server-render

Source code of released version | Source code of development version ***

This package implements generic support for server-side rendering in Meteor apps, by providing a mechanism for injecting fragments of HTML into the `<head>` and/or `<body>` of the application's initial HTML response.

Usage

This package exports a function named `onPageLoad` which takes a callback function that will be called at page load (on the client) or whenever a new request happens (on the server).

The callback receives a `sink` object, which is an instance of either `ClientSink` or `ServerSink` depending on the environment. Both types of `sink` have the same methods, though the server version accepts only HTML strings as content, whereas the client version also accepts DOM nodes.

The current interface of `{Client,Server}Sink` objects is as follows:

```
class Sink {
  // Appends content to the <head>.
  appendToHead(content)

  // Appends content to the <body>.
  appendToBody(content)

  // Appends content to the identified element.
  appendToElementById(id, content)

  // Replaces the content of the identified element.
  renderIntoElementById(id, content)

  // Redirects request to new location.
  redirect(location, code)


  // server only methods

  // sets the status code of the response.
  setStatusCode(code)

  // sets a header of the response.
  setHeader(key, value)

  // gets request headers
  getHeaders()
```

```
  // gets request cookies
  getCookies()
}
```

The `sink` object may also expose additional properties depending on the environment. For example, on the server, `sink.request` provides access to the current `request` object, and `sink.arch` identifies the target architecture of the pending HTTP response (e.g. "web.browser").

Here is a basic example of `onPageLoad` usage on the server:

```
import React from "react";
import { renderToString } from "react-dom/server";
import { onPageLoad } from "meteor/server-render";
import App from "/imports/Server.js";

onPageLoad(sink => {
  sink.renderIntoElementById("app", renderToString(
    <App location={sink.request.url} />
  ));
});
```

Likewise on the client:

```
import React from "react";
import ReactDOM from "react-dom";
import { onPageLoad } from "meteor/server-render";

onPageLoad(async sink => {
  const App = (await import("/imports/Client.js")).default;
  ReactDOM.hydrate(
    <App />,
    document.getElementById("app")
  );
});
```

Note that the `onPageLoad` callback function is allowed to return a `Promise` if it needs to do any asynchronous work, and thus may be implemented by an `async` function (as in the client case above).

Note also that the client example does not end up calling any methods of the `sink` object, because `ReactDOM.hydrate` has its own similar API. In fact, you are not even required to use the `onPageLoad` API on the client, if you have your own ideas about how the client should do its rendering.

Here is a more complicated example of `onPageLoad` usage on the server, involving the `styled-components` npm package:

```
import React from "react";
```

```
import { onPageLoad } from "meteor/server-render";
import { renderToString } from "react-dom/server";
import { ServerStyleSheet } from "styled-components"
import App from "/imports/Server";

onPageLoad(sink => {
  const sheet = new ServerStyleSheet();
  const html = renderToString(sheet.collectStyles(
    <App location={sink.request.url} />
  ));

  sink.renderIntoElementById("app", html);
  sink.appendToHead(sheet.getStyleTags());
});
```

In this example, the callback not only renders the `<App />` element into the element with `id="app"`, but also appends any `<style>` tag(s) generated during rendering to the `<head>` of the response document.

Although these examples have all involved React, the `onPageLoad` API is designed to be generically useful for any kind of server-side rendering.

React 16 `renderToNodeStream` Since React 16, it is possible to render a React app to a node stream which can be piped to the response. This can decrease time to first byte, and improve performance of server rendered apps.

Here is a basic example of using streams:

```
import React from "react";
import { renderToNodeStream } from "react-dom/server";
import { onPageLoad } from "meteor/server-render";
import App from "/imports/Server.js";

onPageLoad(sink => {
  sink.renderIntoElementById("app", renderToNodeStream(
    <App location={sink.request.url} />
  ));
});
```