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;
8  exports.propertyNamesAndValues = convertPropertyNamesAndValues;
9  exports.name = convertName;
10 exports.browser = convertBrowser;
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) {
29         return new StealConversion(context, pkg, steal, root);
```

```
    }

    var conv = new StealConversion(context, pkg, steal, root);
    var waiting = conv.waiting;

    if(steal.meta) {
        steal.meta = convertPropertyNames(context, pkg, steal.meta, root,
                                          waiting);
    }
    if(steal.map) {
        steal.map = convertPropertyNamesAndValues(context, pkg, steal.map,
                                                  root, waiting);
    }
    if(steal.paths) {
        steal.paths = convertPropertyNames(context, pkg, steal.paths, root,
                                           waiting);
    }
    // needed for builds
    if(steal.buildConfig) {
        var buildConv = convertSteal(context, pkg, steal.buildConfig, root);
        conv.buildConversion = buildConv;

        steal.buildConfig = conv.config;
    }

    return conv;
}

var lazyConfig = {
    // Queue package reconfiguration whenever a package is first loaded.
    // This is for progressively loaded package.jsons
    updateConfigOnPackageLoad: function(conv, isPackageInfoSaved,
        isConfigApplied, isBuildConfigApplied) {
        var fns = [function(){return lazyConfig.cloneConversion.call(this, conv)}];
        if(isPackageInfoSaved) {
            fns.push(lazyConfig.resavePackageInfo);
        }
        if(isConfigApplied) {
            fns.push(lazyConfig.applyConfig);
        }
        var fn = utils.flow(fns);

        convertLater(conv.context, conv.waiting, fn);

        if(isBuildConfigApplied && conv.buildConversion) {
            var c = conv.buildConversion;
            fn = utils.flow([
                function(){return lazyConfig.cloneConversion.call(this, c)},
                lazyConfig.resavePackageInfo,
```

```
                lazyConfig.applyConfig
            ]);
            convertLater(c.context, c.waiting, fn);
        }
    },
    resavePackageInfo: function(conv) {
        var info = utils.pkg.findPackageInfo(conv.context, conv.pkg);
        info.steal = info.system = conv.steal;
        return conv;
    },
    applyConfig: function(conv) {
        var config = conv.steal;
        var context = this;

        // Temporarily remove steal.main so that it doesn't set System.main
        var stealMain = config.main;
        delete config.main;
        delete config.transpiler;
        context.loader.config(config);
        config.main = stealMain;
        return conv;
    },
    cloneConversion: function(conv) {
        var context = this;
        var local = utils.extend({}, conv.config, true);
        var lConv = convertSteal(context, conv.pkg, local, conv.isRoot);
        return lConv;
    }
};

exports.updateConfigOnPackageLoad = lazyConfig.updateConfigOnPackageLoad;

// converts only the property name
function convertPropertyNames (context, pkg, map , root, waiting) {
    if(!map) {
        return map;
    }
    var clone = {}, value;
    for(var property in map ) {
        value = convertName(context, pkg, map, root, property, waiting);
        if(typeof value === 'string') {
            clone[value] = map[property];
        }

        // do root paths b/c we don't know if they are going to be included with the package name
        if(root) {
            value = convertName(context, pkg, map, false, property, waiting);
            if(typeof value === 'string') {
                clone[value] = map[property];
```

```
            }
        }
    }
    return clone;
}

// converts both property name and value
function convertPropertyNamesAndValues (context, pkg, map, root, waiting) {
    if(!map) {
        return map;
    }
    var clone = {}, val, name;
    for(var property in map ) {
        val = map[property];
        name = convertName(context, pkg, map, root, property, waiting);
        val = typeof val === "object"
            ? convertPropertyNamesAndValues(context, pkg, val, root, waiting)
            : convertName(context, pkg, map, root, val, waiting);
        if(typeof name !== 'undefined' && typeof val !== 'undefined') {
            clone[name] = val;
        }
        // keep map entry if the key isn't a package but value might
        if (name && typeof val === "undefined") {
            clone[name] = map[property];
        }
    }
    return clone;
}

function convertName (context, pkg, map, root, name, waiting) {
    var parsed = utils.moduleName.parse(name, pkg.name, null, context),
        depPkg, requestedVersion;
    if( name.indexOf("#") >= 0 ) {
        // If this is a fully converted name just return the name.
        if(utils.moduleName.isFullyConvertedNpm(parsed)) {
            return name;
        }

        if(parsed.packageName === pkg.name) {
            parsed.version = pkg.version;
        } else {
            // Get the requested version's actual version.
            requestedVersion = crawl.getDependencyMap(context.loader, pkg, root)[parsed.packageNam
            depPkg = crawl.matchedVersion(context, parsed.packageName, requestedVersion);
            // This hasn't been crawled yet, so convert later
            if(!depPkg) {
                waiting.push({
                    packageName: parsed.packageName,
                    requestedVersion: requestedVersion
```

```
177                 });
178                 return;
179             }
180             parsed.version = depPkg.version;
181         }
182         return utils.moduleName.create(parsed);
183
184     } else {
185         if(root && name.substr(0,2) === "./" ) {
186             return name.substr(2);
187         } else {
188             // this is for a module within the package
189             if (name.substr(0,2) === "./" ) {
190                 return utils.moduleName.create({
191                     packageName: pkg.name,
192                     modulePath: name,
193                     version: pkg.version,
194                     plugin: parsed.plugin
195                 });
196             } else {
197                 // SYSTEM.NAME
198                 if(  pkg.name === parsed.packageName || ( (pkg.system && pkg.system.name) === pars
199                     depPkg = pkg;
200                 } else {
201                     var requestedProject = crawl.getDependencyMap(context.loader, pkg, root)[parse
202                     if(!requestedProject) {
203                         return name;
204                     }
205                     requestedVersion = requestedProject.version;
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
```

```
                parsed.version = depPkg.version;
                if(!parsed.modulePath) {
                    parsed.modulePath = utils.pkg.main(depPkg);
                }
                return utils.moduleName.create(parsed);
            }

        }

    }
}

/**
 * Converts browser names into actual module names.
 *
 * Example:
 *
 * ```
 * {
 *   "foo": "browser-foo"
 *   "traceur#src/node/traceur": "./browser/traceur"
 *   "./foo" : "./foo-browser"
 * }
 * ```
 *
 * converted to:
 *
 * ```
 * {
 *   // any foo ... regardless of where
 *   "foo": "browser-foo"
 *   // this module ... ideally minus version
 *   "traceur#src/node/traceur": "transpile#./browser/traceur"
 *   "transpile#./foo" : "transpile#./foo-browser"
 * }
 * ```
 */
function convertBrowser(pkg, browser) {
    var type = typeof browser;
    if(type === "string" || type === "undefined") {
        return browser;
    }
    var map = {};
    for(var fromName in browser) {
        convertBrowserProperty(map, pkg, fromName, browser[fromName]);
    }
    return map;
}

```

```
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
282     if(!toName || typeof toName === "string") {
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;
306     var packages = context.pkgInfo;
307     var nameAndVersion = pkg.name+"@"+pkg.version;
308     var localPkg;
309     if(!packages[nameAndVersion]) {
310         crawl.setVersionsConfig(context, pkg, pkg.version);
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,
323             true, context.applyBuildConfig);
```

```
        localPkg = {
            name: pkg.name,
            version: pkg.version,
            fileUrl: utils.path.isRelative(pkg.fileUrl) ?
                pkg.fileUrl :
                utils.relativeURI(context.loader.baseURL, pkg.fileUrl),
            main: pkg.main,
            steal: stealConversion.steal,
            globalBrowser: convertBrowser(pkg, pkg.globalBrowser),
            browser: convertBrowser(pkg, pkg.browser || pkg.browserify),
            jspm: convertJspm(pkg, pkg.jspm),
            jam: convertJspm(pkg, pkg.jam),
            resolutions: {}
        };
        packages.push(localPkg);
        packages[nameAndVersion] = true;
    } else {
        localPkg = utils.filter(packages, function(lpkg){
            if(pkg.name === lpkg.name && pkg.version === lpkg.version) {
                return lpkg;
            }
        })[0];
    }
    return localPkg;
}

/**
 * waiting looks like:
 *
 * [
 *   { packageName: 'can', requestedVersion: '^2.3.0' },
 *   { packageName: 'lodash': 'requestedVersion': '~3.0.0' }
 * ]
 *
 * Pushes these objects into a side table. Whenver a package.json is loaded
 * it will convert and apply config.
 */
function convertLater(context, waiting, fn) {
    utils.forEach(waiting, function(p){
        var packageName = p.packageName;
        var requestedVersion = p.requestedVersion;

        var conv = context.deferredConversions;
        var pkg = conv[packageName];
        if(!pkg) pkg = conv[packageName] = {};
        var vers = pkg[requestedVersion];
        if(!vers) vers = pkg[requestedVersion] = [];
        vers.push(fn);
```

```
373     });
374 }
375
376 /**
377  * 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
380  * config to be applied.
381  */
382 function convertForPackage(context, pkg) {
383     var name = pkg.name;
384     var version = pkg.version;
385
386     var conv = context.deferredConversions;
387     var pkgConv = conv[name];
388     var depPkg, fns, keys = 0;
389     if(pkgConv) {
390         for(var range in pkgConv) {
391             depPkg = crawl.matchedVersion(context, name, range);
392             if(depPkg) {
393                 fns = pkgConv[range];
394                 for(var i = 0, len = fns.length; i < len; i++) {
395                     fns[i].call(context);
396                 }
397                 delete pkgConv[range];
398             } else {
399                 keys++;
400             }
401         }
402         if(keys === 0) {
403             delete conv[name];
404         }
405     }
406 }
```