy

Now if we add some non-perspectives dependencies that occur throughout the tree, we get:

- InPlace
  - core
    - perspectives-proxy
  - perspectives-proxy
  - perspectives-react
    - perspectives-proxy
    - React
    - prop-types
  - React
  - ReactDom
  - react-bootstrap

Dependencies in green are *externalised* by their importing modules. This is a Webpack concept. It allows us to decrease output bundle size because we state that a particular module will be provided *in the environment that consumes the module*.

Perspectives-proxy: a special case

Perspectives-proxy is a stateful module. It builds and keeps a connection to the core. We must only have a single instance of perspectives-proxy in a running Perspectives program, hence we externalise it in all modules except for the topmost one (being, in this case, InPlace).

Making externalising work

We've encountered some restrictions in the way we can externalise (perspectives) modules. First we list how we have Webpack build the various modules:

Module name	libraryTarget value
core	commonjs2
perspectives-proxy	umd
perspectives-react	commonjs2[1]

Webpack configuration can hold a key `libraryTarget` (https://webpack.js.org/configuration/output/#outputlibrarytarget) This determines the type of output that is produced, among them various module systems. Type umd (https://github.com/umdjs/umd) translates to commonjs, amd and a global variable.

While this is the way that these libraries are made available, below we list the way they are externalised by their consumers:

Module name	Imported by	Externalised as
perspectives-proxy	core, perspectives-react	commonjs, commonjs2, amd, root
react	perspectives-react	commonjs2
prop-types	perspectives-react	commonjs2
react	Perspectives-react-integrated-client	commonjs2

Why externalise 'react' from perspectives-react-integrated-client?

It seems as if the top-level module should **not** externalise react; otherwise, where does it come from? However, if we have Webpack include react into the bundle for perspectives-react-integrated-client, it will actually be instantiated twice in the renderer/browser.

[1] We found that giving perspectives-react a target of umd leads to a problem in InPlace.

This is because a module like perspectives-react (but also react-dom, that we have no control over!) actually obtains react through the `require` function of Electron. This latter function reads the react code from the node_modules directory.

So while the dependencies that have externalised react obtain it through the Electron module system, the main application obtains it from its own bundle. Both cache the react module – separately! The Webpack bundle has a 'runtime' that has its own exclusive cache, separate from the cache kept by Electron. Hence, we end up with two copies of react in memory. And that gives rise to problems with hooks[2].

While externalising react from the top-level is no problem for the Electron version, it might cause a problem with a browser-based version.

Externalising modules for screens

The project perspectives-screens produces modules with screen components for various models. There is a module for each model; each module holds one or more screens.

We have Webpack compile the module with libraryTarget `var`. We externalise the following modules, all in the form of a global variable:

- react
- perspectives-react
- react-dom
- react-bootstrap
- @primer/octicons-react
- prop-types

In order to make this work, we have to put these modules in variables on the global scope in InPlace. This is done in the module `externals.js`.

[2] https://reactjs.org/warnings/invalid-hook-call-warning.html