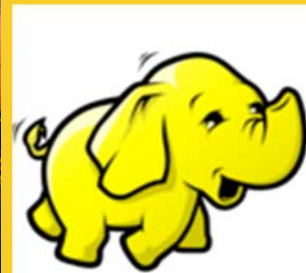




# HDFS - Java API

Originals of slides and source code for examples: <http://www.coreservlets.com/hadoop-tutorial/>  
Also see the customized Hadoop training courses (onsite or at public venues) – <http://courses.coreservlets.com/hadoop-training.html>

**Customized Java EE Training:** <http://courses.coreservlets.com/>  
Hadoop, Java, JSF 2, PrimeFaces, Servlets, JSP, Ajax, jQuery, Spring, Hibernate, RESTful Web Services, Android.  
Developed and taught by well-known author and developer. At public venues or onsite at *your* location.



**For live customized Hadoop training (including prep for the Cloudera certification exam), please email [info@coreservlets.com](mailto:info@coreservlets.com)**

**Taught by recognized Hadoop expert who spoke on Hadoop several times at JavaOne, and who uses Hadoop daily in real-world apps. Available at public venues, or customized versions can be held on-site at your organization.**

- Courses developed and taught by Marty Hall
  - JSF 2.2, PrimeFaces, servlets/JSP, Ajax, jQuery, Android development, Java 7 or 8 programming, custom mix of topics
  - Courses available in any state or country. Maryland/DC area companies can also choose afternoon/evening courses.
- Courses developed and taught by [coreservlets.com](http://coreservlets.com) experts (edited by Marty)
  - Spring, Hibernate/JPA, GWT, Hadoop, HTML5, RESTful Web Services

Contact [info@coreservlets.com](mailto:info@coreservlets.com) for details



# Agenda

- **Java API Introduction**
- **Configuration**
- **Reading Data**
- **Writing Data**
- **Browsing file system**

4

# File System Java API

- **org.apache.hadoop.fs.FileSystem**
  - Abstract class that serves as a generic file system representation
  - Note it's a class and not an Interface
- **Implemented in several flavors**
  - Ex. Distributed or Local

5

# FileSystem Implementations

- **Hadoop ships with multiple concrete implementations:**
  - org.apache.hadoop.fs.LocalFileSystem
    - Good old native file system using local disk(s)
  - org.apache.hadoop.hdfs.DistributedFileSystem
    - Hadoop Distributed File System (HDFS)
    - Will mostly focus on this implementation
  - org.apache.hadoop.hdfs.HttpFileSystem
    - Access HDFS in read-only mode over HTTP
  - org.apache.hadoop.fs.ftp.FTPFileSystem
    - File system on FTP server

6

# FileSystem Implementations

- **FileSystem concrete implementations**
  - Two options that are backed by Amazon S3 cloud
    - org.apache.hadoop.fs.s3.S3FileSystem
    - <http://wiki.apache.org/hadoop/AmazonS3>
  - org.apache.hadoop.fs.kfs.KosmosFileSystem
    - Backed by CloudStore
    - <http://code.google.com/p/kosmosfs>

7

# FileSystem Implementations

- Different use cases for different concrete implementations
- HDFS is the most common choice
  - org.apache.hadoop.hdfs.DistributedFileSystem

8

# SimpleLocalFs.java Example

```
public class SimpleLocalFs {
    public static void main(String[] args) throws Exception{

        Path path = new Path("/");
        if ( args.length == 1){
            path = new Path(args[0]);
        }

        Configuration conf = new Configuration();
        FileSystem fs = FileSystem.get(conf);

        List the files and directories
        under the provided path

        Acquire FileSystem
        Instance

        FileStatus [] files = fs.listStatus(path);
        for (FileStatus file : files ){
            System.out.println(file.getPath().getName());
        }

        Print each sub directory
        or file to the screen
    }
}
```

9

## FileSystem API: Path

- **Hadoop's Path object represents a file or a directory**
  - Not java.io.File which tightly couples to local filesystem
- **Path is really a URI on the FileSystem**
  - HDFS: hdfs://localhost/user/file1
  - Local: file:///user/file1
- **Examples:**
  - new Path("/test/file1.txt");
  - new Path("hdfs://localhost:9000/test/file1.txt");

10

## Hadoop's Configuration Object

- **Configuration object stores clients' and servers' configuration**
  - Very heavily used in Hadoop
    - HDFS, MapReduce, HBase, etc...
- **Simple key-value paradigm**
  - Wrapper for java.util.Properties class which itself is just a wrapper for java.util.Hashtable
- **Several construction options**
  - Configuration conf1 = new Configuration();
  - Configuration conf2 = new Configuration(conf1);
    - Configuration object conf2 is seeded with configurations of conf1 object

11

# Hadoop's Configuration Object

- **Getting properties is simple!**
- **Get the property**
  - `String nnName = conf.get("fs.default.name");`
    - returns null if property doesn't exist
- **Get the property and if doesn't exist return the provided default**
  - `String nnName = conf.get("fs.default.name", "hdfs://localhost:9000");`
- **There are also typed versions of these methods:**
  - `getBoolean`, `getInt`, `getFloat`, etc...
  - Example: `int prop = conf.getInt("file.size");`

12

# Hadoop's Configuration Object

- **Usually seeded via configuration files that are read from CLASSPATH (files like `conf/core-site.xml` and `conf/hdfs-site.xml`):**

```
Configuration conf = new Configuration();
conf.addResource(new Path(HADOOP_HOME + "/conf/core-site.xml"));
```

- **Must comply with the Configuration XML schema, ex:**

```
<?xml version="1.0"?>
<?xml-stylesheet type="text/xsl" href="configuration.xsl"?>
<configuration>
  <property>
    <name>fs.default.name</name>
    <value>hdfs://localhost:9000</value>
  </property>
</configuration>
```

13

# Hadoop Configuration Object

- **conf.addResource() - the parameters are either String or Path**
  - conf.addResource("hdfs-site.xml")
    - CLASSPATH is referenced when String parameter is provided
  - conf.addResource(new Path("/exact/location/file.xml")) :
    - Path points to the exact location on the local file system
- **By default Hadoop loads**
  - core-default.xml
    - Located at hadoop-common-X.X.X.jar/core-default.xml
  - core-site.xml
- **Looks for these configuration files on CLASSPATH**

14

# LoadConfigurations.java Example

```
public class LoadConfigurations {
    private final static String PROP_NAME = "fs.default.name";

    public static void main(String[] args) {
        Configuration conf = new Configuration();

        System.out.println("After construction: " +
            conf.get(PROP_NAME));

        conf.addResource(new Path(Vars.HADOOP_HOME +
            "/conf/core-site.xml"));
        System.out.println("After addResource: " +
            conf.get(PROP_NAME));

        conf.set(PROP_NAME, "hdfs://localhost:8111");
        System.out.println("After set: " + conf.get(PROP_NAME));
    }
}
```

1. Print the property with empty Configuration




2. Add properties from core-dsite.xml

3. Manually set the property

15

# Run LoadConfigurations

```
$ java -cp  
$PLAY_AREA/HadoopSamples.jar:$HADOOP_HOME/share/had  
common/hadoop-common-2.0.0-  
cdh4.0.0.jar:$HADOOP_HOME/share/hadoop/common/lib/*  
hdfs.LoadConfigurations
```

```
After construction: file:///   
After addResource: hdfs://localhost:9000   
After set: hdfs://localhost:8111 
```

1. Print the  
property with  
empty  
Configuration

2. Add  
properties from  
core-site.xml

3. Manually set  
the property

16

# FileSystem API

- Recall **FileSystem** is a generic abstract class used to interface with a file system
- **FileSystem** class also serves as a factory for concrete implementations, with the following methods
  - public static `FileSystem get(Configuration conf)`
    - Will use information from Configuration such as scheme and authority
    - Recall hadoop loads `conf/core-site.xml` by default
    - Core-site.xml typically sets `fs.default.name` property to something like `hdfs://localhost:8020`
      - `org.apache.hadoop.hdfs.DistributedFileSystem` will be used by default
      - Otherwise known as HDFS

17



# Simple List Example

```
public class SimpleLocalLs {
    public static void main(String[] args) throws Exception {
        Path path = new Path("/");
        if ( args.length == 1){
            path = new Path(args[0]);
        }
        Configuration conf = new Configuration();
        FileSystem fs = FileSystem.get(conf);
        FileStatus [] files = fs.listStatus(path);
        for (FileStatus file : files ){
            System.out.println(file.getPath().getName());
        }
    }
}
```

What happens when you run this code?

18

# Execute Simple List Example

- **The Answer is.... it depends**

```
$ java -cp
$PLAY_AREA/HadoopSamples.jar:$HADOOP_HOME/share/hadoop/commo
n/hadoop-common-2.0.0-
cdh4.0.0.jar:$HADOOP_HOME/share/hadoop/common/lib/*
hdfs.SimpleLs
lib
...
...
var
sbin
etc
```

← .Uses java command, not yarn  
← .core-site.xml and core-default.xml are not on the CLASSPATH  
← .properties are then NOT added to Configuration object  
← .Default FileSystem is loaded => local file system

```
$ yarn jar $PLAY_AREA/HadoopSamples.jar hdfs.SimpleLs
hbase
lost+found
test1
tmp
training
user
```

← .Yarn script will place core-default.xml and core-site.xml on the CLASSPATH  
← .Properties within those files added to Configuration object  
← .HDFS is utilized, since it was specified in core-site.xml

19

# Reading Data from HDFS

1. Create FileSystem
2. Open InputStream to a Path
3. Copy bytes using IOUtils
4. Close Stream

20

## 1: Create FileSystem

- **FileSystem fs = FileSystem.get(new Configuration());**
  - If you run with yarn command, DistributedFileSystem (HDFS) will be created
    - Utilizes fs.default.name property from configuration
    - Recall that Hadoop framework loads core-site.xml which sets property to hdfs (hdfs://localhost:8020)

21

## 2: Open Input Stream to a Path

```
...
InputStream input = null;
    try {
        input = fs.open(fileToRead);
    }
    ...
```

- **fs.open returns `org.apache.hadoop.fs.FSDataInputStream`**
  - Another FileSystem implementation will return their own custom implementation of `InputStream`
- **Opens stream with a default buffer of 4k**
- **If you want to provide your own buffer size use**
  - `fs.open(Path f, int bufferSize)`

22

## 3: Copy bytes using IOUtils

```
IOUtils.copyBytes(inputStream, outputStream, buffer);
```

- **Copy bytes from `InputStream` to `OutputStream`**
- **Hadoop's `IOUtils` makes the task simple**
  - `buffer` parameter specifies number of bytes to buffer at a time

23

## 4: Close Stream

```
...
} finally {
    IOUtils.closeStream(input);
...

```

- **Utilize IOUtils to avoid boiler plate code that catches IOException**

24

## ReadFile.java Example

```
public class ReadFile {
    public static void main(String[] args)
        throws IOException {
        Path fileToRead = new Path("/training/data/readMe.txt");
        FileSystem fs = FileSystem.get(new Configuration());

        InputStream input = null;
        try {
            input = fs.open(fileToRead);
            IOUtils.copyBytes(input, System.out, 4096);
        } finally {
            IOUtils.closeStream(input);
        }
    }
}
```

1: Open FileSystem  
2: Open Input Stream  
3: Copy from Input to Output Stream  
4: Close stream

```
$ yarn jar $PLAY_AREA/HadoopSamples.jar hdfs.ReadFile
Hello from readme.txt
```

25

## Reading Data - Seek

- **FileSystem.open** returns **FSDatInputStream**
  - Extension of `java.io.DataInputStream`
  - Supports random access and reading via interfaces:
    - `PositionedReadable` : read chunks of the stream
    - `Seekable` : seek to a particular position in the stream

26

## Seeking to a Position

- **FSDatInputStream** implements **Seekable** interface
  - `void seek(long pos)` throws `IOException`
    - Seek to a particular position in the file
    - Next read will begin at that position
    - If you attempt to seek past the file boundary `IOException` is emitted
    - Somewhat expensive operation – strive for streaming and not seeking
  - `long getPos()` throws `IOException`
    - Returns the current position/offset from the beginning of the stream/file

27

# SeekReadFile.java Example

```
public class SeekReadFile {
    public static void main(String[] args) throws IOException {
        Path fileToRead = new Path("/training/data/readMe.txt");
        FileSystem fs = FileSystem.get(new Configuration());
        FSDataInputStream input = null;
        try {
            input = fs.open(fileToRead);
            System.out.print("start position=" + input.getPos() + ":
            IOUtils.copyBytes(input, System.out, 4096, false);

            input.seek(11);
            System.out.print("start position=" + input.getPos() + ":
            IOUtils.copyBytes(input, System.out, 4096, false);

            input.seek(0);
            System.out.print("start position=" + input.getPos() + ":
            IOUtils.copyBytes(input, System.out, 4096, false);
        } finally {
            IOUtils.closeStream(input);
        }
    }
}
```

Start at position 0

Seek to position 11

Seek back to 0

28

# Run SeekReadFile Example

```
$ yarn jar $PLAY_AREA/HadoopSamples.jar hdfs.SeekReadFile
start position=0: Hello from readme.txt
start position=11: readme.txt
start position=0: Hello from readme.txt
```

# Write Data

1. Create FileSystem instance
2. Open OutputStream
  - FSDataOutputStream in this case
  - Open a stream directly to a Path from FileSystem
  - Creates all needed directories on the provided path
3. Copy data using IOUtils

30

# WriteToFile.java Example

```
public class WriteToFile {
    public static void main(String[] args) throws IOException {
        String textToWrite = "Hello HDFS! Elephants are awesome!\n";
        InputStream in = new BufferedInputStream(
            new ByteArrayInputStream(textToWrite.getBytes()));
        Path toHdfs = new Path("/training/playArea/writeMe.txt");
        Configuration conf = new Configuration();

        FileSystem fs = FileSystem.get(conf);

        FSDataOutputStream out = fs.create(toHdfs);

        IOUtils.copyBytes(in, out, conf);
    }
}
```

1: Create FileSystem instance

2: Open OutputStream

3: Copy Data

31

## Run WriteToFile

```
$ yarn jar $PLAY_AREA/HadoopSamples.jar hdfs.WriteToFile
$ hdfs dfs -cat /training/playArea/writeMe.txt
Hello HDFS! Elephants are awesome!
```

32

## FileSystem: Writing Data

- **Append to the end of the existing file**  
`fs.append(path)`
  - Optional support by concrete FileSystem
  - HDFS supports
- **No support for writing in the middle of the file**

33



## FileSystem: Writing Data

- **FileSystem's create and append methods have overloaded version that take callback interface to notify client of the progress**

```
FileSystem fs = FileSystem.get(conf);
FSDataOutputStream out = fs.create(toHdfs, new Progressable(){
    @Override
    public void progress() {
        System.out.print("..");
    }
});
```

Report progress to the screen

34

## Overwrite Flag

- **Recall FileSystem's create(Path) creates all the directories on the provided path**
  - create(new Path("/doesn't\_exist/doesn't\_exist/file/txt"))
  - can be dangerous, if you want to protect yourself then utilize the following overloaded method:

```
public FSDataOutputStream create(Path f, boolean overwrite)
```

Set to false to make sure you do not overwrite important data

35

# Overwrite Flag Example

```
Path toHdfs = new
    Path("/training/playArea/writeMe.txt");
FileSystem fs = FileSystem.get(conf);
FSDataOutputStream out = fs.create(toHdfs, false);
```

Set to false to make sure you do  
not overwrite important data

```
$ yarn jar $PLAY_AREA/HadoopSamples.jar hdfs.BadWriteToFile
Exception in thread "main"
org.apache.hadoop.ipc.RemoteException: java.io.IOException:
failed to create file
/training/playArea/anotherSubDir/writeMe.txt on client
127.0.0.1 either because the filename is invalid or the file
exists
...
...
```

Error indicates that the file already exists

36

# Copy/Move from and to Local File System

- **Higher level abstractions that allow you to copy and move from and to HDFS**
  - copyFromLocalFile
  - moveFromLocalFile
  - copyToLocalFile
  - moveToLocalFile

37

# Copy from Local to HDFS

```
FileSystem fs = FileSystem.get(new Configuration());
Path fromLocal = new
    Path("/home/hadoop/Training/exercises/sample_data/hamlet.txt");
Path toHdfs = new Path("/training/playArea/hamlet.txt");

fs.copyFromLocalFile(fromLocal, toHdfs);
```

Copy file from local file system to HDFS

```
$ hdfs dfs -ls /training/playArea/
```

Empty directory before copy

```
$ yarn jar $PLAY_AREA/HadoopSamples.jar hdfs.CopyToHdfs
```

```
$ hdfs dfs -ls /training/playArea/
Found 1 items
-rw-r--r-- hadoop supergroup /training/playArea/hamlet.txt
```

File was copied

38

# Delete Data

```
FileSystem.delete(Path path, Boolean recursive)
```

If recursive == true then non-empty directory will be deleted otherwise IOException is emitted

```
Path toDelete =
    new Path("/training/playArea/writeMe.txt");
boolean isDeleted = fs.delete(toDelete, false);
System.out.println("Deleted: " + isDeleted);
```

```
$ yarn jar $PLAY_AREA/HadoopSamples.jar hdfs.DeleteFile
Deleted: true
```

```
$ yarn jar $PLAY_AREA/HadoopSamples.jar hdfs.DeleteFile
Deleted: false
```

File was already deleted by previous run

39

## FileSystem: mkdirs

- **Create a directory - will create all the parent directories**

```
Configuration conf = new Configuration();
Path newDir = new Path("/training/playArea/newDir");
FileSystem fs = FileSystem.get(conf);
boolean created = fs.mkdirs(newDir);
System.out.println(created);
```

---

```
$ yarn jar $PLAY_AREA/HadoopSamples.jar hdfs.Mkdir
true
```

40

## FileSystem: listStatus

### Browse the FileSystem with listStatus() methods

```
Path path = new Path("/");
Configuration conf = new Configuration();
FileSystem fs = FileSystem.get(conf);
FileStatus [] files = fs.listStatus(path);
for (FileStatus file : files ) {
    System.out.println(file.getPath().getName());
}
```

---

```
$ yarn jar $PLAY_AREA/HadoopSamples.jar
hdfs.SimpleLs
training ← List files under "/" for HDFS
user
```

41

# LsWithPathFilter.java example

```
FileSystem fs = FileSystem.get(conf);
FileStatus [] files = fs.listStatus(path, new PathFilter() {
    @Override
    public boolean accept(Path path) {
        if (path.getName().equals("user")){
            return false;
        }
        return true;
    }
});
for (FileStatus file : files ){
    System.out.println(file.getPath().getName());
}
```

Do not show path whose name equals to "user"

Restrict result of listStatus() by supplying PathFilter object

42

# Run LsWithPathFilter Example

```
$ yarn jar $PLAY_AREA/HadoopSamples.jar hdfs.SimpleLs
training
user

$ yarn jar $PLAY_AREA/HadoopSamples.jar hdfs.LsWithPathFilter
training
```

43

## FileSystem: Globbing

- **FileSystem** supports file name pattern matching via `globStatus()` methods
- **Good** for traversing through a sub-set of files by using a pattern
- **Support** is similar to bash glob: `*`, `?`, etc...

44

## SimpleGlobbing.java

```
public class SimpleGlobbing {
    public static void main(String[] args)
        throws IOException {
        Path glob = new Path(args[0]);
        FileSystem fs = FileSystem.get(new Configuration());
        FileStatus [] files = fs.globStatus(glob);

        for (FileStatus file : files ) {
            System.out.println(file.getPath().getName());
        }
    }
}
```

Read glob from command line

Similar usage to listStatus method

45

# Run SimpleGlobbing

```
$ hdfs dfs -ls /training/data/glob/
```

```
Found 4 items
```

```
drwxr-xr-x - hadoop supergroup      0 2011-12-24 11:20 /training/data/glob/2007
drwxr-xr-x - hadoop supergroup      0 2011-12-24 11:20 /training/data/glob/2008
drwxr-xr-x - hadoop supergroup      0 2011-12-24 11:21 /training/data/glob/2010
drwxr-xr-x - hadoop supergroup      0 2011-12-24 11:21 /training/data/glob/2011
```

```
$ yarn jar $PLAY_AREA/HadoopSamples.jar hdfs.SimpleGlobbing /training/data/glob/201*
2010
2011
```

Usage of glob with \*



46

# FileSystem: Globbing

Glob	Explanation
?	Matches any single character
*	Matches zero or more characters
[abc]	Matches a single character from character set {a,b,c}.
[a-b]	Matches a single character from the character range {a...b}. Note that character a must be lexicographically less than or equal to character b.
[^a]	Matches a single character that is not from character set or range {a}. Note that the ^ character must occur immediately to the right of the opening bracket.
\c	Removes (escapes) any special meaning of character c.
{ab,cd}	Matches a string from the string set {ab, cd}
{ab,c{de,fh}}	Matches a string from the string set {ab, cde, cfh}

47

Source: FileSystem.globStatus API documentation

# FileSystem

- **There are several methods that return 'true' for success and 'false' for failure**
  - delete
  - rename
  - mkdirs
- **What to do if the method returns 'false'?**
  - Check Namenode's log
    - Located at \$HADOOP\_LOG\_DIR/

48

# BadRename.java

```
FileSystem fs = FileSystem.get(new Configuration());
Path source = new Path("/does/not/exist/file.txt");
Path nonExistentPath = new Path("/does/not/exist/file1.txt");
boolean result = fs.rename(source, nonExistentPath);
System.out.println("Rename: " + result);
```

```
$ yarn jar $PLAY_AREA/HadoopSamples.jar hdfs.BadRename
Rename: false
```



**Namenode's log at \$HADOOP\_HOME/logs/hadoop-hadoop-namenode-hadoop-laptop.log**

```
2011-12-25 01:18:54,684 WARN
org.apache.hadoop.hdfs.StateChange: DIR*
FSDirectory.unprotectedRenameTo: failed to rename
/does/not/exist/file.txt to /does/not/exist/file1.txt
because source does not exist
```

49





# Wrap-Up

**Customized Java EE Training:** <http://courses.coreservlets.com/>

Hadoop, Java, JSF 2, PrimeFaces, Servlets, JSP, Ajax, jQuery, Spring, Hibernate, RESTful Web Services, Android.  
Developed and taught by well-known author and developer. At public venues or onsite at *your* location.

## Summary

- **We learned about**
  - HDFS API
  - How to use Configuration class
  - How to read from HDFS
  - How to write to HDFS
  - How to browse HDFS



# Questions?

More info:

<http://www.coreservlets.com/hadoop-tutorial/> – Hadoop programming tutorial

<http://courses.coreservlets.com/hadoop-training.html> – Customized Hadoop training courses, at public venues or onsite at *your* organization

<http://courses.coreservlets.com/Course-Materials/java.html> – General Java programming tutorial

<http://www.coreservlets.com/java-8-tutorial/> – Java 8 tutorial

<http://www.coreservlets.com/JSF-Tutorial/jsf2/> – JSF 2.2 tutorial

<http://www.coreservlets.com/JSF-Tutorial/primefaces/> – PrimeFaces tutorial

<http://coreservlets.com/> – JSF 2, PrimeFaces, Java 7 or 8, Ajax, jQuery, Hadoop, RESTful Web Services, Android, HTML5, Spring, Hibernate, Servlets, JSP, GWT, and other Java EE training

**Customized Java EE Training: <http://courses.coreservlets.com/>**

Hadoop, Java, JSF 2, PrimeFaces, Servlets, JSP, Ajax, jQuery, Spring, Hibernate, RESTful Web Services, Android.

Developed and taught by well-known author and developer. At public venues or onsite at *your* location.