Dll scope hoisting

DllPlugin documentation

This example demonstrates the usage of `entryOnly` option in combination with module concatenation / scope hoisting.

By default, `DllPlugin` exposes all the modules referenced in the bundle as separate entries. The manifest includes the individual modules available for use by `DllReferencePlugin`. Since all the modules are being accounted for, this prevents advanced optimizations such as tree shaking.

The `entryOnly` flag tells `DllPlugin` to only expose the modules which are configured as entry points; this affects both the manifest and the resulting bundle. Since some of the modules are no longer included in the "public contract" of the Dll, they can be optimized by merging (concatenating) multiple modules together or removing unused code. This allows taking advantage of tree shaking (scope hoisting and dead code removal) optimizations.

In this example, only `example.js` module is exposed, since it's the entry point. Modules `a.js` and `b.js` are concatenated into `example.js`. Module `cjs.js` is left as is since it's in CommonJS format.

The manifest includes `example.js` as the only exposed module and lists the exports as `["a","b","c"]` from the corresponding modules `a.js`, `b.js`, and `cjs.js`. None of the other modules are exposed.

Also, see tree shaking and scope hoisting example.

example.js

```
_{{example.js}}_
```

webpack.config.js

```
_{{webpack.config.js}}_
```

dist/dll.js

```
_{{dist/dll.js}}_
```

dist/dll-manifest.json

```
_{{dist/dll-manifest.json}}_
```

Info

Unoptimized

```
_{{stdout}}_
```

Production mode

```
_{{production:stdout}}_
```