caching-compiler

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

Caching superclasses for Plugin.registerCompiler. CachingCompiler and MultiFileCachingCompiler are classes designed to be used with Plugin.registerCompiler. They implement in-memory and on-disk caches for the files that they process.

CachingCompiler is for compilers whose files can be processed independently of each other (ie, editing one such file only requires that file to be rebuilt). MultiFileCachingCompiler is for compilers whose files can depend on each other, such as CSS preprocessors with `@import` directives.

You should subclass CachingCompiler and define the following methods: `getCacheKey`, `compileOneFile`, `addCompileResult`, and `compileResultSize`.

CompileResult

The data that is cached for each file is of a type that is (implicitly) defined by your subclass. `caching-compiler` refers to this type as `CompileResult`, but this isn't a single type: it's up to your subclass to decide what type of data this is. You should document what your subclass's `CompileResult` type is.

Example usage

Your subclass's compiler should call the superclass compiler specifying the compiler name (used to generate environment variables for debugging and tweaking in-memory cache size) and the default cache size.

By default, CachingCompiler processes each file in "parallel". That is, if it needs to yield to read from the disk cache, or if getCacheKey, compileOneFile, or addCompileResult yields, it will start processing the next few files. To set how many files can be processed in parallel (including setting it to 1 if your subclass doesn't support any parallelism), pass the maxParallelism option to the superclass constructor.

For example (using ES2015 via the ecmascript package):

```
// CompileResult is a {source, sourceMap} object.
class AwesomeCompiler extends CachingCompiler {
  constructor() {
    super({
      compilerName: 'awesome',
      defaultCacheSize: 1024*1024*10,
    });
  }
  getCacheKey(inputFile) {
    return inputFile.getSourceHash();
```

```
  }
  compileResultSize(compileResult) {
    return compileResult.source.length + compileResult.sourceMap.length;
  }
  compileOneFile(inputFile) {
    return Awesomifier.compile(inputFile.getContentsAsString());
  }
  addCompileResult(inputFile, compileResult) {
    inputFile.addJavaScript({
      path: inputFile.getPathInPackage() + '.js',
      sourcePath: inputFile.getPathInPackage(),
      data: compileResult.source,
      sourceMap: compileResult.sourceMap,
    });
  }
}
Plugin.registerCompile({
  extensions: ['awesome'],
}, () => new AwesomeCompiler());
```