ops4j

Ops4j is a framework which makes Java code behave like native tools such as `ls`, `grep` and `dir`. Code is written in small single purpose units called operations. Operations behave like native utilities. Operations provide usage, help and flexible options. Operations can be combined with other operations to solve bigger problems.

Ops4j can be used to augment shell scripting environments with custom, portable, commands which can be used identically across windows, linux and other windows variants.

Quick Examples

One of the most versatile operations is the `map` operation.

> With ops4j we can manufacture realistic data in whatever quantity we require.

```
# Create 2 random contact records with id, name and cell phone number.
$ map -D 2 /id='seq()' /name='gen:name()' /cell='gen:cell()' > contacts.json

# Inspect the contents.
$ cat contacts.json
{"id":1,"name":"Lawerence Trantow","cell":"(326) 424-7300"}
{"id":2,"name":"Audie Schaefer","cell":"933-385-8616"}
```

> Now that we have our data, we can stream data into a mongo then back again.

```
# Save contacts into a mongo collection
$ cat contacts.json | mongo:insert -d ops -c contacts

# Stream the contacts back from mongo
$ mongo:stream -d ops -c contacts

{"id":1,"name":"Lawerence Trantow","cell":"(326) 424-7300"}
{"id":2,"name":"Audie Schaefer","cell":"933-385-8616"}
```

> We can do the same thing with relational databases.

```
# Create the contacts
$ cat contacts.json | jdbc:create contacts

# Stream them back
$ jdbc:stream 'select * from contacts'

{"NAME":"Lawerence Trantow","CELL":"(326) 424-7300"}
{"NAME":"Audie Schaefer","CELL":"933-385-8616"}
```

> Work can be saved to operation repositories (op-repos) and shared with others. Consider this ETL process which adds a timestamp to a record before inserting it into a database outputting benchmark information every 1,000 insertions.

```
$ map /=/ /time='now()' | mongo:insert -d ops -c my-collection | benchmark 1000
```

we can save this by wrapping it into a pipeline and serializing

Core Concepts

There are a number of core concepts which bear mentioning.

CONCEPT	DESCRIPTION
operation	A unit of code which operates upon each JSON document within the stream. For example, the operation `mongo:insert` will insert each JSON document in the stream into the designated mongo collection.
node operation	A unit of code which operates at a single JSON node level. For example, the `encrypt` node operation will emit a text node containing the encrypted and base-64 encoded value of the input node.
operation repository	A place to save configured operations by name. Example Repositories: filesystem, mongo, jdbc
operation lifecycle	A metadata object which describes the lifecycle of a given operation. INITIALIZE = intialize the operation. OPEN = open the operation RUN = process each json record in the stream. CLOSE = close the operation CLEANUP = perform any cleanup for the operation.
operation data	Also known as OpData, a single record of data upon which an operation performs it's task. Ultimately, OpData is a thin wrapper on top of Jackson JSON data.

Under the hood

With ops4j, developers write to a specific contract defined in `Op`. Code written to this contract are known as operations and can be used as native OS tooling.

Consider the highly useful `pause` operations. Developers write logic like:

```
@AutoService(Op.class) @Command(name = "pause",
    description = "Pause execution for the specified number of milliseconds.")
public class Pause extends BaseOp<Pause>
{
  @Parameters(index = "0", arity = "1",
      description = "The number of milliseconds to pause.")
  private @Getter @Setter Long pause = 1000L;

  public Pause()
  {
    super("pause");
  }

  public List<OpData> execute(OpData input)
  {
    debug("pause ", getPause(), " ms.");

    ThreadUtil.sleep(getPause());
    return input.asList();
  }

  public static void main(String args[]) throws OpsException
  {
    OpCLI.cli(new Pause(), args);
  }
}
```

and gain a tool which can give the user usage information:

```
$ pause -h
Usage: pause [-C=<view>] [-L=<logLevel>] [-N=<name>] <pause>

Pause execution for the specified number of milliseconds.

Additional CLI Options: [-hHP] [-D=<dataSource>] [-O=<outputType>]
                        [-S=<serializationType>] [-LL=<String=LogLevel>]...
```

This code will expose itself directly to the CLI.For their efforts will receive a traditional Java component which can be

Operations can be combined with other operations to solve greater problems. These solutions can be saved to `operation repositories` for common or personal use.

Ops4j puts Java to work on everyday problems without all of the ceremony required by more specialized environments such as J2EE or Spring based microservices.

Removed from it's ivory tower pedestal, this code is now used continuously as one would use a hammer. As the OSS adage goes, `Many eyes make all bugs shallow`. Defects of such common use utility code should be more quickly fixed.

Ops4j removes environmental barriers by transforming the native os shell (bash) into an agile environment for solving problems quickly. With ops4j the code is no longer buried under layers of process and environmental variables. With ops4j, we deal with our code (operations).

Ops4J is a Java framework which integrates Java code seamlessly into the native OS CLI. Ops4J code behaves identically to other shell utilities. These shell commands can be combined with other shell commands to solve even larger problems. This makes ops4j a nice tool for automation.

Ops4j solutions take the form of shell commands which can be stored and run anywhere.

concepts

CONCEPT	DESCRIPTION
operation	A single unit of code which operates upon each JSON document within the stream. For example, the operation `mongo:insert` will insert each JSON document in the stream into the designated mongo collection.
node operation	A single unit of code which operates at a single JSON node level. For example, the `encrypt` node operation will emit a text node containing the encrypted and base-64 encoded value of the input node.
operation Repository	A place to save configured operations by name. Example Repositories: filesystem, mongo, jdbc
operation lifecycle	A metadata object which describes the lifecycle of a given operation. INITIALIZE = intialize the operation. OPEN = open the operation RUN = process each json record in the stream. CLOSE = close the operation CLEANUP = perform any cleanup for the operation.
operation data	Also known as OpData, a single record of data upon which an operation performs it's task. Ultimately, OpData is a thin wrapper on top of Jackson JSON data.