```
Ax 3 contributors 🚱 😭 🌘
```

```
406 lines (369 sloc) | 10.9 KB
  1
       "format cjs";
  2
  3
      var crawl = require('./npm-crawl');
  4
      var utils = require("./npm-utils");
  5
  6
      exports.steal = convertSteal;
  7
      exports.propertyNames = convertPropertyNames;
      exports.propertyNamesAndValues = convertPropertyNamesAndValues;
  9
      exports.name = convertName;
      exports.browser = convertBrowser;
 10
 11
      exports.browserProperty = convertBrowserProperty;
 12
      exports.jspm = convertJspm;
 13
      exports.toPackage = convertToPackage;
 14
      exports.forPackage = convertForPackage;
 15
 16
      function StealConversion(context, pkg, steal, config, isRoot, waiting) {
 17
          this.context = context;
 18
          this.pkg = pkg;
 19
          this.steal = steal;
 20
          this.config = utils.extend({}, steal, true);
 21
          this.isRoot = isRoot;
 22
          this.waiting = [];
 23
      }
 24
 25
      // Translate helpers ========
 26
      // Given all the package.json data, these helpers help convert it to a source.
 27
       function convertSteal(context, pkg, steal, root, ignoreWaiting, resavePackageInfo) {
 28
           if(!steal) {
               return new StealConversion(context, pkg, steal, root);
 29
```

```
30
         }
31
32
         var conv = new StealConversion(context, pkg, steal, root);
33
         var waiting = conv.waiting;
34
35
         if(steal.meta) {
36
             steal.meta = convertPropertyNames(context, pkg, steal.meta, root,
37
                                                 waiting);
38
         }
39
         if(steal.map) {
40
             steal.map = convertPropertyNamesAndValues(context, pkg, steal.map,
41
                                                         root, waiting);
42
         }
43
         if(steal.paths) {
             steal.paths = convertPropertyNames(context, pkg, steal.paths, root,
45
                                                 waiting);
46
         }
47
         // needed for builds
48
         if(steal.buildConfig) {
49
             var buildConv = convertSteal(context, pkg, steal.buildConfig, root);
50
             conv.buildConversion = buildConv;
51
52
             steal.buildConfig = conv.config;
53
         }
54
55
         return conv;
56
     }
57
58
     var lazyConfig = {
59
         // Queue package reconfiguration whenever a package is first loaded.
60
         // This is for progressively loaded package.jsons
         updateConfigOnPackageLoad: function(conv, isPackageInfoSaved,
61
62
             isConfigApplied, isBuildConfigApplied) {
63
             var fns = [function(){return lazyConfig.cloneConversion.call(this, conv)}];
64
             if(isPackageInfoSaved) {
65
                 fns.push(lazyConfig.resavePackageInfo);
66
             }
67
             if(isConfigApplied) {
68
                 fns.push(lazyConfig.applyConfig);
69
             }
70
             var fn = utils.flow(fns);
71
72
             convertLater(conv.context, conv.waiting, fn);
73
74
             if(isBuildConfigApplied && conv.buildConversion) {
75
                 var c = conv.buildConversion;
76
                 fn = utils.flow([
77
                      function(){return lazyConfig.cloneConversion.call(this, c)},
78
                      lazyConfig.resavePackageInfo,
```

```
79
                       lazyConfig.applyConfig
80
                   1);
81
                   convertLater(c.context, c.waiting, fn);
              }
82
83
          },
84
          resavePackageInfo: function(conv) {
85
              var info = utils.pkg.findPackageInfo(conv.context, conv.pkg);
86
              info.steal = info.system = conv.steal;
87
              return conv;
          },
88
89
          applyConfig: function(conv) {
90
              var config = conv.steal;
91
              var context = this;
92
93
              // Temporarily remove steal.main so that it doesn't set System.main
94
              var stealMain = config.main;
95
              delete config.main;
96
              delete config.transpiler;
97
              context.loader.config(config);
98
              config.main = stealMain;
99
              return conv;
100
          },
101
          cloneConversion: function(conv) {
102
              var context = this;
103
              var local = utils.extend({}, conv.config, true);
              var lConv = convertSteal(context, conv.pkg, local, conv.isRoot);
104
105
              return 1Conv;
106
          }
      };
107
108
109
      exports.updateConfigOnPackageLoad = lazyConfig.updateConfigOnPackageLoad;
110
111
      // converts only the property name
112
      function convertPropertyNames (context, pkg, map , root, waiting) {
113
          if(!map) {
              return map;
114
115
          }
116
          var clone = {}, value;
117
          for(var property in map ) {
118
              value = convertName(context, pkg, map, root, property, waiting);
119
              if(typeof value === 'string') {
120
                  clone[value] = map[property];
121
              }
122
123
              // do root paths b/c we don't know if they are going to be included with the package name
124
              if(root) {
125
                  value = convertName(context, pkg, map, false, property, waiting);
126
                  if(typeof value === 'string') {
                       clone[value] = map[property];
127
```

```
128
                  }
129
              }
130
          }
131
          return clone;
132
      }
133
134
      // converts both property name and value
135
      function convertPropertyNamesAndValues (context, pkg, map, root, waiting) {
136
          if(!map) {
137
              return map;
138
          }
139
          var clone = {}, val, name;
140
          for(var property in map ) {
141
              val = map[property];
142
              name = convertName(context, pkg, map, root, property, waiting);
              val = typeof val === "object"
143
144
                   ? convertPropertyNamesAndValues(context, pkg, val, root, waiting)
145
                   : convertName(context, pkg, map, root, val, waiting);
              if(typeof name !== 'undefined' && typeof val !== 'undefined') {
146
147
                  clone[name] = val;
148
              }
149
              // keep map entry if the key isn't a package but value might
              if (name && typeof val === "undefined") {
150
151
                  clone[name] = map[property];
152
              }
153
          }
154
          return clone;
155
      }
156
157
      function convertName (context, pkg, map, root, name, waiting) {
158
          var parsed = utils.moduleName.parse(name, pkg.name, null, context),
159
              depPkg, requestedVersion;
160
          if( name.indexOf("#") >= 0 ) {
161
              // If this is a fully converted name just return the name.
162
              if(utils.moduleName.isFullyConvertedNpm(parsed)) {
                  return name;
163
164
              }
165
166
              if(parsed.packageName === pkg.name) {
167
                  parsed.version = pkg.version;
168
              } else {
169
                  // Get the requested version's actual version.
170
                  requestedVersion = crawl.getDependencyMap(context.loader, pkg, root)[parsed.packageNam
171
                  depPkg = crawl.matchedVersion(context, parsed.packageName, requestedVersion);
172
                  // This hasn't been crawled yet, so convert later
173
                  if(!depPkg) {
174
                      waiting.push({
175
                           packageName: parsed.packageName,
176
                           requestedVersion: requestedVersion
```

```
177
                       });
178
                       return;
179
180
                  parsed.version = depPkg.version;
181
              }
182
              return utils.moduleName.create(parsed);
183
          } else {
184
185
              if(root && name.substr(0,2) === "./" ) {
186
                  return name.substr(2);
187
              } else {
188
                  // this is for a module within the package
                  if (name.substr(0,2) === "./" ) {
189
190
                       return utils.moduleName.create({
191
                           packageName: pkg.name,
192
                           modulePath: name,
193
                           version: pkg.version,
194
                           plugin: parsed.plugin
195
                      });
196
                  } else {
                       // SYSTEM.NAME
197
198
                       if( pkg.name === parsed.packageName || ( (pkg.system && pkg.system.name) === pars
                           depPkg = pkg;
199
200
                      } else {
201
                           var requestedProject = crawl.getDependencyMap(context.loader, pkg, root)[parse
202
                           if(!requestedProject) {
203
                               return name;
204
                           requestedVersion = requestedProject.version;
205
206
                           depPkg = crawl.matchedVersion(context, parsed.packageName, requestedVersion);
207
                           // If we still didn't find one just use the first available version.
208
                           if(!depPkg) {
209
                               var versions = context.versions[parsed.packageName];
210
                               depPkg = versions && versions[requestedVersion];
211
212
                               if(!depPkg) {
213
                                   waiting.push({
214
                                       packageName: parsed.packageName,
215
                                       requestedVersion: requestedVersion
216
                                   });
217
                                   return;
218
                               }
219
                           }
220
                       }
221
                       // SYSTEM.NAME
222
                       if(depPkg.system && depPkg.system.name) {
223
                           parsed.packageName = depPkg.system.name;
224
                       }
225
```

```
226
                      parsed.version = depPkg.version;
227
                      if(!parsed.modulePath) {
228
                           parsed.modulePath = utils.pkg.main(depPkg);
                      }
229
                      return utils.moduleName.create(parsed);
230
231
                  }
232
              }
233
234
235
          }
236
      }
237
238
239
       * Converts browser names into actual module names.
240
       * Example:
241
242
243
       * {
244
           "foo": "browser-foo"
245
246
           "traceur#src/node/traceur": "./browser/traceur"
           "./foo" : "./foo-browser"
247
       * }
248
249
250
251
       * converted to:
252
       * ```
253
       * {
254
         // any foo ... regardless of where
255
256
          "foo": "browser-foo"
257
       * // this module ... ideally minus version
           "traceur#src/node/traceur": "transpile#./browser/traceur"
258
       * "transpile#./foo" : "transpile#./foo-browser"
259
260
       * }
261
262
       */
263
      function convertBrowser(pkg, browser) {
264
          var type = typeof browser;
265
          if(type === "string" || type === "undefined") {
266
              return browser;
267
          }
268
          var map = {};
269
          for(var fromName in browser) {
270
              convertBrowserProperty(map, pkg, fromName, browser[fromName]);
271
          }
272
          return map;
273
      }
274
```

```
275
276
      function convertBrowserProperty(map, pkg, fromName, toName) {
277
          var packageName = pkg.name;
278
279
          var fromParsed = utils.moduleName.parse(fromName, packageName);
280
          var toResult = toName;
281
          if(!toName || typeof toName === "string") {
282
283
              var toParsed = toName ? utils.moduleName.parse(toName, packageName)
284
                   : "@empty";
285
              toResult = utils.moduleName.create(toParsed);
286
          } else if(utils.isArray(toName)) {
287
              toResult = toName;
288
          }
289
290
          map[utils.moduleName.create(fromParsed)] = toResult;
291
      }
292
293
      function convertJspm(pkg, jspm){
294
          var type = typeof jspm;
295
          if(type === "undefined" || type === "string") {
296
              return jspm;
297
          }
298
          return {
299
              main: jspm.main
300
          };
301
      }
302
303
304
      function convertToPackage(context, npmPkg, index) {
305
          var pkg = npmPkg;
          var packages = context.pkgInfo;
306
307
          var nameAndVersion = pkg.name+"@"+pkg.version;
308
          var localPkg;
309
          if(!packages[nameAndVersion]) {
              crawl.setVersionsConfig(context, pkg, pkg.version);
310
311
              if(pkg.browser){
312
                  delete pkg.browser.transform;
313
              }
314
              // fake load obj, because we don't have one here
315
              pkg = utils.json.transform(context.loader, {
316
                  address: pkg.fileUrl,
317
                  name: pkg.fileUrl.split('/').pop(),
318
                  metadata: {}
319
              }, pkg);
320
              var steal = utils.pkg.config(pkg);
321
              var stealConversion = convertSteal(context, pkg, steal, index === 0);
322
              lazyConfig.updateConfigOnPackageLoad(stealConversion, context.resavePackageInfo,
                  true, context.applyBuildConfig);
323
```

```
324
325
              localPkg = {
326
                  name: pkg.name,
                  version: pkg.version,
327
                  fileUrl: utils.path.isRelative(pkg.fileUrl) ?
328
329
                       pkg.fileUrl :
330
                       utils.relativeURI(context.loader.baseURL, pkg.fileUrl),
                  main: pkg.main,
331
332
                   steal: stealConversion.steal,
333
                   globalBrowser: convertBrowser(pkg, pkg.globalBrowser),
334
                  browser: convertBrowser(pkg, pkg.browser || pkg.browserify),
335
                  jspm: convertJspm(pkg, pkg.jspm),
336
                   jam: convertJspm(pkg, pkg.jam),
337
                  resolutions: {}
338
              };
339
              packages.push(localPkg);
340
              packages[nameAndVersion] = true;
341
342
              localPkg = utils.filter(packages, function(lpkg){
343
                  if(pkg.name === lpkg.name && pkg.version === lpkg.version) {
344
                       return lpkg;
345
                   }
              })[0];
346
347
          }
348
          return localPkg;
349
      }
350
351
352
       * waiting looks like:
353
354
355
           { packageName: 'can', requestedVersion: '^2.3.0' },
356
           { packageName: 'lodash': 'requestedVersion': '~3.0.0' }
       * ]
357
358
359
       * Pushes these objects into a side table. Whenver a package.json is loaded
360
       * it will convert and apply config.
361
       */
362
      function convertLater(context, waiting, fn) {
363
          utils.forEach(waiting, function(p){
364
              var packageName = p.packageName;
365
              var requestedVersion = p.requestedVersion;
366
367
              var conv = context.deferredConversions;
368
              var pkg = conv[packageName];
369
              if(!pkg) pkg = conv[packageName] = {};
370
              var vers = pkg[requestedVersion];
              if(!vers) vers = pkg[requestedVersion] = [];
371
372
              vers.push(fn);
```

```
373
          });
374
      }
375
376
377
       ^{st} When progressively loading package.jsons we need to convert any config
378
       * that is waiting on a package.json to load. This function is called after
379
       * a package is loaded and will call all of the callbacks that cause the
       * config to be applied.
380
       */
381
382
      function convertForPackage(context, pkg) {
383
          var name = pkg.name;
          var version = pkg.version;
384
385
          var conv = context.deferredConversions;
386
387
          var pkgConv = conv[name];
          var depPkg, fns, keys = 0;
388
389
          if(pkgConv) {
390
              for(var range in pkgConv) {
                   depPkg = crawl.matchedVersion(context, name, range);
391
392
                   if(depPkg) {
393
                       fns = pkgConv[range];
                       for(var i = 0, len = fns.length; i < len; i++) {</pre>
394
                           fns[i].call(context);
395
396
                       }
397
                       delete pkgConv[range];
398
                   } else {
399
                       keys++;
400
                   }
401
              }
              if(keys === 0) {
402
403
                   delete conv[name];
404
              }
405
          }
406
      }
```