IdPJava1.8

Scripted Attribute Definitions, IdP V2, and Java version 1.8

In Java version 1.8 a new scripting engine (codename 'Nashorn') was introduced. This has some incompatibilities with the 'Rhino' scripting engine used by earlier versions. The new engine supports a language with minor syntactic differences and one major semantic difference, all of which make it impossible to 'just upgrade' the JVM running an IdP V2 to Java version 1.8.

This document describes two techniques to allow you to move forward.

1) Changing your Configuration to work JDKV8's 'Nashorn' engine.

Before you start

As always, make a backup of your configuration. The process of conversion will involve a certain amount of test and retest and so you should allow for downtime during which you can do this work.

Ideally, develop your changed scripts and configuration in a separate sandbox.

Syntactic changes.

Description of these differences is out of scope for this article. Refer to the copious documentation available (for instance here). Suffice to say that the mechanisms by which you create Java objects have changed and whereas previously you would have:

Rhino syntax for Java classes

```
importPackage(Packages.edu.internet2.middleware.shibboleth.common.attribute.provider);
fullname = new BasicAttribute("fullname");
```

in 1.8 you will say either

Nashorn syntax for Java classes

```
var BasicAttribute = Java.type("edu.internet2.middleware.shibboleth.common.attribute.provider.BasicAttribute");
fullname = new BasicAttribute("fullname");
```

or

Nashorn work around for Java classes

```
// load compatibility script
load("nashorn:mozilla_compat.js");
importPackage(Packages.edu.internet2.middleware.shibboleth.common.attribute.provider);
```

Although, as we shall see, this specific case (the creation of the output attribute) is not required

Semantic changes

As far as the Shibboleth IdP V2 is concerned, the most important difference in JavaScript between Java version 1.7 and 1.8 is how information is passed between the scripting language environment and the IdP. In earlier versions a script can create an attribute (as above) and it will be available to the IdP. With Nashorn this is no longer possible and only attributes which exist when the script is started are available. In practice this means that

1. The output attribute needs to have been created by a Static Data Connector which is declared as a dependency of the scripted attribute.
2. The attribute creation is removed from the script and replaced by a line of code to remove the values added to the dummy input attribute.

It should be emphasized that the technique of inheriting existing attributes in a script and just changing the contents is supported and indeed exploited by some of the examples.

A worked example

Starting with the attribute definition:

Example Attribute definition for Java 1.7

```
<AttributeResolver xmlns="urn:mace:shibboleth:2.0:resolver" xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance"
                   xmlns:resolver="urn:mace:shibboleth:2.0:resolver" xmlns:ad="urn:mace:shibboleth:2.0:resolver:
ad"
                   xmlns:dc="urn:mace:shibboleth:2.0:resolver:dc"
                   xsi:schemaLocation="urn:mace:shibboleth:2.0:resolver classpath:/schema/shibboleth-2.0-
attribute-resolver.xsd
                                       urn:mace:shibboleth:2.0:resolver:dc classpath:/schema/shibboleth-2.0-
attribute-resolver-dc.xsd
                                       urn:mace:shibboleth:2.0:resolver:ad classpath:/schema/shibboleth-2.0-
attribute-resolver-ad.xsd">
    <resolver:AttributeDefinition xmlns="urn:mace:shibboleth:2.0:resolver:ad" xsi:type="ad:Script" id="scripted"
>
        <resolver:Dependency ref="static"/>
        <Script><![CDATA[
                   importPackage(Packages.edu.internet2.middleware.shibboleth.common.attribute.provider);
                   scripted = new BasicAttribute("scripted");
                   scripted.getValues().addAll(affiliation.getValues())]]>
        </Script>
    </resolver:AttributeDefinition>

    <resolver:DataConnector xmlns="urn:mace:shibboleth:2.0:resolver:dc" xsi:type="dc:Static" id="static">
        <Attribute id="affiliation">
            <Value>student</Value>
            <Value>student-worker-parttime</Value>
            <Value>parent</Value>
        </Attribute>
    </resolver:DataConnector>
</AttributeResolver>
```

In order to convert to Java version 1.8 we

- Add a new data connector with the dummy attribute.
- Change the script to remove the creation of the attribute and replace it with the removal of the dummy value.
- Add the new data connector as a dependency.

The final definition looks like this:

Script converted to Java 1.8

```
<AttributeResolver xmlns="urn:mace:shibboleth:2.0:resolver" xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance"
                   xmlns:resolver="urn:mace:shibboleth:2.0:resolver" xmlns:ad="urn:mace:shibboleth:2.0:resolver:
ad"
                   xmlns:dc="urn:mace:shibboleth:2.0:resolver:dc"
                   xsi:schemaLocation="urn:mace:shibboleth:2.0:resolver classpath:/schema/shibboleth-2.0-
attribute-resolver.xsd
                                       urn:mace:shibboleth:2.0:resolver:dc classpath:/schema/shibboleth-2.0-
attribute-resolver-dc.xsd
                                       urn:mace:shibboleth:2.0:resolver:ad classpath:/schema/shibboleth-2.0-
attribute-resolver-ad.xsd">
    <resolver:AttributeDefinition xmlns="urn:mace:shibboleth:2.0:resolver:ad" xsi:type="ad:Script" id="scripted"
>
        <resolver:Dependency ref="static"/>
        <resolver:Dependency ref="forScript"/>
        <Script><![CDATA[
              scripted.getValues().clear(); // remove the dummy value
              scripted.getValues().addAll(affiliation.getValues());]]>
        </Script>
    </resolver:AttributeDefinition>

   <resolver:DataConnector xmlns="urn:mace:shibboleth:2.0:resolver:dc" xsi:type="dc:Static" id="static">
        <Attribute id="affiliation">
            <Value>student</Value>
            <Value>student-worker-parttime</Value>
            <Value>parent</Value>
        </Attribute>
    </resolver:DataConnector>

    <resolver:DataConnector xmlns="urn:mace:shibboleth:2.0:resolver:dc" xsi:type="dc:Static" id="forScript">
        <Attribute id="scripted">
            <Value>dummy</Value>
        </Attribute>
    </resolver:DataConnector>

</AttributeResolver>
```

It should be noted that this particular definition works under both scripting engines since there is now no (Java) object creation.

2) Running the Rhino engine under Java 1.8

As an interim step, it is possible to make the Rhino scripting language available to an IdP running under 1.8. This is not a recommended long term strategy but it can be useful as an interim step, particularly since the two engines coexist and you can gradually move each script over from one language to another.

To install the Rhino engine into your IdP:

- Download the package from http://shibboleth.net/downloads/identity-provider/extensions/rhino-js-jdk8
- Copy the contents of the `lib` directory in the package into the `lib` directory of your distribution (there are two jars).
- Rebuild the war file.
    - Do not forget to ensure that any local changes to the jsp and web.xml files are updated.
- Redeploy the war file.
- Restart the IdP.

At this stage a new scripting language “rhino-nonjdk” is available. This is the current (1.7R4) version of the Mozilla “rhino” engine. .

To make a scripted attribute be interpreted by this engine you need to change the language:

Using the rhino engine

```
<resolver:AttributeDefinition xmlns="urn:mace:shibboleth:2.0:resolver:ad" xsi:type="ad:Script" id="scripted"
language="rhino-nonjdk">
<!-- The rest of the definition can be left unchanged -->
```

At this stage you can upgrade to Java version 1.8.