

# Overthere

Design and implementation of a Java remote file and execution framework

[xebialabs.com](http://xebialabs.com)

Vincent Partington  
CTO, XebiaLabs

# Speaker



- Been working with Java since 1996. Built first production Servlet in 1997.
- At Xebia since 2003, CTO of XebiaLabs since 2008.
- Interests: middleware, deployment automation, performance, security. Basically anything Java *and* hard!

# Agenda



- Why Overthere?
- Exploring the API.
- Designing, extending and testing Overthere.

# The problem

- For an *internet language* and an *enterprise language*, Java has pretty poor support for manipulating remote resources.
- Apache's HttpClient is way better than JDK's HTTP support.
- There is no facility to manipulate remote files (FTP, WebDAV, SFTP, SCP, CIFS, etc.)
- There is no facility to run commands on remote machines.

# The solution

- **Interfaces:** `java.io.File` and `java.lang.Process` should've been interfaces!
- **Factory:** And there should be a factory to create instances of them!!
- **SPI:** And there should be an SPI to hook in new implementations!!!

- `OverthereFile` - equivalent of `java.io.File`.
- `OverthereProcess` - equivalent of `java.lang.Process`.
- `Overthere` - the factory.
- `OverthereConnectionBuilder` - the SPI

# Create a connection (Unix)

```
ConnectionOptions options = new ConnectionOptions();
options.set(ADDRESS, "overthere");
options.set(USERNAME, "demo");
options.set(PASSWORD, "secret");
options.set(OPERATING_SYSTEM, UNIX);
options.set(CONNECTION_TYPE, SFTP);
OverthereConnection connection = Overthere.getConnection("ssh", options);
```

► Show ExecuteOnUnix

# Create a connection (Windows)

```
ConnectionOptions options = new ConnectionOptions();
options.set(ADDRESS, "wls-11g-win");
options.set(USERNAME, "Administrator");
options.set(PASSWORD, "xebialabs");
options.set(OPERATING_SYSTEM, WINDOWS);
options.set(CONNECTION_TYPE, TELNET);
OverthereConnection connection = Overthere.getConnection("cifs", options);
```

► Show ExecuteOnWindows

# Execute a command

```
OverthereConnection connection = Overthere.getConnection("ssh", options);  
try {  
    connection.execute(consoleHandler(), CmdLine.build("cat", "/etc/motd"));  
} finally {  
    connection.close();  
}
```

► [Show ExecuteOnUnix](#)

# Start a process

```
OverthereProcess process = connection.startProcess(CmdLine.build("cat", "/etc/motd"));
BufferedReader stdout = new BufferedReader(new InputStreamReader(process.getStdout()));
try {
    String line;
    while((line = stdout.readLine()) != null) {
        System.err.println(line);
    }
} finally {
    stdout.close();
}
int exitCode = process.waitFor();
System.err.println("Exit code: " + exitCode);
```

► [Show StartProcess](#)

# Read a file

```
OverthereFile motd = connection.getFile("/etc/motd");
BufferedReader r = new BufferedReader(new InputStreamReader(motd.getInputStream()));
try {
    String line;
    while((line = r.readLine()) != null) {
        System.err.println(line);
    }
} finally {
    r.close();
}
```

► [Show ReadFile](#)

# Write a file

```
OverthereFile motd = connection.getFile("/tmp/new-motd");
PrintWriter w = new PrintWriter(motd.getOutputStream());
try {
    w.println("An Overthere a day keeps the doctor away");
    w.println("Written on " + new java.util.Date());
} finally {
    w.close();
}
```

► Show WriteFile

# Get info about a file

```
OverthereFile motd = connection.getFile("/etc/motd");
System.out.println("Length          : " + motd.length());
System.out.println("Last modified : " + motd.lastModified());
System.out.println("Exists          : " + motd.exists());
System.out.println("Can read       : " + motd.canRead());
System.out.println("Can write     : " + motd.canWrite());
System.out.println("Can execute   : " + motd.canExecute());
```

► Show GetFileInfo

# Manipulate a file

```
connection.execute(consoleHandler(), CmdLine.build("cp", "/etc/motd", "/tmp/motd1"));
OverthereFile motd1 = connection.getFile("/tmp/motd1");
OverthereFile motd2 = connection.getFile("/tmp/motd2");

motd1.renameTo(motd2);

motd2.copyTo(motd1);

motd1.delete();
motd2.delete();
```

► [Show ManipulateFile](#)

# Manipulate a directory

```
connection.execute(consoleHandler(), CmdLine.build("cp", "-r", "/var/log/apt", "/tmp/logs1"));
OverthereFile logs1 = connection.getFile("/tmp/logs1");
OverthereFile logs2 = connection.getFile("/tmp/logs2");

logs1.renameTo(logs2);

logs2.copyTo(logs1);

logs1.deleteRecursively();
logs2.deleteRecursively();
```

► Show ManipulateDirectory

# Supported technologies



- SSH - for Unix and Windows:
  - SFTP - using SFTP for file transfer.
  - SCP - using SCP for file transfer.
  - SUDO - using SUDO for command execution (and SCP for file transfer).
  - INTERACTIVE\_SUDO - as SUDO but tried to answer sudo password prompts.
- CIFS - for Windows:
  - TELNET - using Telnet for command execution.
  - WINRM - using WinRM for command execution (Java implementation).
  - WINRS - using WinRM for command execution (native Windows implementation w/ `winrs.exe`).
- LOCAL - Layer on top of `java.io.File` and `java.lang.Process`.

# Advanced topics

- Designing Overthere.
- Extending Overthere.
- Testing Overthere.

# Designing Overthere (1/4)



- Make `OverthereFile` an interface or a subclass of `java.io.File`?

# Designing Overthere (2/4)



- Exceptions or booleans for `delete()`, `mkdir()`, `makedirs()` and `renameTo()`?
- Use checked or unchecked exceptions?

# Designing Overthere (3/4)

- Use JSch or SSH/J to handle SSH connections?

# Designing Overthere (4/4)



- Telnet or WinRM for remote execution on Windows?

# Extending Overthere - SPI



- At startup, all classes implementing `OverthereConnectionBuilder` are found.
- When a connection is created, all builders are invoked in turn to see if they can build a connection for that protocol and with those connection options.

```
public interface OverthereConnectionBuilder {  
    OverthereConnection connect();  
}
```

- ▶ Show `LocalConnection` and `SshConnectionBuilder`.

# Testing Overthere



- How to test?
  - Against target operating systems - Unix and Windows.
  - Portable - no VMware images.
  - From IDE (ad hoc) and from build systems.

# Subproject: `itest-support`



- Framework to allow for integration testing with virtual images.
- An `ItestHost` is a host against which to run tests.
- Created by invoking the `ItestHostFactory`.
- Depending on settings in `itest.properties` and system properties one of two things happens:
  - A pointer to an existing host (VMware image or manually launched EC2 instance) is returned.
  - An AMI is launched and when it has started up, a pointer to the EC2 is returned.
- At the end of the `itest`, any EC2 instances launched on-the-fly are terminated.

# Some examples

- Parametrized tests to run multiple connection types on one platform:
  - SCP, SFTP, etc. on Unix or
  - TELNET, WINRM\_HTTP, etc. on Windows.
- ▶ Show `OverthereOnUnixITest` and its parent classes.

# SSH tunneling

- Port 445 blocked by Amazon and many internet providers.
  - Solution: create SSH tunnels when starting itests. And don't forget to use the new host and port in your test code!
- ▶ `Show OverthereOnWindowsItest.setTypeAndOptions()`

- Provisioning an EC2 AMI can take a while, especially provisioning a Windows AMI.
- Might even want to run just one test while developing a new feature.
- ▶ Show `overthere/itest.properties` and `overthere/src/test/resources/itest.properties`.

# Open source



- Overthere 1.0 is part of Deployit 3.5 and up.
- Overthere will also be released as open source, under the BSD license.
- Check the Github repository:  
<https://github.com/xebialabs/overthere>
- Have fun and don't forget to mention any issues!
- Even better: send us a pull request with a fix. :-)

# More information



- Overthere: <https://github.com/xebialabs/overthere>
- Xebialabs: <http://www.xebialabs.com/>
- Integration testing: <http://bit.ly/ess5Xh>