SailPoint

Developer Days2023

</title>

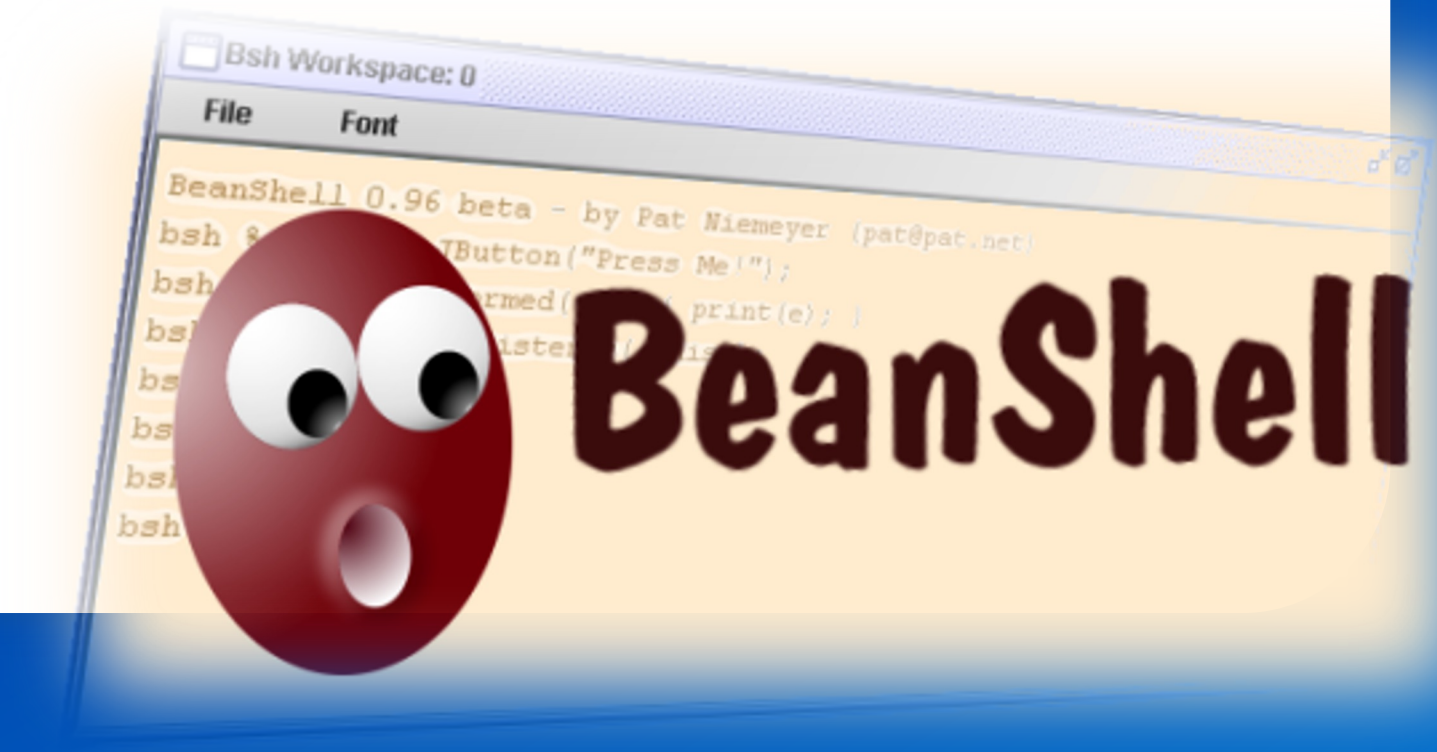
IdentityIQ & BeanShell

</title>

Speaker: Christian Cairney
title: Principal Solution Architect
company: SailPoint

Agenda

- BeanShell Overview
- Best Practice
- Testing



```
if localVarHTTPResponse.StatusCode >= 300 {
        newErr := &GenericOpenAPIError{
            body:  localVarBody,
            error: localVarHTTPResponse.Status,
        }
```

Overview

```
    if localVarHTTPResponse.StatusCode == 400 {
            var v ErrorResponseDto
            err = a.client.decode(&v, localVarBody, localVarHTT
            if err != nil {
                newErr.error = err.Error()
                return localVarReturnValue, localVarHTTPResp
            }
```

BeanShell Overview

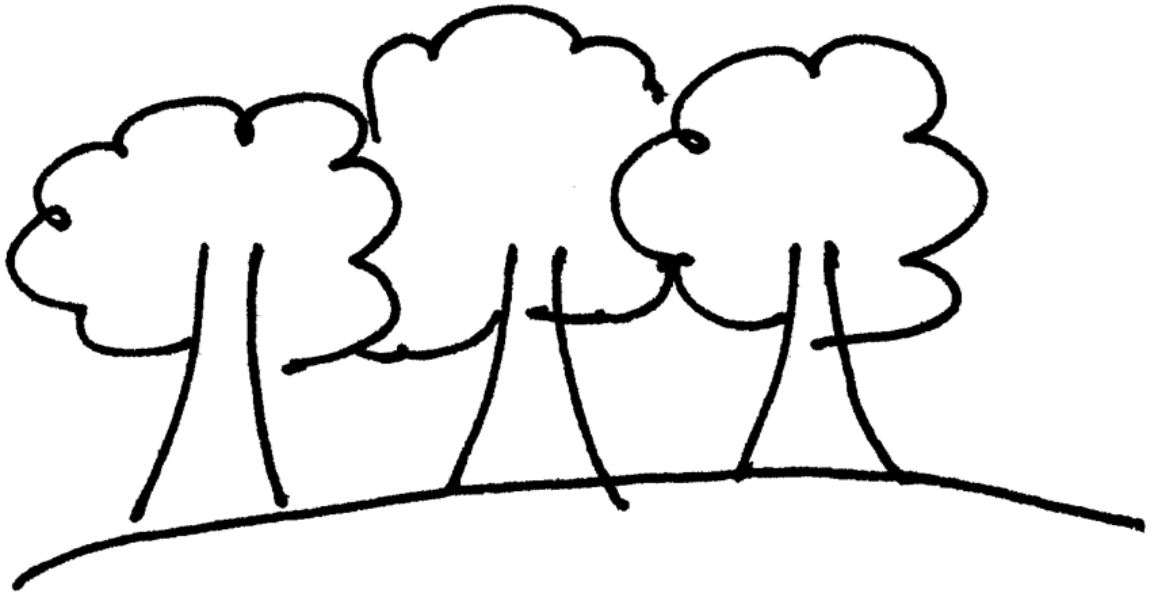
BeanShell Scripting language,
based on Java

BeanShell Overview – Parsing

- Takes source code and interprets it (not a compiler):
  - Lexical parsing checks the grammar and builds an Abstract Syntax Tree (AST)
  - BeanShell Authors used JavaCC implemented BeanShell parser
- Parsing is time consuming
  - Note: Rule Libraries are parsed per rule using them
- IdentityIQ caches the BeanShell instance (in BSFManager)
- The cache lasts a number of execution times across all threads
- Once the cache is destroyed the BeanShell code is re-parsed.

BeanShell Overview - Execution

- BeanShell heavily uses Java Reflection to execute the AST.
- Nodes contained in the AST tree are iterated over and executed by the BeanShell interpreter
- Variables and state are persisted in the BeanShell Namespace



BeanShell Overview - Performance

- Won't be as fast as Java, but does it matter?
  - Performance is still very good.
  - Many of the statements in BeanShell code are IO related, e.g. SailPointObject retrieval.
- Watch out for large Rule Libraries
- High iteration Rule's should be fine tuned for performance
- Complex BeanShell, consider moving to Java
  - IdentityIQ Plugins can be used for this
  - Hot deployed, just like Rules!
  - Classes can be access from BeanShell (if enabled)

```
if localVarHTTPResponse.StatusCode >= 300 {
        newErr := &GenericOpenAPIError{
            body:  localVarBody,
            error: localVarHTTPResponse.Status,
        }
```

Best Practices

```
if localVarHTTPResponse.StatusCode == 400 {
        var v ErrorResponseDto
        err = a.client.decode(&v, localVarBody, localVarHTT
        if err != nil {
            newErr.error = err.Error()
            return localVarReturnValue, localVarHTTPResp
        }
```

Best Practices – Logging

- For each Rule, create your own logger:

```
import org.apache.commons.logging.LogFactory;

log = LogFactory.getLog("org.rules.my_rule_name");
```

- "log" variable use optional, you can use whatever is required: "logger", "rulelog", "liblog"….
  - Recommend sticking with the classic "log".
- Don't set Log Level in code, allow for external injection to set the log level.
  - There are tools and plugins which manage the log level dynamically!

Best Practices – Logging

- Remember that expressions are always evaluated before the method is called:

```
log.debug( identity.toXml() );
```

- Even if debug is not enabled, the expression “identity.toXml()” is evaluated and passed to the debug method anyway.
- The debug method itself evaluates if the data is used.
- Only evaluate if debug is enabled:

```
if (log.isDebugEnabled()) log.debug( identity.toXml() );
```

Best Practices - Logging

- Checking debug is enabled adds noise to the code

```
if (log.isDebugEnabled()) log.debug( identity.toXml() );
```

- Only have to check if the debug statement is evaluating

```
log.debug( identity );
log.debug(“ Determining some logic here”);
```

- Are both light-weight.. No expressions are being evaluated, no need to check the log level.

Logging Façade

- You can use the sl4j logging façade which provides a richer API

```
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;

String test = someMethod(identity);
log = LoggerFactory.getLogger(”org.rules.my.rule_name");
log.debug(”Identity object ‘{}’ has test value {}", identity,
          "wobbler”, test);
```

- We can avoid evaluating an expression by using parameterized logging
- Allow the logger to perform method toString(), don’t call it yourself otherwise…it will be evaluated!

Best Practices - Query

- When querying for objects in BeanShell (or Java for that matter) be aware of that the following methods pull all the objects into memory first:
  - Iterator it = context.search(Identity.class, qo);
  - List list = context.getObjects(Identity.class, qo);
- Projected queries, is a database cursor
  - Iterator it = context.search(Identity.class, qo, fieldList);

Best Practices – Updating Objects

- Lock the object first (Object Util)
  `https://community.sailpoint.com/t5/Technical-White-Papers/BSDG-20-Locking-Identity-Objects-for-Modification/ta-p/76456`
- If fetching objects, decache when used
  - Note: context.decache(object) may not work as you expect!
  - Recommend, use context.decache()

Best Practices – Updating Objects

- Updating objects in IdentityIQ 8.0 and above will close database cursors. A change from previous releases
  https://community.sailpoint.com/t5/IdentityIQ-Articles/IdentityIQ-8-0-and-commitTransaction-While-Using-an-Iterator/ta-p/143225
- Use these iterators to update Object Datasets
  - IncrementalObjectIterator
  - IdIterator
- Update the QueryOptions:

```
QueryOptions qo = new QueryOptions();
qo.setCloneResults(true);
```

Best Practices – Hibernate

- Avoid using commitTransaction in Rule Hooks
  - Underlying process will also have its session committed.
  - No roll back possible after this point.
- decache is your friend, and it's cheap!
- If the rule is designed to query lots of objects… is that the best design?
- SailPoint Object ".toXml()" method is very expensive, avoid unless necessary.
- Avoid side effects of logging
- Avoid expensive evaluations, if possible short cut them where necessary

Best Practices – BeanShell ClassLoader

- BeanShell class loader can import Jar files dynamically
  - Be careful, performance is not great
- Avoid `import package.*,` wild card imports take a long time to complete.
- Plugin classes can be made available to the BeanShell class loader!

Best Practices – Monitor

- Use logs and their time stamps to discern execution times
- Use sailpoint.api.Meter to track execution time
  https://community.sailpoint.com/t5/Technical-White-Papers/BSDG-8-Measuring-the-Performance-of-Bean-Shell-Code/ta-p/73129

```
Meter.enterByName(”rule-identity-iteration");
…
Meter.exitByName(”rule-identity-iteration");
```

```
if localVarHTTPResponse.StatusCode >= 300 {
        newErr := &GenericOpenAPIError{
            body:  localVarBody,
            error: localVarHTTPResponse.Status,
        }
```

Testing

```
if localVarHTTPResponse.StatusCode == 400 {
        var v ErrorResponseDto
        err = a.client.decode(&v, localVarBody, localVarHTT
        if err != nil {
            newErr.error = err.Error()
            return localVarReturnValue, localVarHTTPResp
        }
```

Testing

- **iiq console –** Test rules in the IdentityIQ instance
- **IdentityIQ debug page –** Test rules, does not support arguments
- **PS JUnit helper –** Test rules in Java Code
- **DevSAK Plugin –** Test rules remotely

[Thank you!]