Object Query Language (OQL)

OQL is SQL-like query language to query Java heap. OQL allows to filter/select information wanted from Java heap. While pre-defined queries such as "show all instances of class X" are already supported by HAT, OQL adds more flexibility. OQL is based on JavaScript expression language.

OQL query is of the form

```
select <JavaScript expression to select>
[ from [instanceof] <class name> <identifier>
[ where <JavaScript boolean expression to filter> ] ]
```

where class name is fully qualified Java class name (example: java.net.URL) or array class name. [C is char array name, [Ljava.io.File; is name of java.io.File[] and so on. Note that fully qualified class name does not always uniquely identify a Java class at runtime. There may be more than one Java class with the same name but loaded by different loaders. So, class name is permitted to be id string of the class object. If **instanceof** keyword is used, subtype objects are selected. If this keyword is not specified, only the instances of exact class specified are selected. Both **from** and **where** clauses are optional.

In **select** and (optional) **where** clauses, the expression used in JavaScript expression. Java heap objects are wrapped as convenient script objects so that fields may be accessed in natural syntax. For example, Java fields can be accessed with obj.field_name syntax and array elements can be accessed with array[index] syntax. Each Java object selected is bound to a JavaScript variable of the identifier name specified in **from** clause.

OQL Examples

- select all Strings of length 100 or more

```
select s from java.lang.String s where s.value.length >= 100
```

- select all int arrays of length 256 or more

```
select a from [I a where a.length >= 256
```

- show content of Strings that match a regular expression

```
select s.value.toString() from java.lang.String s
where /java/.test(s.value.toString())
```

- show path value of all File objects

```
select file.path.value.toString() from java.io.File file
```

- show names of all ClassLoader classes

```
select classof(cl).name
```

```
from instanceof java.lang.ClassLoader cl
```

- show instances of the Class identified by given id string

```
select o from instanceof 0xd404b198 o
```

Note that 0xd404b198 is id of a Class (in a session). This is found by looking at the id shown in that class's page.

OQL built-in objects, functions

heap object

The **heap** built-in object supports the following methods:

- **heap.forEachClass** -- calls a callback function for each Java Class

```
heap.forEachClass(callback);
```

- **heap.forEachObject** -- calls a callback function for each Java object

```
heap.forEachObject(callback, clazz, includeSubtypes);
```

`clazz` is the class whose instances are selected. If not specified, defaults to java.lang.Object. `includeSubtypes` is a boolean flag that specifies whether to include subtype instances or not. Default value of this flag is true.

- **heap.findClass** -- finds Java Class of given name

```
heap.findClass(className);
```

where `className` is name of the class to find. The resulting Class object has following properties:
  - name - name of the class.
  - superclass - Class object for super class (or null if java.lang.Object).
  - statics - name, value pairs for static fields of the Class.
  - fields - array of field objects. field object has name, signature properties.
  - loader - ClassLoader object that loaded this class.
  - signers - signers that signed this class.
  - protectionDomain - protection domain to which this class belongs.

  Class objects have the following methods:
  - isSubclassOf - tests whether given class is direct or indirect subclass of this class or not.
  - isSuperclassOf - tests whether given Class is direct or indirect superclass of this class or not.
  - subclasses - returns array of direct and indirect subclasses.
  - superclasses - returns array of direct and indirect superclasses.
- **heap.findObject** -- finds object from given object id

```
heap.findObject(stringIdOfObject);
```

- **heap.classes** -- returns an enumeration of all Java classes
- **heap.objects** -- returns an enumeration of Java objects

```
heap.objects(clazz, [includeSubtypes], [filter])
```

`clazz` is the class whose instances are selected. If not specified, defaults to java.lang.Object. `includeSubtypes` is a boolean flag that specifies whether to include subtype instances or not. Default value of this flag is true. This method accepts an optional filter expression to filter the result set of objects.

- **heap.finalizables** -- returns an enumeration of Java objects that are pending to be finalized.
- **heap.livepaths** -- return an array of paths by which a given object is alive. This method accepts optional second parameter that is a boolean flag. This flag tells whether to include paths with weak reference(s) or not. By default, paths with weak reference(s) are not included.

```
select heap.livepaths(s) from java.lang.String s
```

Each element of this array itself is another array. The later array is contains an objects that are in the 'reference chain' of the path.

- **heap.roots** -- returns an Enumeration of Roots of the heap. Each Root object has the following properties:
  - id - String id of the object that is referred by this root
  - type - descriptive type of Root (JNI Global, JNI Local, Java Static etc)
  - description - String description of the Root
  - referrer - Thread Object or Class object that is responsible for this root or null

Examples:

- access static field 'props' of class java.lang.System

```
select heap.findClass("java.lang.System").statics.props
```

- get number of fields of java.lang.String class

```
select heap.findClass("java.lang.String").fields.length
```

- find the object whose object id is given

```
select heap.findObject("0xf3800b58")
```

- select all classes that have name pattern java.net.*

```
select filter(heap.classes(), "/java.net./.test(it.name)")
```

functions on individual objects

- allocTrace(jobject)
- classof(jobject)
- forEachReferrer(callback, jobject)

- identical(o1, o2)
- objectid(jobject)
- reachables(jobject, excludedFields)
- referrers(jobject)
- referees(jobject)
- refers(jobject)
- root(jobject)
- sizeof(jobject)
- toHtml(obj)

allocTrace function

This returns allocation site trace of a given Java object if available. allocTrace returns array of frame objects. Each frame object has the following properties:

- className - name of the Java class whose method is running in the frame.
- methodName - name of the Java method running in the frame.
- methodSignature - signature of the Java method running in the frame.
- sourceFileName - name of source file of the Java class running in the frame.
- lineNumber - source line number within the method.

classof function

Returns Class object of a given Java Object. The result object supports the following properties:

- name - name of the class.
- superclass - Class object for super class (or null if java.lang.Object).
- statics - name, value pairs for static fields of the Class.
- fields - array of field objects. Field objects have name, signature properties.
- loader - ClassLoader object that loaded this class.
- signers - signers that signed this class.
- protectionDomain - protection domain to which this class belongs.

Class objects have the following methods:

- isSubclassOf - tests whether given class is direct or indirect subclass of this class or not.
- isSuperclassOf - tests whether given Class is direct or indirect superclass of this class or not.
- subclasses - returns array of direct and indirect subclasses.
- superclasses - returns array of direct and indirect superclasses.

Examples:

- show class name of each Reference type object

```
select classof(o).name from instanceof java.lang.ref.Reference o
```

- show all subclasses of java.io.InputStream

```
select heap.findClass("java.io.InputStream").subclasses()
```

- show all superclasses of java.io.BufferedInputStream

```
select heap.findClass("java.io.BufferedInputStream").superclasses()
```

forEachReferrer function

calls a callback function for each referrer of a given Java object.

identical function

Returns whether two given Java objects are identical or not.

Example:

```
select identical(heap.findClass("Foo").statics.bar, heap.findClass("AnotherClass").statics.bar)
```

objectid function

Returns String id of a given Java object. This id can be passed to heap.findObject and may also be used to compare objects for identity.

Example:

```
select objectid(o) from java.lang.Object o
```

reachables function

Returns an array of Java objects that are transitively referred from the given Java object. Optionally accepts a second parameter that is comma separated field names to be excluded from reachability computation. Fields are written in class_name.field_name pattern.

Examples:

- print all reachable objects from each Properties instance.

```
select reachables(p) from java.util.Properties p
```

- print all reachables from each java.net.URL but omit the objects reachable via the fields specified.

```
select reachables(u, 'java.net.URL.handler') from java.net.URL u
```

referrers function

Returns an enumeration of Java objects that hold reference to a given Java object.

Examples:

- print number of referrers for each java.lang.Object instance

```
select count(referrers(o)) from java.lang.Object o
```

- print referrers for each java.io.File object

```
select referrers(f) from java.io.File f
```

- print URL objects only if referred by 2 or more

```
select u from java.net.URL u where count(referrers(u)) > 2
```

referees function

Returns an array of Java objects to which the given Java object directly refers to.

Example: to print all static reference fields of java.io.File class

```
select referees(heap.findClass("java.io.File"))
```

refers function

Returns whether first Java object refers to second Java object or not.

root function

If given object is a member of root set of objects, this function returns a descriptive Root object describing why it is so. If given object is not a root, then this function returns null.

sizeof function

Returns size of the given Java object in bytes Example:

```
select sizeof(o) from [I o
```

toHtml function

Returns HTML string for the given Java object. Note that this is called automatically for objects selected by select expression. But, it may be useful to print more complex output. Example: print hyperlink in bold font weight

```
select "<b>" + toHtml(o) + "</b>" from java.lang.Object o
```

Selecting multiple values

Multiple values can be selected using JavaScript object literals or arrays.

Example: show name and thread for each thread object

```
select { name: t.name? t.name.toString() : "null", thread: t }
from instanceof java.lang.Thread t
```

array/iterator/enumeration manipulation functions

These functions accept an array/iterator/enumeration and an expression string [or a callback function] as input. These functions iterate the array/iterator/enumeration and apply the

expression (or function) on each element. Note that JavaScript objects are associative arrays. So, these functions may also be used with arbitrary JavaScript objects.

- concat(array1/enumeration1, array2/enumeration2)
- contains(array/enumeration, expression)
- count(array/enumeration, expression)
- filter(array/enumeration, expression)
- length(array/enumeration)
- map(array/enumeration, expression)
- max(array/enumeration, [expression])
- min(array/enumeration, [expression])
- sort(array/enumeration, [expression])
- sum(array/enumeration, [expression])
- toArray(array/enumeration)
- unique(array/enumeration, [expression])

concat function

Concatenates two arrays or enumerations (i.e., returns composite enumeration).

contains function

Returns whether the given array/enumeration contains an element the given boolean expression specified in code. The code evaluated can refer to the following built-in variables.

- it -> currently visited element
- index -> index of the current element
- array -> array/enumeration that is being iterated

Example: select all Properties objects that are referred by some static field some class.

```
select p from java.util.Properties p
where contains(referrers(p), "classof(it).name == 'java.lang.Class'")
```

count function

count function returns the count of elements of the input array/enumeration that satisfy the given boolean expression. The boolean expression code can refer to the following built-in variables.

- it -> currently visited element
- index -> index of the current element
- array -> array/enumeration that is being iterated

Example: print number of classes that have specific name pattern

```
select count(heap.classes(), "/java.io./.test(it.name)")
```

filter function

filter function returns an array/enumeration that contains elements of the input array/enumeration that satisfy the given boolean expression. The boolean expression code can

refer to the following built-in variables.

- it -> currently visited element
- index -> index of the current element
- array -> array/enumeration that is being iterated
- result -> result array/enumeration

Examples:

- show all classes that have java.io.* name pattern

```
select filter(heap.classes(), "/java.io./.test(it.name)")
```

- show all referrers of URL object where the referrer is not from java.net package

```
select filter(referrers(u), "! /java.net./.test(classof(it).name)")
from java.net.URL u
```

length function

length function returns number of elements of an array/enumeration.

map function

Transforms the given array/enumeration by evaluating given code on each element. The code evaluated can refer to the following built-in variables.

- it -> currently visited element
- index -> index of the current element
- array -> array/enumeration that is being iterated
- result -> result array/enumeration

map function returns an array/enumeration of values created by repeatedly calling code on each element of input array/enumeration.

Example: show all static fields of java.io.File with name and value

```
select map(heap.findClass("java.io.File").statics, "index + '=' + toHtml(it)")
```

max function

returns the maximum element of the given array/enumeration. Optionally accepts code expression to compare elements of the array. By default numerical comparison is used. The comparison expression can use the following built-in variables:

- lhs -> left side element for comparison
- rhs -> right side element for comparison

Examples:

- find the maximum length of any String instance

```
select max(map(heap.objects('java.lang.String', false), 'it.value.length'))
```

- find string instance that has the maximum length

```
select max(heap.objects('java.lang.String'), 'lhs.value.length > rhs.value.length')
```

min function

returns the minimum element of the given array/enumeration. Optionally accepts code expression to compare elements of the array. By default numerical comparison is used. The comparison expression can use the following built-in variables:

- lhs -> left side element for comparison
- rhs -> right side element for comparison

Examples:

- find the minimum size of any Vector instance

```
select min(map(heap.objects('java.util.Vector', false), 'it.elementData.length'))
```

- find Vector instance that has the maximum length

```
select min(heap.objects('java.util.Vector'), 'lhs.elementData.length < rhs.elementData.length')
```

sort function

sorts given array/enumeration. Optionally accepts code expression to compare elements of the array. By default numerical comparison is used. The comparison expression can use the following built-in variables:

- lhs -> left side element for comparison
- rhs -> right side element for comparison

Examples:

- print all char[] objects in the order of size.

```
select sort(heap.objects('[C'), 'sizeof(lhs) - sizeof(rhs)')
```

- print all char[] objects in the order of size but print size as well.

```
select map(sort(heap.objects('[C'), 'sizeof(lhs) - sizeof(rhs)'), '{ size: sizeof(it), obj: it }')
```

sum function

This function returns the sum of all the elements of the given input array or enumeration. Optionally, accepts an expression as second param. This is used to map the input elements before summing those.

Example: return sum of sizes of the reachable objects from each Properties object

```
select sum(map(reachables(p), 'sizeof(it)'))
from java.util.Properties p

// or omit the map as in ...
select sum(reachables(p), 'sizeof(it)')
from java.util.Properties p
```

toArray function

This function returns an array that contains elements of the input array/enumeration.

unique function

This function returns an array/enumeration containing unique elements of the given input array/enumeration

Example: select unique char[] instances referenced from Strings. Note that more than one String instance can share the same char[] for the content.

```
// number of unique char[] instances referenced from any String
select count(unique(map(heap.objects('java.lang.String'), 'it.value')))

// total number of Strings
select count(heap.objects('java.lang.String'))
```

More complex examples

Print histogram of each class loader and number of classes loaded by it

```
select map(sort(map(heap.objects('java.lang.ClassLoader'),
'{ loader: it, count: it.classes.elementCount }'), 'lhs.count < rhs.count'),
'toHtml(it) + "<br>"')
```

The above query uses the fact that, **java.lang.ClassLoader** has a private field called **classes** of type **java.util.Vector** and Vector has a private field named **elementCount** that is number of elements in the vector. We select multiple values (loader, count) using JavaScript object literal and map function. We sort the result by count (i.e., number of classes loaded) using sort function with comparison expression.

Show parent-child chain for each class loader instance

```
select map(heap.objects('java.lang.ClassLoader'),
   function (it) {
      var res = '';
      while (it != null) {
         res += toHtml(it) + "->";
         it = it.parent;
      }
      res += "null";
      return res + "<br>";
   })
```

Note that we use **parent** field of **java.lang.ClassLoader** class and walk until parent is null using the callback function to map call.

Printing value of all System properties

```
select map(filter(heap.findClass('java.lang.System').statics.props.table, 'it != null'),
        function (it) {
            var res = "";
            while (it != null) {
                res += it.key.value.toString() + '=' +
                       it.value.value.toString() + '<br>';
                it = it.next;
            }
            return res;
        });
```

The above query uses the following facts:

- java.lang.System has static field by name 'props' of type java.util.Properties.
- java.util.Properties has field by 'table' of type java.util.Hashtable$Entry (this field is inherited from java.util.Hashtable). This is the hashtable buckets array.
- java.util.Hashtable$Entry has 'key', 'value' and 'next' fields. Each entry points the next entry (or null) in the same hashtable bucket.
- java.lang.String class has 'value' field of type char[].

Note that this query (and many other queries) may not be stable - because private fields of Java platform classes may be modified/removed without any notification! (implementation detail). But, using such queries on user classes may be safe - given that user has the control over the classes.