BEANSHELL

BeanShell is a Java-like dynamically typed language that allows to execute scripts in a running Java environment.

- We can use it to enhance the functionality of your application or to test and debug snippets of code without the lengthy compile and re-deploy cycles.

- Hybris platform incorporates BeanShell since its release 4

Wht we do in beanshell ?

create some Hybris object, persist it and feed it to the service method

API Overview

From the user perspective, the API is very simple. It is based on four main interfaces:

- ScriptingLanguagesService provides methods for obtaining a ScriptExecutable object.
- ScriptExecutable represents an executable object, which provides methods for executing it.
- ScriptContent represents an object that wraps script content in a specific language type.
- ScriptExecutionResult represents the script execution result.

```
======================
import de.hybris.platform.product.impl.DefaultProductService;
import de.hybris.platform.core.model.order.OrderModel;
import de.hybris.platform.servicelayer.search.FlexibleSearchQuery;
import de.hybris.platform.servicelayer.search.SearchResult;

modelService = spring.getBean("modelService");
flexibleSearchService = spring.getBean("flexibleSearchService");
defaultAbstractOrderService = spring.getBean("defaultAbstractOrderService");



 String[] orderList={"00002001"};

 for(String orderNum:orderList)
       {

String fetchquery="select {o.pk} from {Order as o} where {o.code}='"+orderNum+"'";


FlexibleSearchQuery query = new FlexibleSearchQuery(fetchquery);
SearchResult results = flexibleSearchService.search(query);

for (OrderModel orderModel : results.getResult()) {

  orderModel.setDescription("Cent");
  modelService.save(orderModel);


  print(orderModel.getCode()+" is : "+orderModel.getStatus().getCode());
}
```

```
}
```

==========

```
import de.hybris.platform.product.impl.DefaultProductService;
import de.hybris.platform.core.model.order.OrderModel;
import de.hybris.platform.servicelayer.search.FlexibleSearchQuery;
import de.hybris.platform.servicelayer.search.SearchResult;

modelService = spring.getBean("modelService");
flexibleSearchService = spring.getBean("flexibleSearchService");
defaultCheckOrderService = spring.getBean("defaultCheckOrderService");



 String[] orderList={"00002001"};

 for(String orderNum:orderList)
       {

String fetchquery="select {o.pk} from {Order as o} where {o.code}="'+orderNum+'"";


FlexibleSearchQuery query = new FlexibleSearchQuery(fetchquery);
SearchResult results = flexibleSearchService.search(query);

for (OrderModel orderModel : results.getResult()) {

  orderModel.setDescription("Cent");
  modelService.save(orderModel);

  print(defaultCheckOrderService.check(orderModel));

  print(orderModel.getCode()+" is : "+orderModel.getStatus().getCode());
}

}
```

==========

```
import de.hybris.platform.product.impl.DefaultProductService;
import de.hybris.platform.core.model.order.OrderModel;
import de.hybris.platform.servicelayer.search.FlexibleSearchQuery;
import de.hybris.platform.servicelayer.search.SearchResult;
import org.apache.log4j.Logger;

modelService = spring.getBean("modelService");
flexibleSearchService = spring.getBean("flexibleSearchService");

Logger LOG=Logger.getLogger("myBeanshell");

void callMethod(OrderModel order){

  LOG.info("hellloooooo");
```

```
 print("lo ji");

}

 String[] orderList={"00002001"};

 for(String orderNum:orderList)
       {

String fetchquery="select {o.pk} from {Order as o} where {o.code}="'+orderNum+'"";


FlexibleSearchQuery query = new FlexibleSearchQuery(fetchquery);
SearchResult results = flexibleSearchService.search(query);


for (OrderModel orderModel : results.getResult()) {

   callMethod(orderModel);
  orderModel.setDescription("Cent");
  modelService.save(orderModel);

  print(orderModel.getCode()+" is : "+orderModel.getStatus().getCode());
}

}

print("Hi");



======================
```

Executing Scripts from the File System

Assuming that our sample script is located on the disk, with the path /Users/zeus/scripts/setMimesForMedias.groovy,you can execute it as follows:

```
import org.springframework.core.io.Resource;
import org.springframework.core.io.FileSystemResource;

final Resource resource
 = new FileSystemResource("/Users/zeus/scripts/setMimesForMedias.groovy");

// Let's assume we have scriptingLanguagesService injected by the Spring
final ScriptContent scriptContent
       = new ResourceScriptContent(resource);
final ScriptExecutable executable
       = scriptingLanguagesService.getExecutableByContent(scriptContent);

// now we can execute script
final ScriptExecutionResult result = executable.execute();

// to obtain result of execution
System.out.println(result.getScriptResult());
```

==========

Executing Scripts by Using Classpath
Or, if you have the same script but in the classpath in the folder scripts:

```
import org.springframework.core.io.Resource;
import org.springframework.core.io.ClassPathResource;

final Resource resource
      = new ClassPathResource("scripts/setMimesForMedias.groovy");

// Let's assume we have scriptingLanguagesService injected by the Spring
final ScriptContent scriptContent
      = new ResourceScriptContent(resource);
final ScriptExecutable executable
      = scriptingLanguagesService.getExecutableByContent(scriptContent);

// now we can execute script
final ScriptExecutionResult result = executable.execute();

// to obtain result of execution
System.out.println(result.getScriptResult());
```

==========

Executing Scripts Stored Remotely
You can also store the script content remotely, for instance as a gist at github.comInformation published on non-SAP site under the URLhttps://gist.githubusercontent.com/zeus/testMimesForMedias.groovy. In this way, it is easy to get hold of and execute:

```
import org.springframework.core.io.Resource;
import org.springframework.core.io.UrlResource;

final Resource resource
       = new UrlResource(
   "https://gist.githubusercontent.com/zeus/setMimesForMedias.groovy");

// Let's assume we have scriptingLanguagesService injected by the Spring
final ScriptContent scriptContent
       = new ResourceScriptContent(resource);
final ScriptExecutable executable
       = scriptingLanguagesService.getExecutableByContent(scriptContent);

// now we can execute script
final ScriptExecutionResult result
       = executable.execute();

// to obtain result of execution
System.out.println(result.getScriptResult());
```