Extending and Customizing {brandname} 10.0

Table of Contents

Chapter 1. Extending {brandname}

{brandname} can be extended to provide the ability for an end user to add additional configurations, operations and components outside of the scope of the ones normally provided by {brandname}.

1.1. Custom Commands

{brandname} makes use of a command/visitor pattern to implement the various top-level methods you see on the public-facing API.

While the core commands - and their corresponding visitors - are hard-coded as a part of {brandname}'s core module, module authors can extend and enhance {brandname} by creating new custom commands.

As a module author (such as infinispan-query, etc.) you can define your own commands.

You do so by:

1. Create a `META-INF/services/org.infinispan.commands.module.ModuleCommandExtensions` file and ensure this is packaged in your jar.
2. Implementing `ModuleCommandFactory`, `ModuleCommandInitializer` and `ModuleCommandExtensions`
3. Specifying the fully-qualified class name of the `ModuleCommandExtensions` implementation in `META-INF/services/org.infinispan.commands.module.ModuleCommandExtensions`.
4. Implement your custom commands and visitors for these commands

1.1.1. An Example

Here is an example of an `META-INF/services/org.infinispan.commands.module.ModuleCommandExtensions` file, configured accordingly:

org.infinispan.commands.module.ModuleCommandExtensions

```
org.infinispan.query.QueryModuleCommandExtensions
```

For a full, working example of a sample module that makes use of custom commands and visitors, check out {brandname} Sample Module .

1.1.2. Preassigned Custom Command Id Ranges

This is the list of `Command` identifiers that are used by {brandname} based modules or frameworks. {brandname} users should avoid using ids within these ranges. (RANGES to be finalised yet!) Being this a single byte, ranges can't be too large.

{brandname} Query:	100 - 119
Hibernate Search:	120 - 139

Hot Rod Server:	140 - 141

1.2. Extending the configuration builders and parsers

If your custom module requires configuration, it is possible to enhance {brandname}'s configuration builders and parsers. Look at the custom module tests for a detail example on how to implement this.

Chapter 2. Custom Interceptors

It is possible to add custom interceptors to {brandname}, both declaratively and programatically. Custom interceptors are a way of extending {brandname} by being able to influence or respond to any modifications to cache. Example of such modifications are: elements are added/removed/updated or transactions are committed. For a detailed list refer to CommandInterceptor API.

2.1. Adding custom interceptors declaratively

Custom interceptors can be added on a per named cache basis. This is because each named cache have its own interceptor stack. Following xml snippet depicts the ways in which a custom interceptor can be added.

```
<local-cache name="cacheWithCustomInterceptors">
      <!--
      Define custom interceptors.  All custom interceptors need to extend
org.jboss.cache.interceptors.base.CommandInterceptor
      -->
      <custom-interceptors>
         <interceptor position="FIRST" class="com.mycompany.CustomInterceptor1">
               <property name="attributeOne">value1</property>
               <property name="attributeTwo">value2</property>
         </interceptor>
         <interceptor position="LAST" class="com.mycompany.CustomInterceptor2"/>
         <interceptor index="3" class="com.mycompany.CustomInterceptor1"/>
         <interceptor before="org.infinispanpan.interceptors.CallInterceptor" class=
"com.mycompany.CustomInterceptor2"/>
         <interceptor after="org.infinispanpan.interceptors.CallInterceptor" class=
"com.mycompany.CustomInterceptor1"/>
      </custom-interceptors>
</local-cache>
```

2.1.1. Adding custom interceptors programatically

In order to do that one needs to obtain a reference to the AdvancedCache . This can be done as follows:

```
CacheManager cm = getCacheManager();//magic
Cache aCache = cm.getCache("aName");
AdvancedCache advCache = aCache.getAdvancedCache();
```

Then one of the *addInterceptor()* methods should be used to add the actual interceptor. For further documentation refer to AdvancedCache javadoc.

2.2. Custom interceptor design

When writing a custom interceptor, you need to abide by the following rules.

- Custom interceptors must extend BaseCustomInterceptor
- Custom interceptors must declare a public, empty constructor to enable construction.
- Custom interceptors will have setters for any property defined through property tags used in the XML configuration.