

# **Boudoir**

**Istruzioni Operative Unit Test**

| <b>Re v.</b> | <b>Data</b> | <b>Descrizione revisione</b> | <b>Autore</b>   |
|--------------|-------------|------------------------------|-----------------|
| 1.0          | 12/02/2014  | Prima versione               | Stefano Adriani |

# Indice

|  |                    |
|--|--------------------|
| <a href="#">Introduzione.....</a>  | <a href="#">4</a>  |
| <a href="#">1.1Elenco delle funzionalità oggetto del test.....</a>         | <a href="#">4</a>  |
| <a href="#">2Procedure di test.....</a>                                    | <a href="#">5</a>  |
| <a href="#">2.1Duplice opzione di build.....</a>                           | <a href="#">5</a>  |
| <a href="#">2.2Comandi del Remote Manager.....</a>                         | <a href="#">7</a>  |
| <a href="#">2.3Duplice opzione di log.....</a>                             | <a href="#">8</a>  |
| <a href="#">2.4Analisi delle query (Benchmark SQL).....</a>                | <a href="#">9</a>  |
| <a href="#">2.5Diverse condizioni di uscita e analisi statistiche.....</a> | <a href="#">10</a> |
| <a href="#">2.6Mocked connections.....</a>                                 | <a href="#">10</a> |
| <a href="#">2.7Connessione via TNS name.....</a>                           | <a href="#">11</a> |
| <a href="#">2.8Test di carico e saturazione del pool.....</a>              | <a href="#">11</a> |

# Introduzione

Il presente documento descrive le Istruzioni Operative suggerite per svolgere i test funzionali della libreria Boudoir.

## 1.1 Elenco delle funzionalità oggetto del test

Il procedimento di test descritto nel presente documento mira a verificare le seguenti funzionalità:

1. Duplice opzione di build: via ANT e via IDE
2. Comandi del Remote manager
3. Duplice opzione di log: Log4j e POJO
4. Analisi puntuale delle query SQL via “benchmark” (invocabile da Remote Manager)
5. Diverse condizioni di uscita e analisi statistica (dati raccolti con scope application)
6. Connessione al database via “mocked connections”
7. Connessione al database via file TNS name
8. Test di carico del pool di connessioni al database

## 2 Procedure di test

### 2.1 Duplice opzione di build

#### Compilazione via IDE

La libreria può essere compilato normalmente via IDE (qui faremo riferimento ad Eclipse). Il risultato del build viene solitamente prodotto nella cartella definita nel file `.classpath` di Eclipse (chiave *output*) e può essere eseguito invocando la classe `SimpleClient` in uno di questi modi:

- 1) All'interno dell'IDE: cliccando sul pulsante *Run* (icona verde con triangolo bianco) oppure selezionando la voce *Run* dal menù *Run*
- 2) Da shell esterna (vedere esempio qui sotto)

**Requisiti:** per eseguire il build sotto Eclipse è necessario

- Importare l'intero progetto all'interno dell'IDE (in qualsiasi modo)
- Aggiungere le librerie nel CLASSPATH dell'IDE: *Proprietà del progetto – Java Build Path – Libraries – Add Jars* : selezionare tutti jar contenuti nella directory `lib` del progetto
- Per eseguire l'applicazione `SimpleClient` da Eclipse aprire il menù *Run – Run Configurations – Classpath*. Selezionare la voce *User Entries*, cliccare su *Advanced* e scegliere *Add Folders* per indicare all'IDE il percorso della directory `src/main/config`.

#### Esempio di invocazione del servizio su piattaforma Windows

```
SET CLASSPATH=C:/home/work/workspace/boudoir/classes;  
  
SET LIBRARY_HOME=C:/home/work/workspace/boudoir/lib  
SET LOG4J_HOME=%LIBRARY_HOME%/log4j-1.2.15.jar  
SET JDBC_HOME=%LIBRARY_HOME%/ojdbc6.jar  
REM SET JDBC_HOME=%LIBRARY_HOME%/postgresql-9.2-1003.jdbc4.jar  
REM SET JDBC_HOME=%LIBRARY_HOME%/mysql-connector-java-5.1.26-bin.jar  
  
SET CLASSPATH=%CLASSPATH%;%LOG4J_HOME%;%JDBC_HOME%  
  
java -cp %CLASSPATH% adriani.boudoir.test.SimpleClient %1 %2 %3
```

Figura 2.1

Nell'esempio qui sopra la variabile di sistema `LIBRARY_HOME` punta esattamente alla stessa `lib` utilizzata da Eclipse: in questo modo l'esecuzione via shell o via IDE è del tutto equivalente.

Il build via IDE si appoggia sulle seguenti risorse:

- `bd_configuration.properties`: configurazione generale del servizio
- `bd_log4j.properties`: configurazione di Log4j

collocate all'interno della directory `resources`.

**Nota:** per evitare che Eclipse interpreti i percorsi del tipo `src/main/java` come nome del package verificare che nel menù *Java Build Path – Source* siano presenti tutte le cartelle dei sorgenti, come in figura 2.2. Altrimenti queste vanno aggiunte con l'opzione *Add Folder*.

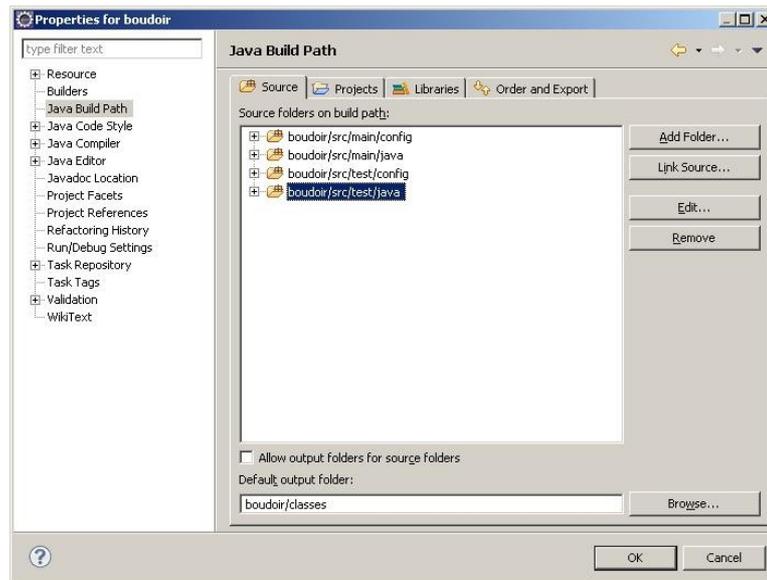


Figura 2.2 – Configurazione delle cartelle sorgenti in Eclipse

### Compilazione via ANT

La libreria può essere compilata normalmente via ANT, in alternativa alla compilazione tramite l'IDE.

**Nota:** ANT è lo strumento preferenziale per esportare la libreria completa nel formato JAR, pronta per l'utilizzo da parte di altri progetti. Questo perché ANT permette di configurare le modalità di composizione del JAR in maniera più flessibile di Eclipse.

**Requisiti:** per eseguire il build sotto ANT è necessario

- Far puntare la variabile JAVA\_HOME ad una JDK installata sulla macchina locale. In alternativa, se ANT viene usato solamente all'interno dell'IDE è sufficiente indicare una JDK all'IDE: *Proprietà di Eclipse – Java – Installed JREs – Add*: selezionare una JDK e spuntarla come unica opzione di lavoro
- Aprire la View ANT di Eclipse e aggiungere il file `build.xml`

**Requisiti:** per lanciare la versione della libreria prodotta dal build di ANT è possibile eseguire la classe `SimpleClient` da shell, previa verifica dei seguenti requisiti:

- Far puntare la variabile JAVA\_HOME ad una JDK installata sulla macchina locale
- Definire lo stesso CLASSPATH di figura 2.1, con la differenza che classi, risorse e file di configurazione vanno cercati nella cartella `build` anziché `classes` (vedi esempio di figura 2.3)

#### Esempio di esecuzione del servizio compilato con ANT

```
SET CLASSPATH=C:/home/work/workspace/boudoir/build;
C:/home/work/workspace/boudoir/build/config
java -cp %CLASSPATH% adriani.boudoir.test.SimpleClient %1 %2 %3
```

Figura 2.3

### Test della funzionalità

A seguire le istruzioni operative per verificare il corretto funzionamento delle procedure di build:

### Testare la modalità di build con ANT o Eclipse

- 1) Modificare almeno una delle proprietà del file `bd_configuration.properties` e verificare l'esito della modifica
- 2) Modificare almeno una delle proprietà del file `bd_log4j.properties` e verificare l'esito della modifica

Figura 2.4

## 2.2 Comandi del Remote Manager

### Test della funzionalità

Qui sotto è riportata la tabella di verifica dei comandi del Remote Manager. Per collaudare la funzionalità:

- Eseguire la classe `SimpleClient` inviando un comando, che deve essere preceduto dal prefisso definito dalla proprietà `manager.prefix` del file `bd_configuration.properties`.

### Esempio di test del Remote Manager

```
# Nell'ipotesi di avere manager.prefix = c, per invocare il comando 1 eseguire:  
run.bat c1
```

Figura 2.5

e verificare che i comandi abbiano i seguenti esiti

| CODICE | COMANDO  |
|--------|--|
| 1      | Stampa nel file di LOG la <b>configurazione</b> attuale del web service                  |
| 2      | Stampa nel file di LOG alcuni dati sull'ambiente di esecuzione <b>run-time</b>           |
| 3      | Stampa nel file di LOG la lista dei <b>threads</b> visibili ad Axis2                     |
| 4      | Stampa nel file di LOG alcuni dati sulle connessioni ai <b>database</b>                  |
| 5      | Stampa nel file di LOG i risultati della <b>analisi</b> e statistiche di funzionamento   |
| 6      | Setta on/off la modalità di <b>Analisi</b> (parametro <code>service.analyze</code> )     |
| 7      | Setta on/off la modalità <b>Verbose</b> (parametro <code>service.verbose</code> )        |
| 8      | Setta on/off la modalità <b>Debug</b> (parametro <code>service.debug</code> )            |
| 9      | Setta on/off la modalità <b>Remote Manager</b> (parametro <code>service.manager</code> ) |
| 10     | Setta on/off la modalità <b>Benchmark</b> (parametro <code>service.benchmark</code> )    |
| 11     | Stampa nel file di LOG lo status del <b>pool JDBC</b> che gestisce le connessioni ai DB  |

### Note

- 1) Inviando il comando 9 (phone number: 0000009) il sistema di Remote Manager viene disattivato.
- 2) Il collaudo specifico delle funzionalità di Analisi e Benchmark è illustrato nei prossimi paragrafi.
- 3) Un collaudo più esaustivo dello status del **pool JDBC** è delegato ai testi di carico (vedi "2.8 Test di carico" a pagina 11).

## 2.3 Duplice opzione di log

Editare il file `bd_configuration.properties` e identificare il seguente flag:

```
logger.type = LOG4J
```

il quale può assumere due valori: `LOG4J` e `JAVA`. Il primo valore seleziona il log via la libreria `Log4J`, il secondo valore seleziona il sistema di log normale Java (`POJO`).

### Test della funzionalità

#### Test del log via Log4j

- 1) Impostare il valore `LOG4J` nel file `bd_configuration.properties`
- 2) Impostare a `NO` tutti i flag di tipo `service` nel file `bd_configuration.properties` (cioè `service.verbose`, `service.debug` ecc.)
- 3) Verificare che le informazioni di log vengono prodotte come indicato dal file `bd_log4j.properties`. In particolare modificare le configurazioni del file `bd_log4j.properties` per verificare:

*Target:* cambiare le etichette `stdout`, `myfile` delle proprietà *logger*

*Verbosità:* cambiare i livelli `DEBUG`, `INFO`, `WARN`, `ERROR`, `FATAL` delle proprietà *logger*

*Appender:* cambiare il tipo di *Appender*: `FileAppender` oppure `DailyRollingFileAppender` (\*)

#### Test del log via Java

- 1) Impostare il valore `JAVA` nel file `bd_configuration.properties`
- 2) Impostare a `NO` tutti i flag di tipo `service` nel file `bd_configuration.properties` (cioè `service.verbose`, `service.debug` ecc.)
- 3) Verificare che le informazioni di log vengono prodotte come indicato dalle seguenti proprietà (dello stesso file):

`logger.level`: livello di debug (`INFO`, `WARNING` o `SEVERE`)

`logger.rotate`: minuti di rotazione automatica del file di log (\*)

`logger.dir`: nome della directory di log (percorso relativo)

`path.windows`: percorso della directory di log su Windows (path assoluto iniziale)

`path.linux`: percorso della directory di log su Linux (path assoluto iniziale)

(\*) Per il test di rotazione automatica via `Log4j` è preferibile utilizzare la libreria nel contesto di un altro progetto, in modo da verificare il cambiamento del sistema di log durante il cambiamento di data.

Un modo di verificare la rotazione automatica in modalità stand alone (esecuzione della libreria via `SimpleClient`) consiste nel ritardare l'esecuzione della `SimpleClient` mediante il metodo `keepAlive`, eseguibile da linea di comando con l'opzione: `-t 4 <secondi>`.

Nel caso di log con opzione `Java` si consiglia di impostare il tempo di rotazione a 1 minuto, e di invocare il metodo `keepAlive` con un ritardo di almeno 1-2 minuti, in modo da garantire l'evento di rotazione.

Figura 2.6

## 2.4 Analisi delle query (Benchmark SQL)

Il sistema di benchmark permette di loggare su file i tempi di esecuzione dei singolo statement SQL.

### Test della funzionalità

#### 1° test: attivazione di benchmark standard

- 1) Impostare il sistema di log su LOG4J nel file `bd_configuration.properties`
- 2) Attivare il sistema di benchmark nel file di configurazione:  
`library.benchmark = YES`
- 3) Invocare tutte le query di esempio implementate nella `SimpleClient`, lanciando l'applicazione con l'opzione `-q`. Esempio: `run.bat -q 0` (per lanciare la prima query).
- 4) Verificare che i risultati dell'analisi vengano prodotti all'interno del file identificato dal path:  
  
`logger.dir`: nome della directory di log (percorso relativo)  
`path.windows`: percorso della directory di log su Windows (path assoluto iniziale)  
`path.linux`: percorso della directory di log su Linux (path assoluto iniziale)  
  
Il file dovrebbe avere un nome del tipo `analysis_21_12_13.txt` (giorno 21, ore 12:13). Il file è contenuto all'interno della directory del mese corrente (esempio: *maggio* corrisponde alla directory 5).
- 5) Controllare che la rotazione del file avvenga in base al flag `logger.rotate` del file di configurazione. Per il test di rotazione automatica vedere la nota (\*) del test di log (paragrafo 2.3). In questo caso può essere utile invocare la `ServiceClient` con l'opzione `-qt`.
- 6) Verificare che il normale log sia distinto dal sistema di benchmark, ovvero corrisponda a quello definito dalla configurazione di Log4j

#### 2° test: attivazione di benchmark con log condiviso

- 1) Impostare il sistema di log su JAVA nel file `bd_configuration.properties`
- 2) Eseguire gli stessi test del caso precedente, con la differenza che ora si dovrebbe avere:
  - File di log e file di benchmark nella stessa directory (la stessa del caso precedente)
  - File di log e file di benchmark in rotazione con lo stesso periodo
  - Nessun file prodotto da Log4j

Figura 2.7

## 2.5 Diverse condizioni di uscita e analisi statistiche

Il presente test comprende due funzionalità distinte: verifica della gestione degli errori e conformità degli stessi coi dati raccolti a run-time (analisi non persistente). La classe `SimpleLogic` della *Simple Application* (pacchetto di test) permette di testare automaticamente le diverse condizioni d'uscita.

### Test della funzionalità

#### Verifica delle diverse condizioni di uscita

E' possibile eseguire la classe `SimpleClient` con l'opzione `-a (all)` per lanciare il test automatico che emula tutte le diverse condizioni d'uscita, ovvero:

- NO RESULT: conteggiata come "Calls to the service that provided no result"
- LOGIC ERROR: conteggiata come "Calls interrupted by a logic error"
- MACRO ERROR: conteggiata come "Calls interrupted by a macro error"
- DATABASE ERROR: conteggiata come "Calls stopped because the database was missing"
- VALIDATION ERROR: conteggiata come "Calls completed with a validation error"
- COMMAND CODE: conteggiata come "Calls due to a command code"
- NO ERROR: conteggiata come "Calls completed returning valid data"

Dopo l'esecuzione del test verrà stampato a video (o nel file di log) le statistiche raccolte dalla classe `ServiceAnalyzer`, che dovrebbero avere il seguente esito:

- Tutti i conteggi pari ad 1, ad eccezione di
- Chiamate senza risultato (NO RESULT) pari a 2

Figura 2.8

## 2.6 Mocked connections

Quando la modalità `mocked connections` è attiva i `PreparedStatement JDBC` (ma non gli altri `statement`) vengono "cachati" all'interno di un'apposita tabella del pool. Attivando le `mocked connections` è possibile verificare se i tempi di esecuzione degli `statement SQL` sono già ottimizzati (dal driver o dal database) o se possono essere ottimizzati ulteriormente (almeno per quanto riguarda le performances).

### Test della funzionalità

#### Esecuzione di una query SQL via mocked connections

- 1) Lanciare il metodo automatico di test `SimpleClient -t 7`, passando come argomento il parametro "y" (yes), cioè: `SimpleClient -t 7 y`. L'applicazione eseguirà un paio di chiamate alla `SimpleLogic` usando le `mocked connections`.
- 2) Verificare la presenza dei seguenti output nel log su `STDOUT`:
  - `MockConnection - Cached new pstmt: <riferimento oggetto>`
  - `MockConnection - Using cached pstmt: <riferimento oggetto (uguale al precedente)>`
  - `ServiceDAO - testPreparedStatement: closed pstmt? false`
- 3) Eseguire un'altra chiamata senza passare il parametro "y" e verificare la presenza dei seguenti output nel log su `STDOUT`:
  - `MockConnection - Using temporary pstmt: <riferimento 1° oggetto>`
  - `ServiceDAO - testPreparedStatement: closed pstmt? True`
  - `MockConnection - Using temporary pstmt: <riferimento 2° oggetto (diverso dal precedente)>`

Figura 2.9

## 2.7 Connessione via TNS name

Se viene trovato un file del tipo `tnsnames.ora` nel percorso indicato dalla configurazione del servizio tale file ha la precedenza sugli altri parametri di configurazione. Se tale file non viene trovato la connessione al database avviene secondo i normali parametri.

### Test della funzionalità

#### Verifica della connessione via TNS name

- 1) Identificare o creare un file `tnsnames.ora` che contenga le credenziali di autenticazione al database
- 2) Specificare il percorso del file `tnsnames.ora` nella proprietà `oracle.tnsnames` del file `configuration.properties` (o equivalente)
- 3) Inserire un indirizzo **errato** nel parametro `db_01.host` di configurazione, invocare la libreria e verificare che le connessioni vengano aperte correttamente. In particolare si dovrebbe avere un messaggio del tipo  

```
INFO jdbcpool <nome database> - Loaded TNS oracle configuration for <nome database>
```
- 4) Specificare un percorso **errato** nella proprietà `oracle.tnsnames` di configurazione, invocare il servizio e verificare che il database **non** venga raggiunto
- 5) Ripristinare l'indirizzo **corretto** nel parametro `db_01.host` di configurazione, invocare il servizio e verificare che il database venga di nuovo raggiunto

Figura 2.10

## 2.8 Test di carico e saturazione del pool

E' possibile lanciare un test automatico del pool di connessioni al database invocando il comando

```
SimpleClient -t 6
```

tale metodo invoca il metodo principale della SimpleClient lanciando diversi thread concorrenti, in modo da verificare che il pool vada in saturazione quando sottoposto ad un numero di richieste maggiori della dimensione del pool. Per questo motivo le modalità e l'esito del test dipendono fortemente dalla configurazione del pool, esplicitata nel file

```
test_config.properties
```

La configurazione consigliata è la seguente:

- `pool.maximum` = 5
- `pool.timeout` = 5
- `pool.mocked` = NO

L'applicazione di test provvederà a lanciare 6 thread concorrenti (uno in più della capacità del pool) arrestando ogni invocazione per circa un secondo, allo scopo di assicurare che ogni connessione rimanga in stato di *lock*, forzando così la saturazione del pool. Per dettagli vedere il metodo

```
SimpleClient.testLoadPool(int num, int sec)
```

### Test di carico del pool

- 1) Lanciare l'applicazione di test con l'opzione "test 6", ovvero:

```
SimpleClient -t 6
```

- 2) Verificare che durante l'esecuzione l'output su STDOUT si arresti per tre volte, con il seguente risultato:

- *Primo arresto*: tutti gli oggetti del pool sono in stato `Locked`
- *Secondo arresto*: tutti gli oggetti del pool sono in stato `Unlocked`
- *Terzo arresto*: il pool è stato svuotato

- 3) **Opzionale**: volendo è possibile verificare la corretta gestione della saturazione del pool controllando il file di log. A tal fine occorre identificare gli errori (etichetta `ERROR`) nel file di log e verificare che le chiamate di quel thread (identificato dalla sigla tra parentesi quadre) siano sempre precedute da un messaggio di "pool esaurito", ad esempio

```
2013-05-23 11:22:40,478 [http-8080-2] adriani.boudoir.db
ERROR jdbcpool 1544/dullt2 - Reached maximum pool capacity: 5
...
2013-05-23 11:22:40,494 [http-8080-2] adriani.boudoir.test
ERROR SimpleDAO - failed preparation of the prepared statement
```

Figura 2.11

Come ulteriore verificare è possibile monitorare il numero di connessioni lato backend attraverso una query del tipo (esempio per Oracle):

```
SELECT status, TO_CHAR(logon_time, 'yyyy-mm-dd') call_time, last_call_et, COUNT(*)
total
FROM v$session
WHERE username = 'ART'
GROUP BY status, TO_CHAR(logon_time, 'yyyy-mm-dd'), last_call_et
ORDER BY total desc ;
```

In tal modo si dovrebbe evidenziare la differenza tra lo stato del pool "pieno" e "vuoto" (una delle righe ritornate dovrebbe cambiare subito dopo lo svuotamento del pool).