Build-Time Javascript Code Analysis

Alexandru Objelean
Moldova State University
alex.objelean@gmail.com

***Abstract* — Javascript can be tough to maintain. The bigger is your project, the harder it will be to ensure that everything is ok. Luckily, there are tools to help you with that.**

One of the recently launched tools is JSHint, a online JavaScript checking tool. The tool is very similar to JSLint, but is designed to be more customizable and community-oriented. JSHint can help you to detect errors and potential problems in JavaScript code and to enforce your team's coding conventions. It is very flexible so you can easily adjust it to your particular coding guidelines and the environment you expect your code to execute in.

The purpose of this article is to show you how to use jsHint as a build-time solution using wro4j maven plugin.

I. INTRODUCTION

Installing an automatic build process for your projects is very common today and best practice. In the java world Maven[3] is a very popular build tool and has proven its matureness over the years. Using maven[3] to build your client-side project has the following advantages:

- Find potential problems in your code. Bugs might be identified and breaking code conventions can be detected early.
- Run the build automatically on a regular base
- Easily handle refactorings and other small changes
- Ability to do continuous integration
- Increase confidence in code quality for each build

II. WHAT IS JSHINT?

JSHint[1] is a JavaScript program that looks for problems in JavaScript programs. It is a code quality tool.

When C was a young programming language, there were several common programming errors that were not caught by the primitive compilers, so an accessory program called lint was developed that would scan a source file, looking for problems.

As the language matured, the definition of the language was strengthened to eliminate some insecurity, and compilers got better at issuing warnings. lint is no longer needed.

JavaScript is a young-for-its-age language. It was originally intended to do small tasks in web pages, tasks for which Java was too heavy and clumsy. But JavaScript is a very capable language, and it is now being used in larger projects. Many of the features that were intended to make the language easy to use are troublesome when projects become complicated. A lint for JavaScript is needed: JSHint, a JavaScript syntax checker and validator.

JSHint takes a JavaScript source and scans it. If it finds a problem, it returns a message describing the problem and an approximate location within the source. The problem is not necessarily a syntax error, although it often is. JSLint looks at some style conventions as well as structural problems. It does not prove that your program is correct. It just provides another set of eyes to help spot problems.

JSHint defines a professional subset of JavaScript, a stricter language than that defined by Third Edition of the ECMAScript Programming Language Standard. The subset is related to recommendations found in Code Conventions for the JavaScript Programming Language.

JavaScript is a sloppy language, but inside it there is an elegant, better language. JSHint helps you to program in that better language and to avoid most of the slop. JSHint will reject programs that browsers will accept because JSHint is concerned with the quality of your code and browsers are not. You should accept all of JSHint's advice.

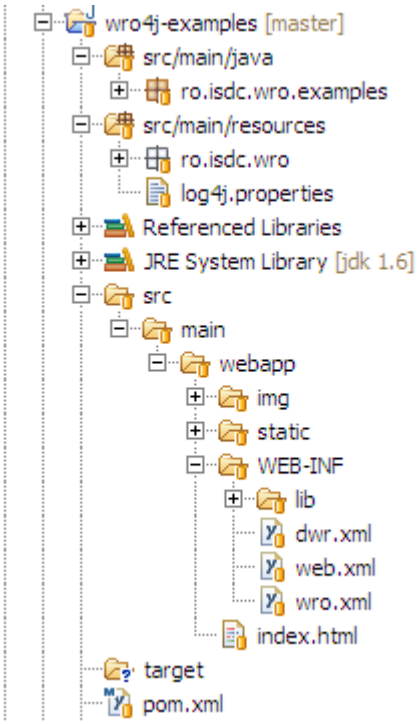
III. USING WRO4J MAVEN PLUGIN

wro4j[2] mavenp[3] plugin provides a new goal called jshint which can help you to start auditing and enforcing JS code through a mechanism like JSHint[1].

In order to use it you have to follow several simple steps.

Project layout:

By default, wro4j maven plugin relies on a typical maven[3] project layout.



This structure can be different, depending on your project. The implicit wro4j maven plugin settings will assume you are using this structure. By default it will search for a file called wro.xml at the following location src/main/webapp/WEB-INF/wro.xml. The location of this file is configurable. You'll find more about all available configuration below. The purpose of wro.xml file is to describe the way static resources are grouped and the order in which these should be processed. By default the plugin will process each resource one by one.

Configure pom.xml

Add wro4j maven plugin to the list of plugins in your web project:

```
<plugins>
  <plugin>
    <groupid>ro.isdc.wro4j</groupid>
    <artifactid>wro4j-maven-
plugin</artifactid>
    <version>${wro4j.version}</version>
    <executions>
      <execution>
        <phase>compile</phase>
        <goals>
          <goal>jshint</goal>
        </goals>
      </execution>
    </executions>
    <configuration>
      <options>devel,evil,noarg</options>
    </configuration>
  </plugin>
</plugins>
```

${wro4j.version} - is the latest wro4j version. The goal which instructs wro4j to run jshint tool is called jshint. Besides this goal, wro4j maven plugin provides also run goal which performs the compression of the resources (both js & css), but this is not in the scope of this post.
This plugin allows you configure the options used by jshint. Specifying these options is optional. For the complete list of available options, visit the JsHint project homepage. As you can see in the above example, options configuration contains a comma separated values.

Detailed plugin configurations:

- **options** - comma separated values used to instruct jsHint how to analyze the js code.
- **failNever** - boolean flag. When true - the project build will succeed even if there are errors reported by jsHint. By default this value is false.
- **ignoreMissingResources** - if false, the build will fail if at least one resource is missing or cannot be accessed, otherwise only a warning will be displayed.
- **targetGroups** - (optional) a comma separated list of groups you want to build. If you do not specify this parameter, a file for each defined group will be generated.
- **wroManagerFactory** - Optional attribute, used to specify a custom implementation of MavenContextAwareManagerFactory interface. When this tag is not specified, a default implementation is used. This attribute is useful if you want to configure other processors than default one.
- **wroFile** - the location of wro.xml file which defines how resources are grouped. By default its value is this: ${basedir}/src/main/webapp/WEB-INF/wro.xml . If you have a different project structure or a different location of wro.xml, then you should change the value of this parameter.
- **contextFolder** - defines the location of web application context, useful for locating resources relative to servletContext. By default its value is: ${basedir}/src/main/webapp/

These parameters gives you enough control to customize the wro4j maven plugin to work with any project structure. When running the plugin and any problems are encountered, you'll see a detailed list of the errors which indicate the file containing the problems, line number where the problem is and the detailed message describing the problem. Here is an example:

```
[error] 1 errors found while processing
resource:
classpath:ro/isdc/wro/maven/plugin/js/u
ndef.js Errors are:
[ro.isdc.wro.extensions.processor.algor
ithm.jshint.JsError@19d75ee[
  line=4
  character=4
  reason='jQuery' is not defined.
  evidence=})(jQuery);
]]
```

These informations should be enough to help you to fix any problems you may have.

IV. CONCLUSIONS

Using wro4j maven[3] plugin with **jsHint** goal can help you to ensure coding standards on your front-end resources when using a project which respect a maven project structure.

REFERENCES

[1] JSHint Home Page: http://jshint.com/
[2] Web Resource Optimization for java framework: http://code.google.com/p/wro4j/
[3] "Overview of ApacheMaven 2 Effective Implementation." Brett Porter, Packt Publishing