Fungal User's Guide

A High-Performance Java Kernel

Table of Contents

1. About Fungal 1
    1.1. The team 1
2. Introduction 2
    2.1. Highlights 2
3. Download 3
    3.1. Download 3
    3.2. Git Access 3
4. Installation 5
    4.1. Standalone environments 5
5. Configuration 6
    5.1. Kernel configuration 6
        5.1.1. bindAddress 6
        5.1.2. classLoader 6
        5.1.3. command 7
        5.1.4. configuration 7
        5.1.5. deploy 7
        5.1.6. eventListener 7
        5.1.7. home 7
        5.1.8. hotDeploy 8
        5.1.9. hotDeployInterval 8
        5.1.10. library 8
        5.1.11. name 8
        5.1.12. parallelDeploy 9
        5.1.13. remoteAccess 9
        5.1.14. remotePort 9
        5.1.15. system 9
        5.1.16. threadGroup 9
6. Booting 10
    6.1. Booting the kernel 10
7. Lifecycle 13
    7.1. Lifecycle of beans 13
8. Deployment 14
    8.1. Deployers 14
        8.1.1. Advanced deployers 14
    8.2. Archive deployment 15
9. Classloading 17
    9.1. Class loading architecture 17
    9.2. Class loader types 17
10. Remote communition 19
    10.1. Remote protocol 19
    10.2. Built-in commands 19
        10.2.1. Help 19
        10.2.2. Deploy 20
        10.2.3. Undeploy 20

10.3. Internal commands 21
10.3.1. GetCommand 21
11. Usage 22
11.1. Basic usage 22
12. Community 23
12.1. Website 23
12.2. Issue tracking 23
13. Troubleshooting 24
13.1. I think I have found a bug 24
13.2. I would like to implement a feature 24

1

About Fungal

The goal of the Fungal project is to provide a high-performance Java kernel environment.

1.1. The team

Jesper Pedersen acts as the lead for the Fungal project. He can be reached at jesper (dot) pedersen (at) comcast (dot) net.

2

Introduction

The Fungal kernel is a Plain Old Java Object (POJO) based kernel which can build a container environment for Java services.

The focus of the Fungal kernel is simplicity and high performance.

The Fungal kernel has no dependencies on any 3rd party libraries as the kernel is built directly on the Java 6 API.

2.1. Highlights

Highlights of the Fungal kernel:

- 100% Pure Java 6 implementation
- Bean definition language
- Lifecycle of beans
- Parallel deployment
- Dependency management
- Classloader architecture
- Deployer framework
- Hot deployment
- Communication protocol
- Core services

3

Download

The official Fungal project page is http://jesperpedersen.github.com/fungal/ where you can download the software.

3.1. Download

The download location is zone: http://jesperpedersen.github.com/fungal/

Each release is labelled with a version number and an identifier.

```
fungal-<major>.<minor>.<patch>[.<identifier>]
```

where

- Major: The major version number. Signifies major changes in the implementation.
- Minor: The minor version number. Signifies functional changes to a major version.
- Patch: The patch version number. Signifies a binary compatible change to a minor version.
- Identifier: The identifier. Identifies the level of the quality of the release.
    - None / GA: Stable release
    - CR: Candidate for Release quality. The implementation is functional complete.
    - Beta: Beta quality. The implementation is almost functional complete.
    - Alpha: Alpha quality. The implementation is a snapshot of the development.

An example

```
fungal-1.0.0.tar.gz
```

which is the first stable release of the project.

3.2. Git Access

If you want to experiment with the latest developments you may checkout the latest code from Git trunk. Be aware that the information provided in this manual might then not be accurate.

The Git repository is located under:

```
git checkout git://github.com/fungal.git
```

```

```

You can find additional information about this in the developer guide.

4

Installation

This chapter describes how the Fungal kernel should be installed.

4.1. Standalone environments

The Fungal kernel provides the foundation for standalone Java environments where it enables deployment of the services which makes up those environments.

The Fungal kernel should be deployed as part of the application classloader for the application. This is typically done by bundling all the classes Fungal that makes up the kernel.

The bundle would also include the main class for the environment, which sets up the kernel and its configuration and all utility classes such as kernel event listeners.

This setup can be done by simply `unjar`ing the `fungal.jar` archive

```
unjar xf fungal.jar
```

and then copy the applications main class and its dependencies into the directory.

The application can then be started using a `Main-Class` attribute in the `MANIFEST.MF` file or by specifying the main class on the command line.

5

Configuration

5.1. Kernel configuration

The kernel is configured through the

```
com.github.fungal.api.configuration.KernelConfiguration
```

class, which allows the developer to configure the kernel in the most optimal way for the container configuration.

It is a best practice to give the configuration a name using the `name` method, otherwise the kernel will use the default name of `fungal`.

Based on the kernel configuration a number of system properties are set

Table 5.1. Kernel system properties

Name	Description
<name>.home	The root of the container configuration
<name>.bindaddress	The network bind address of the container configuration

Furthermore, the kernel makes use the Java system property `java.io.tmpdir` for temporary files if needed.

5.1.1. `bindAddress`

The `bindAddress` parameter specifies where the communication server which handles remote access should be bound.

An example

```
kernelConfiguration.bindAddress("localhost");
```

5.1.2. `classLoader`

The `classLoader` parameter specifies the classloading model for the kernel.

An example

```
kernelConfiguration.classLoader(ClassLoaderFactory.PARENT_FIRST);
```

5.1.3. `command`

The `command` parameter allows the developer to install additional commands that should be available remotely.

An example

```
kernelConfiguration.command(new ShutdownCommand());
```

5.1.4. `configuration`

The `configuration` parameter specifies the configuration directory for the environment where all configuration files are located under `home`.

An example

```
kernelConfiguration.configuration("config");
```

5.1.5. `deploy`

The `deploy` parameter specifies the deploy directory for the environment where all user deployment are located under `home`.

An example

```
kernelConfiguration.deploy("deploy");
```

5.1.6. `eventListener`

The `eventListener` parameter allows the developer to install additional kernel event listeners.

An example

```
kernelConfiguration.eventListener(new PreClassLoaderEventListener());
```

5.1.7. `home`

The `home` parameter specifies the top-level directory of the environment.

An example

```
kernelConfiguration.home(new File(".").getParent().toURI().toURL());
```

5.1.8. `hotDeploy`

The `hotDeploy` parameter specifies if hot deployment should be enabled for the `deploy` directory.

An example

```
kernelConfiguration.hotDeploy(true);
```

5.1.9. `hotDeployInterval`

The `hotDeployInterval` parameter specifies the number of seconds between each scan.

An example

```
kernelConfiguration.hotDeployInterval(5);
```

5.1.10. `library`

The `library` parameter specifies the library directory where all the environment libraries are located under `home`.

An example

```
kernelConfiguration.library("lib");
```

5.1.11. `name`

The `name` parameter specifies the unique name of the environment which is used to generate for example a MBean name space and system properties.

An example

```
kernelConfiguration.name("fungal");
```

5.1.12. `parallelDeploy`

The `parallelDeploy` parameter specifies if deployments should be done in parallel for the `deploy` and `system` directories.

An example

```
kernelConfiguration.parallelDeploy(true);
```

5.1.13. `remoteAccess`

The `remoteAccess` parameter specifies if remote access to the kernel should be enabled.

An example

```
kernelConfiguration.remoteAccess(true);
```

5.1.14. `remotePort`

The `remotePort` parameter specifies the port number for remote access.

An example

```
kernelConfiguration.remotePort(1202);
```

5.1.15. `system`

The `system` parameter specifies the system directory where system deployments are located under `home`.

An example

```
kernelConfiguration.system("system");
```

5.1.16. `threadGroup`

The `threadGroup` parameter specifies the thread group that should be used for kernel threads.

An example

```
kernelConfiguration.threadGroup(new MyThreadGroup());
```

6

Booting

6.1. Booting the kernel

The kernel is booted using a

```
${<name>.home}/config/bootstrap.xml
```

file, where references to URLs for bean deployments.

The layout of bootstrap.xml is defined in

```
${<name>.home}/config/bootstrap.xsd
```

The

```
${<name>.home}/config/deployment.xsd
```

file defines how bean deployments are constructed.

In a bean deployment file there is support for the following constructs:

- `<bean>`

  which initialize a bean based on the class attribute.

- `<property>`

  which sets a property on the bean - being an object or a primitive.

- `<inject>`

  injects a bean or a property from a bean (defines an implicit dependency on the bean).

- `<constructor>`

  which defines the constructor method to use for a bean.

- `<parameter>`

  defines a parameter value for a constructor.

- `<factory>`

  defines a factory bean for a constructor.

- `<depends>`

  defines an explicit dependency to another bean.

- `<install>`

  defines an install method.

- `<uninstall>`

  defines an uninstall method.

- `<incallback>`

  defines a method which is invoked with all beans that has the type of the first parameter when the installed.

- `<uncallback>`

  defines a method which is invoked with all beans that has the type of the first parameter when the uninstalled.

- `<map>`

  defines a map data structure.

- `<list>`

  defines a list data structure.

- `<set>`

  defines a set data structure.

- `<null>`

  defines a null value.

- `<this>`

  defines a reference to the object instance.

In order to define locations relative to the install root of the Fungal container configuration the variable

```
${<name>.home}
```

can be used. An example would be `${jboss.jca.home}`.

There is support for accessing system properties using the "${property}" mechanism with an optional default value

```
${jboss.jca.host:localhost}
```

A configuration example would be

```
<!-- Transaction manager -->
<bean name="TransactionManager" class="com.arjuna.ats.jbossatx.jta.TransactionManagerService">
   <property name="transactionTimeout">300</property>
   <property name="objectStoreDir">${jboss.jca.home}/tmp/tx-object-store</property>
   <property name="mbeanServer"><inject bean="JMX" property="mbeanServer"/></property>
   <property name="transactionStatusManagerInetAddress">${jboss.jca.host:localhost}</property>
   <property name="transactionStatusManagerPort">4713</property>
   <property name="recoveryInetAddress">${jboss.jca.host:localhost}</property>
   <property name="recoveryPort">4712</property>
   <property name="socketProcessIdPort">0</property>
   <depends>NamingServer</depends>
 </bean>
```

7

Lifecycle

7.1. Lifecycle of beans

There is support for lifecycle methods. These include

- `public void create()`

  Called after a bean has been constructed.

- `public void start()`

  Called after create.

- `public void stop()`

  Called when bean is stopped.

- `public void destroy()`

  Called after stop.

The lifecycle methods can be ignored by using the following constructs:

- `<ignoreCreate/>`

  which ignores the `create` method.

- `<ignoreStart/>`

  which ignores the `start` method.

- `<ignoreStop/>`

  which ignores the `stop` method.

- `<ignoreDestroy/>`

  which ignores the `destroy` method.

Furthermore all `install` methods defined for the bean are called after the `start` method. All `uninstall` methods defined for the bean are called before the `stop` method.

8

Deployment

8.1. Deployers

Fungal features a very simple deployers mechanism.

Each deployer is defined as a bean in a bean deployment XML file and they must implement the

```
com.github.fungal.spi.deployers.Deployer
```

interface.

The interface contains the

```
/**
 * Deploy
 * @param url The URL for the deployment
 * @param parent The parent classloader
 * @return The deployment; or null if no deployment was made
 * @exception DeployException Thrown if an error occurs
 */
public Deployment deploy(URL url, ClassLoader parent) throws DeployException;
```

method which is invoked once with each of the files in the $CONF_HOME/deploy/ directory.

The `Deployment` interface returned represents the deployment unit or `null` if a deployment unit wasn't created.

The implementation of the deployer bean must be thread-safe. This can be done by using the `synchronized` keyword on the `deploy` method.

8.1.1. Advanced deployers

The kernel supports deploying multiple artifacts of the same type at the same time for optimal performance.

If a deployer supports this mechanism it must implement the

```
com.github.fungal.spi.deployers.CloneableDeployer
```

interface.

The interface contains the

```
/**
 * Clone the deployer
 * @return A copy of the deployer
 * @exception CloneNotSupportedException Thrown if the copy operation isn't supported
 */
public Deployer clone() throws CloneNotSupportedException;
```

method which constructs a new instance of the deployer.

Implementing this interface means that the `deploy` method doesn't have to be `synchronized`. However any state that should be shared between all the deployer instances should be `static` and have atomic access.

8.2. Archive deployment

Fungal uses multiple threads during the booting sequence.

When multiple threads are used in the kernel a dependency mechanism between beans is needed. The kernel keeps track of each bean deployment and assigns a status

- `NOT_STARTED`

  The bean hasn't started.

- `STARTING`

  The bean is starting.

- `STARTED`

  The bean has fully started.

- `STOPPING`

  The bean is stopping.

- `ERROR`

  The bean is in error.

Caveats: The kernel currently doesn't detect cyclic dependencies between deployment units.

Furthermore the dependencies between beans are recorded in order to be able to safely shutdown deployments.

The Fungal kernel can deploy the archives in the `deploy` directory in parallel if specified by the `KernelConfiguration.parallelDeploy()` method. Otherwise are all the deployment units booted in sequence as defined by the `File.listFiles()` method.

The Fungal kernel features a hot deployer, which scans the deployment directory at the specified intervals for new,

changed or removed deployment units. The properties of the hot deployer can be controlled through the `KernelConfiguration` object or through JMX.

9

Classloading

This chapter describes the class loading abilities of the Fungal kernel.

9.1. Class loading architecture

Fungal features a 3-tier classloader architecture:

1. Application classloader (AppCL).

   Top-level classloader that loads the Fungal kernel. This class loader is controlled by the application that uses the kernel.

2. Kernel classloader (KernelCL).

   A class loader with AppCL as its parent which loads all libraries used by the kernel.

3. Deployment classloader (DCL).

   Each deployment can run in its own classloader with KernelCL as its parent.

The application using the Fungal kernel chooses what should be located in the application classloader (AppCL). Typically the Fungal kernel is bundled in this classloader.

The kernel classloader model is chosen using the `KernelConfiguration.classLoader()` method which takes a classloader type. This layer is typically using the `TYPE_PARENT_FIRST` type for a "flat" classloader model for its libraries or the `TYPE_EXPORT` type for an environment where multiple versions of the same library are deployed in the global scope.

The default configuration for the kernel is `TYPE_PARENT_FIRST` as specified in `KernelConfiguration.classLoader()`.

The deployment classloader is specific to the application and can involve more than one layer of classloaders. Typically this layer uses `TYPE_PARENT_FIRST` for a configuration where the environment's libraries have precedence or `TYPE_PARENT_LAST` if libraries bundled with the application has precedence.

9.2. Class loader types

The Fungal kernel class loader mechanisms are controlled through the

```
com.github.fungal.api.classloading.ClassLoaderFactory
```

```
```

class which creates class loaders based on the parameters given.

The following types are supported:

- `TYPE_PARENT_FIRST`

  The parent class loader is checked before the child class loader.

- `TYPE_PARENT_LAST`

  The parent class loader is after the child class loader.

- `TYPE_EXPORT`

  This class loader type can be compared to the OSGi class loader model where each Java library that contains OSGi manifest information will be isolated. This allows multiple versions of the same library to be active at the same time and will allow deployment to declare their dependencies on a specific library version.

  Note, that this type is not a valid OSGi class loader model implementation, but only borrows some of the ideas in that model.

All class loader types are implemented as a `com.github.fungal.api.classloading.KernelClassLoader` which extends the `java.net.URLClassLoader` class.

10

Remote communition

10.1. Remote protocol

Fungal features a remote protocol to allow for example deployment and undeployment of archives.

The protocol is based on the `java.io.ObjectInputStream` and `java.io.ObjectOutputStream` format over a standard `java.net.Socket` connection.

The protocol consist of two parts

1. The name of the command
2. The arguments for the command

The arguments is optional if the command doesn't take any.

All commands return a `java.io.Serializable` which represents the result of running the command.

The name of the command is written using `writeUTF()` and read using `readUTF()`.

Each of the arguments is written using `writeObject()` and read using `readObject()`.

The kernel must be configured with `KernelConfiguration.remoteAccess(true)` in order to enable remote access.

10.2. Built-in commands

If remote access is enabled in the kernel then the following commands are enabled.

10.2.1. Help

Displays the available commands.

Table 10.1. Help: Input

Type	Value	Description
UTF	`help`	Command name

Table 10.2. Help: Output

Type	Value	Description
OBJECT	<message>	List of available commands

10.2.2. Deploy

Deploys a file.

Table 10.3. Deploy: Input

Type	Value	Description
UTF	`deploy`	Command name
OBJECT	<file url>	Location of the deployment unit as an `java.net.URL`

Table 10.4. Deploy: Output

Type	Value	Description
OBJECT	<message>	Empty string if successful. Otherwise error description

10.2.3. Undeploy

Undeploys a file.

Table 10.5. Undeploy: Input

Type	Value	Description
UTF	`undeploy`	Command name
OBJECT	<file url>	Location of the deployment unit as an `java.net.URL`

Table 10.6. Undeploy: Output

Type	Value	Description
OBJECT	<message>	Empty string if successful. Otherwise error description

10.3. Internal commands

The following internal commands are available.

10.3.1. GetCommand

Get the parameter types for a command.

Table 10.7. GetCommand: Input

Type	Value	Description
UTF	`getcommand`	Command name
OBJECT	<command>	The name of the command

Table 10.8. GetCommand: Output

Type	Value	Description
OBJECT	<parameters>	The list of parameter types (`Class[]`). Or `null` if none.

11

Usage

11.1. Basic usage

The Fungal kernel is constructed and started using

```
import com.github.fungal.api.Kernel;
import com.github.fungal.api.KernelFactory;
import com.github.fungal.api.configuration.KernelConfiguration;
import com.github.fungal.api.deployer.MainDeployer;

// Create a kernel configuration and start the kernel
KernelConfiguration kernelConfiguration = new KernelConfiguration();
kernelConfiguration = kernelConfiguration.name("jboss.jca");
kernelConfiguration = kernelConfiguration.remoteAccess(false);
kernelConfiguration = kernelConfiguration.hotDeployment(false);

Kernel kernel = KernelFactory.create(kernelConfiguration);
kernel.startup();

// Get the reference to the main deployer
MainDeployer mainDeployer = kernel.getMainDeployer();

// Reference to my deployment
URL myDeployment = ...;

mainDeployer.deploy(myDeployment);
```

where the `KernelConfiguration` object allows you to configure the kernel setup. The `MainDeployer` allows you to deploy and undeploy deployment units that are supported.

The kernel is stopped using

```
kernel.shutdown();
```

See the JavaDoc for additional details.

12

Community

12.1. Website

The website contains the latest information about the project and links to important information.

The website is located at http://jesperpedersen.github.com/fungal/

12.2. Issue tracking

We are using the GitHub issue system to manage our issues in the project.

These are divided into the following categories

- Feature Request: A feature that you would like see implemented.
- Bug: A software defect.

Our issue tracking system located at http://github.com/jesperpedersen/fungal/issues

13

Troubleshooting

13.1. I think I have found a bug

If you think you have found a bug you should verify this by posting to our forum first.

Our forum is located at

You can also search our issue tracking system located at

13.2. I would like to implement a feature

So you have found an area where you are missing a feature and would like to submit a patch for it, great !

There are a couple of steps to get a feature included

First, you should create a new thread in our development forum where you describe the feature, its design and implementation.

Once there is an agreement on the feature and the design you should proceed with creating the patch.

To maximize your chances of getting the feature in the official build as soon as possible make sure that you run through the following steps:

```
ant clean test
ant clean checkstyle
ant clean findbugs
ant clean cobertura
```

All these should show that,

1. All your test cases for the feature is passing
2. Your code is correctly formatted accoording to project rules
3. There isn't any bug reports from the Findbugs environment
4. There is full code coverage based on the Cobertura report

when done, create a pull request. See the developer guide for additional details.

Happy Coding !