carlosvillademor fix: add null checks before removing parent refs ... History

5 contributors

366 lines (327 sloc) | 11.6 KB ...

```
/*
 Copyrights for code authored by Yahoo! Inc. is licensed under the following
 terms:

 MIT License

 Copyright (c) 2011-2012 Yahoo! Inc. All Rights Reserved.

 Permission is hereby granted, free of charge, to any person obtaining a copy
 of this software and associated documentation files (the "Software"), to
 deal in the Software without restriction, including without limitation the
 rights to use, copy, modify, merge, publish, distribute, sublicense, and/or
 sell copies of the Software, and to permit persons to whom the Software is
 furnished to do so, subject to the following conditions:

 The above copyright notice and this permission notice shall be included in
 all copies or substantial portions of the Software.

 THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
 IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
 FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
 AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
 LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING
 FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER
 DEALINGS IN THE SOFTWARE.
*/

/*
 * A library that enables the hooking of the standard 'require' function, such
```

```
 * that a (possibly partial) mock implementation can be provided instead. This
 * is most useful for running unit tests, since any dependency obtained through
 * 'require' can be mocked out.
 */

"use strict";

var m = require('module'),
    registeredMocks = {},
    registeredSubstitutes = {},
    registeredAllowables = {},
    originalLoader = null,
    originalCache = null,
    defaultOptions = {
        useCleanCache: false,
        warnOnReplace: true,
        warnOnUnregistered: true
    },
    options = {};

/*
 * Merge the supplied options in with a new copy of the default options to get
 * the effective options, and return those.
 */
function getEffectiveOptions(opts) {
    var options = {};

    Object.keys(defaultOptions).forEach(function (key) {
        options[key] = defaultOptions[key];
    });
    if (opts) {
        Object.keys(opts).forEach(function (key) {
            options[key] = opts[key];
        });
    }
    return options;
}

/*
 * The (private) loader replacement that is used when hooking is enabled. It
 * does the work of returning a mock or substitute when configured, reporting
 * non-allowed modules, and invoking the original loader when appropriate.
 * The signature of this function *must* match that of Node's Module._load,
 * since it will replace that when mockery is enabled.
 */
function hookedLoader(request, parent, isMain) {
    var subst, allow, file;

    if (!originalLoader) {
```

```
79          throw new Error("Loader has not been hooked");
80      }
81
82      if (registeredMocks.hasOwnProperty(request)) {
83          return registeredMocks[request];
84      }
85
86      if (registeredSubstitutes.hasOwnProperty(request)) {
87          subst = registeredSubstitutes[request];
88          if (!subst.module && subst.name) {
89              subst.module = originalLoader(subst.name, parent, isMain);
90          }
91          if (!subst.module) {
92              throw new Error("Misconfigured substitute for '" + request + "'");
93          }
94          return subst.module;
95      }
96
97      if (registeredAllowables.hasOwnProperty(request)) {
98          allow = registeredAllowables[request];
99          if (allow.unhook) {
100             file = m._resolveFilename(request, parent);
101             if ((file.indexOf('/') !== -1 || file.indexOf('\\') !== -1) && allow.paths.indexOf(fil
102                 allow.paths.push(file);
103             }
104         }
105     } else {
106         if (options.warnOnUnregistered) {
107             console.warn("WARNING: loading non-allowed module: " + request);
108         }
109     }
110
111     return originalLoader(request, parent, isMain);
112 }
113
114 /*
115  * Enables mockery by hooking subsequent 'require' invocations. Note that *all*
116  * 'require' invocations will be hooked until 'disable' is called. Calling this
117  * function more than once will have no ill effects.
118  */
119 function enable(opts) {
120     if (originalLoader) {
121         // Already hooked
122         return;
123     }
124
125     options = getEffectiveOptions(opts);
126
127     if (options.useCleanCache) {
```

```
        originalCache = m._cache;
        m._cache = {};
        repopulateNative();
    }

    originalLoader = m._load;
    m._load = hookedLoader;
}

/*
 * Disables mockery by unhooking from the Node loader. No subsequent 'require'
 * invocations will be seen by mockery. Calling this function more than once
 * will have no ill effects.
 */
function disable() {
    if (!originalLoader) {
        // Not hooked
        return;
    }

    if (options.useCleanCache) {
        // Previously this just set m._cache to originalCache. This would make
        // node re-require native addons that were required while mockery was
        // enabled, which breaks it in node@>=0.12. Instead populate
        // originalCache with any native addons that were first required since
        // mockery was enabled.
        Object.keys(m._cache).forEach(function(k){
            if (k.indexOf('\.node') > -1 && !originalCache[k]) {
                originalCache[k] = m._cache[k];
            }
        });
        removeParentReferences();
        m._cache = originalCache;
        originalCache = null;
    }

    m._load = originalLoader;
    originalLoader = null;
}

 /*
 * If the clean cache option is in effect, reset the module cache to an empty
 * state. Calling this function when the clean cache option is not in effect
 * will have no ill effects, but will do nothing.
 */
function resetCache() {
    if (options.useCleanCache && originalCache) {
        removeParentReferences();
        m._cache = {};
```

```
        repopulateNative();
    }
}

/*
 * Starting in node 0.12 node won't reload native modules
 * The reason is that native modules can register themselves to be loaded automatically
 * This will re-populate the cache with the native modules that have not been mocked
 */
function repopulateNative() {
  Object.keys(originalCache).forEach(function(k) {
      if (k.indexOf('\.node') > -1 && !m._cache[k]) {
          m._cache[k] = originalCache[k];
      }
  });
}

/*
 * Enable or disable warnings to the console when previously registered mocks
 * and subsitutes are replaced.
 */
function warnOnReplace(enable) {
    options.warnOnReplace = enable;
}

/*
 * Enable or disable warnings to the console when modules are loaded that have
 * not been registered as a mock, a substitute, or allowed.
 */
function warnOnUnregistered(enable) {
    options.warnOnUnregistered = enable;
}

/*
 * Register a mock object for the specified module. While mockery is enabled,
 * any subsequent 'require' for this module will return the mock object. The
 * mock need not mock out all original exports, but no fallback is provided
 * for anything not mocked and subsequently invoked.
 */
function registerMock(mod, mock) {
    if (options.warnOnReplace && registeredMocks.hasOwnProperty(mod)) {
        console.warn("WARNING: Replacing existing mock for module: " + mod);
    }
    registeredMocks[mod] = mock;
}

/*
 * Deregister a mock object for the specified module. A subsequent 'require' for
 * that module will revert to the previous behaviour (which, by default, means
```

```
 * falling back to the original 'require' behaviour).
 */
function deregisterMock(mod) {
    if (registeredMocks.hasOwnProperty(mod)) {
        delete registeredMocks[mod];
    }
}

/*
 * Register a substitute module for the specified module. While mockery is
 * enabled, any subsequent 'require' for this module will be effectively
 * replaced by a 'require' for the substitute module. This is useful when
 * a mock implementation is itself implemented as a module.
 */
function registerSubstitute(mod, subst) {
    if (options.warnOnReplace && registeredSubstitutes.hasOwnProperty(mod)) {
        console.warn("WARNING: Replacing existing substitute for module: " + mod);
    }
    registeredSubstitutes[mod] = {
        name: subst
    };
}

/*
 * Deregister a substitute module for the specified module. A subsequent
 * 'require' for that module will revert to the previous behaviour (which, by
 * default, means falling back to the original 'require' behaviour).
 */
function deregisterSubstitute(mod) {
    if (registeredSubstitutes.hasOwnProperty(mod)) {
        delete registeredSubstitutes[mod];
    }
}

/*
 * Register a module as 'allowed', meaning that, even if a mock or substitute
 * for it has not been registered, mockery will not complain when it is loaded
 * via 'require'. This encourages the user to consciously declare the modules
 * that will be loaded and used in the original form, thus avoiding warnings.
 *
 * If 'unhook' is true, the module will be removed from the module cache when
 * it is deregistered.
 */
function registerAllowable(mod, unhook) {
    registeredAllowables[mod] = {
        unhook: !!unhook,
        paths: []
    };
}
```

```
275
276 /*
277  * Register an array of modules as 'allowed'. This is a convenience function
278  * that performs the same function as 'registerAllowable' but for an array of
279  * modules rather than a single module.
280  */
281 function registerAllowables(mods, unhook) {
282     mods.forEach(function (mod) {
283         registerAllowable(mod, unhook);
284     });
285 }
286
287 /*
288  * Deregister a module as 'allowed'. A subsequent 'require' for that module
289  * will generate a warning that the module is not allowed, unless or until a
290  * mock or substitute is registered for that module.
291  */
292 function deregisterAllowable(mod) {
293     if (registeredAllowables.hasOwnProperty(mod)) {
294         var allow = registeredAllowables[mod];
295         if (allow.unhook) {
296             allow.paths.forEach(function (p) {
297                 delete m._cache[p];
298             });
299         }
300         delete registeredAllowables[mod];
301     }
302 }
303
304 /*
305  * Deregister an array of modules as 'allowed'. This is a convenience function
306  * that performs the same function as 'deregisterAllowable' but for an array of
307  * modules rather than a single module.
308  */
309 function deregisterAllowables(mods) {
310     mods.forEach(function (mod) {
311         deregisterAllowable(mod);
312     });
313 }
314
315 /*
316  * Deregister all mocks, substitutes, and allowed modules, resetting the state
317  * to a clean slate. This does not affect the enabled / disabled state of
318  * mockery, though.
319  */
320 function deregisterAll() {
321     Object.keys(registeredAllowables).forEach(function (mod) {
322         var allow = registeredAllowables[mod];
323         if (allow.unhook) {
```

```
            allow.paths.forEach(function (p) {
                delete m._cache[p];
            });
        }
    });

    registeredMocks = {};
    registeredSubstitutes = {};
    registeredAllowables = {};
}

/**
 * Remove references to modules in the mockery cache from
 * their parents' children.
 */
function removeParentReferences() {
    Object.keys(m._cache).forEach(function(k){
        if (k.indexOf('\.node') === -1) {
            // don't touch native modules, because they're special
            var mod = m._cache[k];
            var idx = mod.parent && mod.parent.children && mod.parent.children.indexOf(mod);
            if (idx > -1) {
                mod.parent.children.splice(idx, 1);
            }
        }
    });
}

// Exported functions
exports.enable = enable;
exports.disable = disable;
exports.resetCache = resetCache;
exports.warnOnReplace = warnOnReplace;
exports.warnOnUnregistered = warnOnUnregistered;
exports.registerMock = registerMock;
exports.registerSubstitute = registerSubstitute;
exports.registerAllowable = registerAllowable;
exports.registerAllowables = registerAllowables;
exports.deregisterMock = deregisterMock;
exports.deregisterSubstitute = deregisterSubstitute;
exports.deregisterAllowable = deregisterAllowable;
exports.deregisterAllowables = deregisterAllowables;
exports.deregisterAll = deregisterAll;
```