Join GitHub today

GitHub is home to over 28 million developers working together to host and review code, manage projects, and build software together.

Sign up

Dismiss

Bck2Brwsr VM to execute Java bytecode in a pluginless browser

2,021 commits | **13** branches | **24** releases | **3** contributors

Branch: **master** | **New pull request** | **Find file**

Clone or download

jtulach mvn install -DskipTests shalln't yield an error
Latest commit 17a25b2 May 31, 2018

benchmarks	[maven-release-plugin] prepare for next development iteration	May 28, 2018
docs	Getting started with Maven, Gradle and console	Jun 1, 2018
javaquery	[maven-release-plugin] prepare for next development iteration	May 28, 2018
ko	[maven-release-plugin] prepare for next development iteration	May 28, 2018
launcher	[maven-release-plugin] prepare for next development iteration	May 28, 2018
rt	mvn install -DskipTests shalln't yield an error	Jun 1, 2018
.gitignore	Ignore gradle dirs	May 1, 2018
.hgignore	Internet Explorer somehow caches values of person.json and people.jso…	Dec 14, 2014
.travis.yml	Run the test only on linux for now	Nov 13, 2017
COPYING	Expanding copyright to 2018	Apr 7, 2018
README.md	Getting started with Maven, Gradle and console	Jun 1, 2018
pom.xml	[maven-release-plugin] prepare for next development iteration	May 28, 2018

README.md

build passing build passing build passing build passing build passing

Bring Java Back to Browser!

Bck2Brwsr VM is a Java virtual machine which is capable to take bytecode and transpile it (either ahead-of-time - e.g. during compilation on desktop - or just-in-time, e.g. in a browser) into appropriate JavaScript code which does the same thing. As a result one can write in Java, compile with JavaC, run it with Bck2Brwsr in any modern browser.

Getting Started

If you have a **Maven** or **Gradle** project it is as easy as adding one plugin to get your Java application into browser. Imagine you have a simple hello world application in `src/main/java` structure of your project:

```
public class Hello {
    public static void main(String... args) {
        System.err.println("Hello from Java in JS!");
    }
}
```

then it is just about adding following plugin into your `pom.xml`:

```
    <build>
        <plugins>
            <plugin>
                <groupId>org.apidesign.bck2brwsr</groupId>
                <artifactId>bck2brwsr-maven-plugin</artifactId>
                <version>0.23</version>
            </plugin>
        </plugins>
    </build>
```

and executing `mvn clean install bck2brwsr:aot bck2brwsr:show` - more info in a dedicated page.

It is similarly easy with **Gradle**. Just by adding (the same) single plugin and configuring your script to use it:

```
buildscript {
    repositories {
        mavenCentral()
    }
    dependencies {
        classpath "org.apidesign.bck2brwsr:bck2brwsr-maven-plugin:0.23"
    }
}

repositories {
    mavenCentral()
}

apply plugin: 'bck2brwsr'
```

you'll be able to invoke `./gradlew bck2brwsrShow` and see your Java application greetings from the browser. An in-depth tutorial is available as well.

How do I deploy?

Just copy the generated files to any web hosting service! No application server needed, no code needs to be executed on the server. The whole application is *just* a set of pages that can be copied anywhere. In case of **Gradle** it consists of:

```
$ find build/web/
build/web/
build/web/index.html
build/web/lib
build/web/lib/emul-0.23-rt.js
build/web/main.js
build/web/bck2brwsr.js
```

The structure of pages generated by **Maven** is similar with the primary HTML file being `target/index.html`.

Talking to Your Java Code

Once your application is running in the browser, you can interact with it from a console. Btw. output of `System.out` and `System.err` is primarily printed into the console. Open the console and try:

```
var System = vm.loadClass("java.lang.System")
System.exit(0)
```

and voilá! your browser is closed. Obviously the above snippet is more useful for other methods in your application than `System.exit`. You can use it to locate any `public class` and invoke any of its `public static` methods.

Launching

Browsers download scripts asynchronously, as such the `vm.loadClass` may not be immediatelly ready, but it may get loaded with a delay. To accomodate such restriction one can use a callback style of the above:

```
vm.loadClass("java.lang.System", function(System) {
  System.exit(0)
});
```

Usage of callback is recommended when executing the first Java method (as that usually means loading of new scripts) and is exactly what the default content of `index.html` does:

```
<script src='bck2brwsr.js'></script>
<script>
var vm = bck2brwsr('main.js');
vm.loadClass('Hello', function(mainClass) {
  mainClass.invoke('main');
});
</script>
```

The first line loads the Bck2Brwsr virtual machine. Then one fills it with transpiled code of one's application `var vm = bck2brwsr('main.js');` together with required libraries. At the end one invokes `main` method of the `Hello` class using the callback syntax.

More?

More info at project home page. Questions to bck2brwsr group.