Query Extraction

This documentation isn't up to date with the latest version of Gatsby.

Outdated areas are:

- queries in dependencies (node_modules) and themes are now extracted as well
- add meta key for hook in JSON in diagram

You can help by making a PR to update this documentation.

Extracting queries from files

Up until now, Gatsby has sourced all nodes into Redux, inferred a schema from them, and created all pages. The next step is to extract and compile all GraphQL queries from your source files. The entrypoint to this phase is query-watcher extractQueries(), which immediately compiles all GraphQL queries by calling into query-compiler.js.

Query compilation

The first thing it does is use babylon-traverse to load all JavaScript files in the site that have GraphQL queries in them. This produces AST results that are passed to the relay-compiler. This accomplishes a couple of things:

1. It informs us of any malformed queries, which are promptly reported back to the user.
2. It builds a tree of queries and fragments they depend on. And outputs a single optimized query string with the fragments.

After this step, Gatsby will have a map of file paths (of site files with queries in them) to Query Objects, which contain the raw optimized query text, as well as other metadata such as the component path and page `jsonName`. The following diagram shows the flow involved during query compilation

```
digraph {
  fragments [ label = "fragments. e.g\l.cache/fragments/fragment1.js", shape = cylinder ];
  srcFiles [ label = "source files. e.g\lsrc/pages/my-page.js", shape = cylinder ];
  components [ label = "redux.state.components\l(via createPage)", shape = cylinder ];
  schema [ label = "Gatsby schema", shape = cylinder, URL = "/docs/schema-generation/" ];
```

```
  subgraph cluster_compiler {
    label = "query-compiler.js";
    fileQueries [ label = "files containing queries", shape = box ];
    babylon [ label = "parse files with babylon\lfilter those with queries" ];
    queryAst [ label = "QueryASTs", shape = box ];
    relayCompiler [ label = "Relay Compiler" ];
    queries [ label = "{ Queries | { filePath | <query> query } }", shape = record ];
    query [ label = "{\l    name: filePath,\l    text: rawQueryText,\l    originalText: orig

  }

  fileQueries -> babylon;
  babylon -> queryAst;
  queryAst -> relayCompiler;
  relayCompiler -> queries;
  queries:query -> query;
  fragments -> fileQueries;
  srcFiles -> fileQueries;
  components -> fileQueries;
  schema -> relayCompiler;

  fragments -> srcFiles [ style = invis ];
  fragments -> components [ style = invis ];
}
```

Store queries in Redux

Gatsby is now in the handleQuery function.

If the query is a `StaticQuery`, Gatsby will call the `replaceStaticQuery` action to save it to the `staticQueryComponents` namespace which is a mapping from a component's path to an object that contains the raw GraphQL Query amongst other things. More details can be found in the doc on Static Queries. Gatsby also removes a component's `jsonName` from the `components` Redux namespace. See Page -> Node Dependencies.

If the query is just a normal every-day query (not StaticQuery), then Gatsby updates its component's `query` in the Redux `components` namespace via the replaceComponentQuery action.

```
digraph {
  compound = true;

  compiler [ label = "query-compiler.js" ];

  subgraph cluster_watcher {
```

```
    label = "query-watcher.js:handleQuery()"
    query [ label = "{\l    name: filePath,\l    text: rawQueryText,\l    originalText: orig
    replaceStaticQuery [ label = "replaceStaticQuery()" ];
    staticQueryComponents [ label = "staticQueryComponents\l (redux)", shape = cylinder ];
    replaceComponentQuery [ label = "replaceComponentQuery()" ];
    components [ label = "components\l (redux)", shape = cylinder ];

    query -> replaceStaticQuery [ label = "if static query" ];
    query -> replaceComponentQuery [ label = "if not static" ];
    replaceStaticQuery -> staticQueryComponents;
    replaceComponentQuery -> components [ label = "set `query` attribute" ];
  }

  compiler -> query [ label = "for each compiled query", lhead = cluster_watcher ];
}
```

Queue for execution

Now that Gatsby has saved your query, it's ready to queue for execution. Query execution is mainly handled by page-query-runner.ts, so it accomplishes this by passing the component's path to `queueQueryForPathname` function.

```
digraph {
  compound = true;
  compiler [ label = "query-compiler.js" ];

  subgraph cluster_watcher {
    label = "query-watcher.js:handleQuery()"
    query [ label = "{\l    name: filePath,\l    text: rawQueryText,\l    originalText: orig

  }

  subgraph cluster_pageQueryRunner {
    label = "page-query-runner.js"
    queueQueryForPathname [ label = "queueQueryForPathname()" ];
  }

  compiler -> query [ label = "for each compiled query", lhead = cluster_watcher ];
  query -> queueQueryForPathname [ label = "queue for execution" ];
}
```

Now let's learn about Query Execution.