

Stresstest

Stresstest Usage Guide

Technische Dokumentation

8. August 2005

Inhaltsverzeichnis

1	Zielsetzung.....	3
2	Funktion des Stresstests.....	3
2.1	Einführung.....	3
2.2	Testablauf für Datenbankabfragen.....	3
2.3	Testablauf für Java-Programme	3
3	Installation Stresstest	4
3.1	Systemvoraussetzungen.....	4
3.1.1	Java Runtime Environment (J2RE 1.4.1).....	4
3.1.2	Datenbank JDBC Client.....	4
3.2	Ausgelieferte Dateien	4
4	Konfiguration des Stresstests.....	5
4.1	Erstellen der Eingabedateien für Datenbanktests.....	5
4.2	Starten des Stresstests	5
4.2.1	Standardwerte	6
4.2.2	Classpath	6
4.2.3	Beispiel eines Startscripts für Unix	7
4.2.4	Beispiel eines Startscripts für Windows	7
5	Java Programm-Schnittstelle für Stresstests	8
5.1	Beispiel für die Verwendung der Programmschnittstelle.....	8
5.2	Methoden des Objekts StresstestSession.....	9
5.2.1	executeChain().....	9
5.2.2	execute(methodName)	9
5.2.3	execute(methodName, fileName)	9
5.2.4	execute(url)	9
5.2.5	get_current_file()	10
5.2.6	get_param()	10
5.2.7	get_execute()	10
5.2.8	get_stresstest()	10
5.2.9	get_connection()	10
5.2.10	get_thread_index()	11
5.2.11	get_session_number().....	11
5.2.12	get_current_value()	11
5.2.13	get_next_value().....	11
5.3	Methoden des Objekts StresstestResponse	12
5.3.1	get_response_code().....	12
5.3.2	get_response_message().....	12
6	Protokolle.....	13
6.1	Standard-Protokoll.....	13
6.2	Thread-Protokoll.....	13
6.3	Unterschied zwischen Thread-Protokoll und Standard-Protokoll	13
7	Ergänzende Tools	14
7.1	Stresstest Import	14
7.1.1	Starten des Stresstest Imports	14
7.2	Stresstest Report.....	15
7.2.1	Starten des Stresstest Reports.....	15

1 Zielsetzung

Der *Usage Guide* beschreibt die Installation, Konfiguration und den Betrieb der Software für Lasttests. Das Handbuch richtet sich an Entwickler, die mit der Installation und dem Betrieb der Software-Komponente beauftragt sind.

Dieses Dokument beschreibt die Installation und Verwendung der Stresstest Software für die Betriebssysteme Unix und Windows.

2 Funktion des Stresstests

2.1 Einführung

Der Stresstest ist ein Lasttestverfahren, mit dessen Hilfe datenbankintensive Applikationen und individuelle Java-Programme getestet werden können. Die Software wird als Java-Kommandozeilentool und über eine Programmschnittstelle betrieben. Der Stresstest ist in der Lage die Zugriffe mehrerer Benutzer auf eine Datenbank oder ein Java-Programm zu simulieren. Dies geschieht mittels Java-Threads, wobei jeder Thread die im normalen Betrieb vorkommenden Zugriffe eines Benutzers simuliert. Mit dem Stresstest kann ermittelt werden wie viele Nutzer bei der aktuellen Serverkonfiguration (Hardware und Software) simultan agieren können und welche Antwortzeiten bei dieser Auslastung entstehen.

Das Lasttestverfahren stellt keinen Ersatz für einen Funktionstest dar, d.h. vor Einsatz des Lasttestverfahrens sollten die zu testenden Datenbankabfragen und Programme von den Entwicklern auf funktionale Performance geprüft werden.

2.2 Testablauf für Datenbankabfragen

Beim Start des Stresstests können mehrere Eingabedateien angegeben werden. In diesen Eingabedateien werden die SQL-Statements abgelegt, die getestet werden sollen. Jede Datei steht hierbei für einen Block von Anweisungen, die die Anwendung ausführt, um z.B. einen bestimmten Datensatz zu finden. Variabel ist die Anzahl der Threads, die die zeitgleich auf der Datenbank arbeitenden Benutzer simulieren, wobei jeder Thread seine eigene Verbindung zur Datenbank aufbaut. Jeder Thread wiederholt die Abarbeitung sämtlicher Eingabedateien solange, bis die Dauer verstrichen ist, die für den Stresstest vereinbart wurde. Die Zusammenfassung am Ende des Stresstests gibt Auskunft darüber, wie oft jede Datei im Zeitfenster von allen Threads jeweils abgearbeitet wurde, sowie die schnellste, die langsamste und die durchschnittliche Antwortzeit der Verarbeitung pro Eingabedatei.

2.3 Testablauf für Java-Programme

Beim Start des Stresstests kann eine individuelle Klasse angegeben werden, die getestet wird. Diese Klasse implementiert ein Interface des Stresstest APIs und testet einzelne Methoden einer individuellen Applikation.

Die Anzahl der Threads, die zeitgleich die Methoden der Applikation ausführen, ist konfigurierbar. Dieser Test zielt zum einen darauf, festzustellen, mit wie vielen simultan agierenden Benutzern welche Antwortzeiten von der Applikation auf einem Server-System zu erwarten sind. Zum anderen zeigt der Test, ob die getestete Applikation faktisch threadfähig ist oder nicht. Viele Java-Programme verhalten sich bei paralleler Ausführung mit mehreren Threads anders als im Individualbetrieb. Jeder Thread wiederholt die Abarbeitung der Methoden einer Applikation solange, bis die Dauer verstrichen ist, die für den Stresstest vereinbart wurde. Die Zusammenfassung am Ende des Stresstests gibt Auskunft darüber, wie oft jede Methode aufgerufen wurde und protokolliert evtl. aufgetretene Fehler.

3 Installation Stresstest

3.1 Systemvoraussetzungen

Stresstest wird als Java-Kommandozeilentool ausgeliefert, d.h. als .jar-Archiv, und kann auf den Betriebssystemen Windows und Unix ausgeführt werden. Folgenden Software-Komponenten werden für den Betrieb vorausgesetzt:

3.1.1 Java Runtime Environment (J2RE 1.4.1)

Das Java Runtime Environment muss korrekt installiert sein und die Umgebungsvariable JAVA_HOME auf das Installationsverzeichnis verweisen.

3.1.2 Datenbank JDBC Client

Ein funktionsfähiger JDBC-Treiber für MySQL, MS SQL Server oder Oracle kompatibel zu Java JRE 1.4.1 wird benötigt, wenn entweder ein Datenbanktest durchgeführt wird oder die Testergebnisse mit Hilfe der ergänzenden Tools *Stresstest Import* und *Stresstest Report* ausgewertet werden sollen. Für die Durchführung von Stresstests mit datenbankfreien Applikationen wird kein JDBC-Treiber benötigt.

3.2 Ausgelieferte Dateien

Folgende Dateien sind im Lieferumfang enthalten:

/stresstest	Ordner für die Stresstest Software
stresstest.sh	Stresstest Startscript für Unix
stresstest.cmd	Stresstest Startscript für Windows
stresstest_import.sh	Stresstest Import Startscript für Unix
stresstest_import.cmd	Stresstest Import Startscript für Windows
stresstest_report.sh	Stresstest Report Startscript für Unix
stresstest_report.cmd	Stresstest Report Startscript für Windows
./config	Ordner für Konfigurationsdatei
sos_settings.ini	Beispiel einer Konfigurationsdatei für datenbankbasierte Stresstests
./lib	Ordner für Java-Archive
sos.connection.jar	Java-Klassen der SOS GmbH für Datenbankzugriffe
sos.stresstest.jar	Java-Klassen der SOS GmbH zur Verarbeitung von Einstellungen
sos.util.jar	Java-Klassen der SOS GmbH für Hilfsfunktionen

Folgende Dateien sind nicht im Lieferumfang enthalten:

msbase.jar	Java-Klassen, die vom MS SQL Server JDBC-Treiber genutzt werden
mssqlserver.jar	Java-Klassen für MS SQL Server JDBC-Treiber
msutil.jar	Java-Klassen mit MS SQL Server Hilfsfunktionen
mysql*.jar	Java-Klassen für MySQL müssen passend zur verwendeten

Datenbankversion von <http://www.mysql.com> geladen werden

ojdbc14.jar

Java-Klassen des Oracle JDBC-Treibers

4 Konfiguration des Stresstests

4.1 Erstellen der Eingabedateien für Datenbanktests

Beim Erstellen der Eingabedateien, ist es möglich SQL-Anweisungen über mehrere Zeilen hinweg zu schreiben, was der besseren Lesbarkeit der Anweisungen dient. Abgeschlossene SQL-Statements werden dadurch voneinander getrennt, dass sich zwischen ihnen mindestens eine Leerzeile befindet.

Kommentare sind möglich, jede Kommentarzeile muss mit `#` oder `;` beginnen.

MySQL: Jede Eingabedatei sollte mit `SET SESSION query_cache_type = OFF` beginnen. Damit wird der Query Cache der Datenbank für den Test abgeschaltet, da die Datenbank sonst die Ergebnisse der Abfragen zwischenspeichert und die Werte dadurch verfälscht werden, da es unwahrscheinlich ist, dass die Datenbank im normalen Betrieb zum Zwischenspeichern der Abfragen in der Lage wäre.

Wenn das Einfügen oder das Aktualisieren von Datensätzen innerhalb von Transaktionen getestet werden soll, empfiehlt es sich logisch zusammengehörende SQL-Anweisungen auf unterschiedlichen Datensätzen auszuführen, um realistischere Bedingungen für den Test zu schaffen, da es nicht wahrscheinlich ist, dass z.B. in einem Test, mit einer hohen Anzahl von Threads, alle zeitgleich auf einen einzigen Datensatz zugreifen. Dazu gibt es die Möglichkeit in die Anweisungen Platzhalter für numerische Felder der Datensätze einzusetzen, die im Stresstest durch die Optionen `-start` und `-stop` gesteuert werden können.

Beispiel:

```
SELECT "TEMPLATE", "TITLE", "QUEUE", "LANG", "USER", "STATUS_DATE", "STATUS",
"STATUS_TEXT", "INBOUND_TYPE", "INBOUND_FILENAME", "FILENAME", "JOB_ID", "MODEL", "TASK",
"REFERENCE" FROM LF_LETTERS WHERE "LETTER"=<%=nextval%>

UPDATE LF_LETTERS SET "TEMPLATE"=5, "TITLE"='Dokumentauftrag Nr. 4209, externe Auftrags-
Nr. 4711', "QUEUE"=0, "LANG"=0, "USER"=3, "STATUS_DATE"=NOW(), "STATUS"=0,
"STATUS_TEXT"='ENQUEUED', "INBOUND_TYPE"=6, "INBOUND_FILENAME"='', "FILENAME"='',
"JOB_ID"=9480, "MODEL"=3, "TASK"=47, "STARTDATE"='2004-08-26 14:38:39',
"REFERENCE"='2190;4711' WHERE "LETTER"=<%=curval%>
```

Der Platzhalter `<%=nextval%>` veranlasst bei seiner Auswertung den Stresstest dazu, den internen Zähler zu inkrementieren. Nachfolgende SQL-Anweisungen, die auf dem/den gleichen Datensatz/Datensätzen zugreifen sollen, können den Platzhalter `<%=curval%>` verwenden. Im obigen Beispiel wird zuerst ein Datensatz ausgewählt, an dem anschließend die Aktualisierung vorgenommen werden soll.

4.2 Starten des Stresstests

Gestartet wird der Stresstest bspw. mit folgenden Argumenten:

```
java -classpath=./lib/mysql-connector-java-3.0.15-ga-bin.jar;
./lib/sos.connection.jar;./lib/sos.util.jar;./lib/sos.stresstest.jar
sos.stresstest.Stresstest
    -config=config/sos_settings.ini
    -output=stresstest.log
    -threads=25
    -time=30
    -input=file1.sql+file2.sql
    -start=1
    -stop=100
```

Beachten Sie bitte, dass die Archive für `-classpath` im obigen Beispiel für Windows mit `;` getrennt sind, unter Unix werden Archive durch `.` getrennt. Die Argumente beim Programmstart haben folgende Bedeutung:

`-config` Datei, in der die Zugangsdaten zur Datenbank enthalten sind
`-classpath` Java-Classpath mit den Java-Archiven der Software

-class	Name der Test-Klasse, die Standard-Klasse ist <code>sos.stresstest.SOSStresstestSession</code> , es können individuelle Klassen für Stresstests entwickelt werden
-output	Name der Protokolldatei für Ausgaben des Stresstests
-append-output	vereinbart, dass das Protokoll für mehrere Lasttest-Abläufe fortgeschrieben oder überschrieben wird, die Voreinstellung lautet <code>true</code> , d.h. das Protokoll wird fortgeschrieben
-threads	Anzahl der simulierten, gleichzeitigen Benutzer im Stresstest
-time	Dauer des Stresstests in Sekunden
-param	frei wählbarer Parameterwert zur Verwendung mit der Stresstest API
-execute	Auswahl einer Testmethode zur Ausführung in der Methode <code>executeC</code> <code>hain()</code> ; mehrere Methoden können durch <code>+</code> getrennt angegeben werden
-input	Eingabedatei bzw. Liste von Eingabedateien (durch <code>+</code> verbunden)
-start	Startzähler, der bei spezifischen SQL-Anweisungen inkrementiert wird
-stop	Wert, bei dessen Erreichen der Zähler wieder auf den Startwert zurückgesetzt wird
-v	(verbosity) Debug-Level für die Threads einstellen. Mögliche Werte: 0 (keine Debug-Ausgabe), 1-9 (Debug-Level)

4.2.1 Standardwerte

Nicht alle Optionen müssen angegeben werden. Optionen mit Standardwerten sind:

-config	Standardwert: leere Zeichenkette
-log-dir	Standardwert: <code>./logs</code>
-output	Standardwert: <code>stdout</code>
-append-output	hängt Protokollausgaben an, Standardwert: <code>true</code>
-threads	Anzahl der simultanen Threads, Standardwert: 1
-time	Dauer des Tests in Sekunden, Standardwert: 60s
-class	Standardwert: <code>SOSStresstestSession</code>
-start	Standardwert: 0
-stop	Standardwert: 0
-v	Standardwert: 0

Es muss entweder der Parameter `-class` oder `-input` angegeben werden; es können beide angegeben werden, wobei der `-class` Parameter nur von Bedeutung ist, wenn eine individuelle Klasse für den Stresstest entwickelt wurde.

Die Parameter `-start` und `-stop` sind optional. Sie müssen nur dann angegeben werden, wenn in den Eingabedateien die Platzhalter `<%=nextval%>` und `<%=curval%>` benutzt werden. Der Parameter `-start` gibt dabei an, auf welchen Wert der Zähler am Anfang gesetzt werden soll und `-stop`, bei welchem Wert der Zähler wieder auf den Startwert zurückgesetzt wird. Die Werte für `-start` und `-stop` sollten so gewählt werden, dass möglichst keine SQL-Anweisungen leere Datensätze zurückliefern.

4.2.2 Classpath

Benötigt werden:

- `./lib/sos.connection.jar`
- `./lib/sos.stresstest.jar`
- `./lib/sos.util.jar`

Je nach Einsatz für das Datenbanksystem werden die folgenden Bibliotheken benötigt:

MS SQL Server	<code>./lib/msbase.jar</code> <code>./lib/mssqlserver.jar</code> <code>./lib/msutil.jar</code>
MySQL	je nach verwendeter Datenbankversion muss ein passender Treiber eingesetzt werden, z.B. <code>./lib/mysql-connector-java-3.0.15-ga-bin.jar</code>
Oracle	<code>./lib/ojdbc14.jar</code>

4.2.3 Beispiel eines Startscripts für Unix

```
#!/bin/bash
```

```
java -cp $CLASSPATH:./lib/msbase.jar:./lib/mssqlserver.jar:./lib/msutil.jar:./lib/mysql-connector-java-3.1.0-alpha-bin.jar:./lib/ojdbc14.jar:./lib/sos.connection.jar:./lib/sos.util.jar:./lib/sos.stresstest.jar sos.stresstest.Stresstest -time=15 -threads=10 -input=stress1.sql+stress2.sql -output=stestest.log
```

In diesem Beispiel wird für eine Dauer von 15s (-time=15) ein Stresstest für 10 simultane Benutzer (-threads=10) ausgeführt. Die SQL-Statements stammen aus den Eingabedateien stress1.sql und stress2.sql. Die Ausgabe wird in die Datei stesstest.log geschrieben.

4.2.4 Beispiel eines Startscripts für Windows

```
java -cp %CLASSPATH%;./lib/msbase.jar;./lib/mssqlserver.jar;./lib/msutil.jar;./lib/mysql-connector-java-3.1.0-alpha-bin.jar;./lib/ojdbc14.jar;./lib/sos.connection.jar;./lib/sos.util.jar;./lib/sos.stresstest.jar sos.stresstest.Stresstest -time=15 -threads=10 -input=stress1.sql+stress2.sql -output=stestest.log
```

In diesem Beispiel wird für eine Dauer von 15s (-time=15) ein Stresstest für 10 simultane Benutzer (-threads=10) ausgeführt. Die SQL-Statements stammen aus den Eingabedateien stress1.sql und stress2.sql. Die Ausgabe wird in die Datei stesstest.log geschrieben.

5 Java Programm-Schnittstelle für Stresstests

Neben Datenbankzugriffen können individuelle Java-Programme getestet werden. Hierzu stellt *Stresstest* eine Schnittstelle bereit, mit der individuelle Methoden registriert werden, die beim Testablauf wiederholt aufgerufen werden. Mehrere Methoden können in einer Aktionskette nacheinander aufgerufen werden und werden im Protokoll des *Stresstests* separat mit ihren Verarbeitungszeiten ausgewiesen.

5.1 Beispiel für die Verwendung der Programmschnittstelle

```
import sos.stresstest.*;

/**
 * @author Andreas Püschel <andreas.pueschel@sos-berlin.com>
 * @since 2004-11-21
 * @version 1.0
 * custom class for stresstest session implementation
 */
public class SampleStresstestSession extends SOSStresstestSession
{
    /**
     * Constructor
     * @param stresstest_thread Thread Identifier
     */
    public SampleStresstestSession( StresstestThread stresstest_thread ) throws Exception
    {
        super( stresstest_thread );
    }

    /**
     * runs the action chain
     */
    public void executeChain() throws Exception
    {
        try {
            // execute test chain
            execute( "method_test" );
        } catch (Exception e) {
            throw new Exception("..error in executeChain(): " + StresstestExceptionMes-
            sage.getMessage(e));
        }
    }

    /**
     * runs the action
     */
    public void method_test() throws Exception {
        for(int i=0; i<3; i++) {
            Thread.sleep(300);
        }
    }
}
```

Im Beispiel wird eine individuelle Klasse `SampleStresstestSession` von der Basisklasse `SOSStresstestSession` abgeleitet, die im ausgelieferten Archiv `sos.stresstest.jar` enthalten ist.

Die Methode `executeChain()` wird von der Beispielklasse implementiert und ruft mittels eines oder mehrerer Aufrufe von `execute(<method>)` die angegebene Methode auf.

Um dieses Beispiel zu starten, können Sie ein Startscript schreiben, dass das Archiv mit Ihrer Klasse in den Classpath aufnimmt (`sample.jar`) und den Namen der Klasse im Parameter `-class` übergibt:

```
java -cp %CLASSPATH%;./sample.jar;./lib/sos.stresstest.jar sos.stresstest.Stresstest
-class=SampleStresstestSession -time=15 -threads=10 -output=stresstest.log
```


5.2 Methoden des Objekts StresstestSession

5.2.1 executeChain()

Methode: `public void executeChain() throws Exception`

Argument: ohne

Rückgabewert: void

Beschreibung: Die Methode muss von einer individuellen Stresstest-Klasse implementiert werden und enthält i.d.R. mehrere Aufrufe anderer Testmethoden, die als Aktionskette ausgeführt werden. Diese Aufrufe erfolgen jeweils mit der Methode `execute()`, z.B.:

```
public void executeChain() throws Exception {  
    execute("myMethod");  
}
```

Mit dem Parameter `-execute` beim Programmstart kann vereinbart werden, dass nur die im Wert des Parameters angegebene Testmethode ausgeführt wird und nicht alle in der Methode `executeChain()` aufgeführten Testmethoden.

5.2.2 execute(methodName)

Methode: `public StresstestResponse execute(String methodName) throws Exception`

Argument: Name der zu testenden Methode

Rückgabewert: Objekt der Klasse `stresstestResponse`

Beschreibung: Die Methode wird in der o.g. Methode `executeChain()` verwendet, um eine oder mehrere individuelle Methoden aufzurufen, die Gegenstand des Stresstests sind.

5.2.3 execute(methodName, fileName)

Methode: `public StresstestResponse execute(String methodName, fileName) throws Exception`

Argument: Name der zu testenden Methode, Dateiname des Startparameters `-input`

Rückgabewert: Objekt der Klasse `stresstestResponse`

Beschreibung: Die Methode wird in der o.g. Methode `executeChain()` verwendet, um eine oder mehrere individuelle Methoden aufzurufen, die Gegenstand des Stresstests sind. Im zweiten Argument wird der Name einer Datei mit Eingabedaten erwartet, der Dateiname kann in der mit `methodName` angegebenen Methode mittels `get_current_file()` abgerufen werden.

5.2.4 execute(url)

Methode: `public StresstestResponse execute(URL url) throws Exception`

Argument: Name der zu testenden Methode

Rückgabewert: Objekt der Klasse `stresstestResponse`

Beschreibung: Die Methode wird in der o.g. Methode `executeChain()` verwendet, um eine oder mehrere individuelle Methoden aufzurufen, die Gegenstand des Stresstests sind. Als Argument wird eine URL erwartet, die von der Methode `execute()` aufgerufen und deren Inhalt ausgelesen wird.

5.2.5 `get_current_file()`

Methode: `public String get_current_file() throws Exception`

Argument: ohne

Rückgabewert: Name der aktuellen Eingabedatei des Parameters *-input* beim Programmstart

Beschreibung: Die Methode liefert den Wert des Parameters *-input* zurück, der beim Programmstart des Stresstests übergeben werden kann. Der Parameter enthält einen oder mehrere Dateinamen, die Methode liefert jeweils den Namen der aktuellen Eingabedatei.

5.2.6 `get_param()`

Methode: `public String get_param() throws Exception`

Argument: ohne

Rückgabewert: Wert des Parameters *-param* beim Programmstart

Beschreibung: Die Methode liefert den Wert des Parameters *-param* zurück, der beim Programmstart des Stresstests übergeben werden kann.

5.2.7 `get_execute()`

Methode: `public String get_execute() throws Exception`

Argument: ohne

Rückgabewert: Wert des Parameters *-execute* beim Programmstart

Beschreibung: Die Methode liefert den Wert des Parameters *-execute* zurück, der beim Programmstart des Stresstests übergeben werden kann.

5.2.8 `get_stresstest()`

Methode: `public Stresstest get_stresstest() throws Exception`

Argument: ohne

Rückgabewert: Objekt des Stresstests

Beschreibung: Die Methode liefert das zentrale Stresstest-Objekt zurück für den Zugriff auf weitere Methoden und Parameter des Tests.

5.2.9 `get_connection()`

Methode: `public SOSConnection get_connection() throws Exception`

Argument: ohne

Rückgabewert: Objekt der Datenbankverbindungsklasse `SOSConnection`

Beschreibung: Die Methode liefert die Instanz der Datenbankverbindungsklasse `sosconnection` zurück. Eine Datenbankverbindung wird automatisch pro Thread instantiiert, wenn im Parameter `-config` ein Dateiname mit gültigen Einstellungen für eine Datenbankverbindung übergeben wurde.

5.2.10 `get_logger()`

Methode: `public SOSLogger get_logger() throws Exception`

Argument: ohne

Rückgabewert: Objekt der Logging-Klasse `sosLogger`

Beschreibung: Die Methode liefert die Instanz der Logging-Klasse `sosLogger` zurück. Ein Logger wird automatisch pro Thread instantiiert.

5.2.11 `get_thread_index()`

Methode: `public int get_thread_index() throws Exception`

Argument: ohne

Rückgabewert: Index des aktuellen Threads

Beschreibung: Die Methode liefert den Index des aktuellen Threads zurück, deren Wertebereich zwischen 1 und der Anzahl der im Wert des Parameters `-threads` angegebenen Anzahl simultaner Threads liegt. Der hier zurück gelieferte Thread-Index wird im Protokoll ausgegeben.

5.2.12 `get_session_number()`

Methode: `public int get_session_number() throws Exception`

Argument: ohne

Rückgabewert: laufende Nummer der aktuellen Sitzung im Thread

Beschreibung: Die Methode liefert die laufende Nummer der Sitzung des aktuellen Threads zurück, die vom Stresstest vergeben wird. Die Sitzungsnummer wird pro Thread beginnend mit 1 inkrementiert. Die hier zurück gelieferte Sitzungsnummer wird im Protokoll ausgegeben.

5.2.13 `get_current_value()`

Methode: `public static long get_current_value() throws Exception`

Argument: ohne

Rückgabewert: aktueller Wert des Zählers beginnend mit dem Wert des Parameters `-start`

Beschreibung: Die Methode liefert den aktuellen Wert des Parameters `-start` zurück, der beim Programmstart des Stresstest übergeben werden kann.

5.2.14 `get_next_value()`

Methode: `public static long get_next_value() throws Exception`

Argument: ohne

Rückgabewert: inkrementierter Wert des Zählers beginnend mit dem Wert des Parameters `-start`

Beschreibung: Die Methode liefert den inkrementierten Wert des Parameters *-start* zurück, der beim Programmstart des Stresstest übergeben werden kann. Der Zähler wird thread-übergreifend durch jeden Aufruf dieser Methode inkrementiert.

5.3 Methoden des Objekts StresstestResponse

5.3.1 `get_response_code()`

Methode: `public int get_response_code() throws Exception`

Argument: ohne

Rückgabewert: Numerischer Rückgabewert

Beschreibung: Die Testmethode `execute()` liefert ein Objekt der Klasse `StresstestResponse` zurück, deren Methode `get_response_code()` den Rückgabewert der Verarbeitung enthält. Bei Aufrufen von URLs mit der Testmethode `execute()` wird von dieser Methode der HTTP Statuscode zurückgeliefert.

5.3.2 `get_response_message()`

Methode: `public String get_response_message() throws Exception`

Argument: ohne

Rückgabewert: Fehlernachricht der aufgerufenen Testmethode

Beschreibung: Die Testmethode `execute()` liefert ein Objekt der Klasse `StresstestResponse` zurück, deren Methode `get_response_message()` den Rückgabewert der Verarbeitung enthält. Bei Aufrufen von URLs mit der Testmethode `execute()` wird von dieser Methode die HTTP Statusmeldung zurückgeliefert.

6 Protokolle

6.1 Standard-Protokoll

Das Standardprotokoll, das während des Tests erzeugt wird, liegt im Arbeitsverzeichnis der Anwendung. Es enthält neben der genauen Laufzeit jedes einzelnen Threads Informationen über im Test aufgetretene Fehler.

Beispiel eines Standard-Protokolls

```
2004-08-16 10:31:56 2004-08-16 10:31:57 0 1 union.sql 140
```

Die einzelnen Werte der Zeilen im Protokoll haben folgende Bedeutung:

2004-08-16 10:31:56 = Datum und Zeit des Starts des Lasttests

2004-08-16 10:31:57 = Datum und Zeit der Erzeugung des Protokolleintrags

0 = laufende Nummer des Threads

1 = laufende Nummer des wiederholten Durchlaufens der Eingabedatei bzw. Funktion

union.sql = Name der Eingabedatei oder Funktion

140 = Antwortzeit der Datenbank in Millisekunden

Diese Protokolldatei wird vom Stresstest fortlaufend beschrieben, sie wird nicht automatisch bei Beginn einer Testreihe gelöscht, damit mehrere Testreihen in eine Protokolldatei aufgenommen und zur Auswertung mit *Stresstest Import* in eine Datenbank übernommen werden können. Sie können die Protokolldatei entweder manuell löschen oder bei Beginn einer neuen Testreihe den Parameter `-append-output` auf den Wert `false` setzen, wodurch die Protokolldatei überschrieben wird.

6.2 Thread-Protokoll

Diese Protokolle liegen im Unterverzeichnis `./logs` bzw. in dem beim Startparameter `-log-dir` angegebenen Verzeichnis. Die Dateien werden während des Testablaufs von jedem Thread angelegt bzw. bei `-append-output=true` (Standard) weitergeführt. In diesen Dateien befindet sich bei erfolgreichem Test nur die Startzeit des Threads und im Fehlerfall ein ausführlicher Stack-Trace des im Thread aufgetretenen Fehlers.

Beispiel eines Thread-Protokolls

```
10:31:56.979 ----- Montag August 16 10:31:56 2004
```

10:31:56.979 = Zeitstempel bei Thread-Start

----- = kein Fehler ansonsten würde hier [error] stehen

Montag August 16 10:31:56 2004 = Datum und Zeit bei Thread Start

Um die Übersicht zu behalten sollte diese Datei vor oder nach jeder Testreihe gelöscht werden.

6.3 Unterschied zwischen Thread-Protokoll und Standard-Protokoll

Der Unterschied besteht darin, dass das Standard-Protokoll eine Zusammenfassung sämtlicher Thread-Protokolle ist, sortiert nach der Thread-Nummer. Die Protokollierung jedes einzelnen Threads ist nötig, da andernfalls eine Synchronisierungslogik der Threads erforderlich würde, die den Stresstest ausbremste. Das Standard-Protokoll kann dazu verwendet werden, um die gesammelten Daten in eine Datenbank zu importieren.

7 Ergänzende Tools

7.1 Stresstest Import

Das *Stresstest Import* Tool ist in der Lage die vom Stresstest erzeugten Protokoll-Dateien auszulesen und die Ergebnisse in einer Datenbank abzuspeichern.

7.1.1 Starten des Stresstest Imports

Gestartet wird der *Stresstest Import* bspw. mit folgenden Argumenten:

```
java -classpath=./lib/mysql-connector-java-3.0.15-ga-bin.jar;
./lib/sos.connection.jar;./lib/sos.util.jar;./lib/sos.stresstest.jar
sos.stresstest.StresstestImport
    -config=config/sos_settings.ini
    -log=stresstest.log
```

Beachten Sie bitte, dass die Archive für *-classpath* im obigen Beispiel für Windows mit ";" getrennt sind, unter Unix werden Archive durch "." getrennt. Die Argumente beim Programmstart haben folgende Bedeutung:

<i>-classpath</i>	Java-Classpath mit den Java-Archiven des Stresstests
<i>-config</i>	Datei, in der die Zugangsdaten zur Datenbank enthalten sind
<i>-log</i>	Protokoll-Datei, in der die Ausgabe des Stresstests enthalten ist, d.i. der Wert des Parameters
	<i>-output</i> beim Start des Stresstests
<i>-clean</i>	löscht alle vorhandenen Datensätze vor dem Import des Protokolls, gültige Werte sind <i>true</i> und <i>false</i> , Voreinstellung ist <i>false</i> .

Beispiel für ein Importscript für Unix

```
#!/bin/bash
java -cp $CLASSPATH:./lib/msbase.jar:./lib/mssqlserver.jar:./lib/msutil.jar:./lib/mysql-
connector-java-3.0.15-ga-bin.jar:./lib/ojdbc14.jar:./lib/sos.connection.jar:
./lib/sos.util.jar:./lib/sos.stresstest.jar:./lib/sos.stresstest.jar
sos.stresstest.StresstestImport $*
```

Beispiel für ein Importscript für Windows

```
java -cp %CLASSPATH%;./lib/msbase.jar;./lib/mssqlserver.jar;./lib/msutil.jar;./lib/mysql-
connector-java-3.0.15-ga-
bin.jar;./lib/ojdbc14.jar;./lib/sos.connection.jar;./lib/sos.util.jar;./lib/sos.stresstest
.jar sos.stresstest.StresstestImport %*
```

Die benötigten .jar-Dateien im Classpath richten sich nach der Datenbank, in die die Ergebnisse importiert werden sollen (siehe Stresstest). Die Parameter *-config* und *-log* haben die Standardwerte *./config/sos_settings.ini* und *stresstest.log*. Die Tabelle für die Ergebnisse wird automatisch beim ersten Starten des *Stresstest Imports* angelegt und heißt STRESSTEST.

7.2 Stresstest Report

Der *Stresstest Report* wertet Stresstest-Ergebnisse aus, die von *Stresstest Import* in einer Datenbank abgespeichert wurden. Das Tool ermöglicht es Ergebnisse einer weiterentwickelten Test-Funktion mit den vorherigen Versionen zu vergleichen, sofern die weiterentwickelte Test-Funktion und die Vorgängerversion mit dem gleichen Namen in die Datenbank importiert wurden. Dies ist für mehrere Funktionen gleichzeitig möglich. Außerdem ist der Vergleich zwischen mehreren Testreihen möglich.

7.2.1 Starten des Stresstest Reports

Gestartet wird Stresstest Report bspw. mit folgenden Argumenten:

```
java -classpath=../lib/mysql-connector-java-3.0.15-ga-bin.jar;
../lib/sos.connection.jar;../lib/sos.util.jar;../lib/sos.stresstest.jar
sos.stresstest.StresstestReport
-config=config/sos_settings.ini
-compare=1
-functions=stress1.sql+stress2.sql
```

Beachten Sie bitte, dass die Archive für *-classpath* im obigen Beispiel für Windows mit ";" getrennt sind, unter Unix werden Archive durch ":" getrennt. Die Argumente beim Programmstart haben folgende Bedeutung:

- classpath Java-Classpath mit den Java-Archiven der Software
- config Datei, in der die Zugangsdaten zur Datenbank enthalten sind
- compare Anzahl zurückliegender Testreihen, die verglichen werden (Standardwert: 1)
- functions Optionale Funktion, die verglichen werden soll. Mehrere Funktionen werden durch + getrennt; ohne diese Angabe werden alle Funktionen verglichen

Die Angaben im Classpath richten sich nach der verwendeten Datenbank, in der die Protokoll-Dateien des Stresstests importiert wurden (siehe Stresstest).

Die Standardwerte der Parameter sind:

- config ../config/sos_settings.ini
- functions keine Angabe, d.h. es werden alle Funktionen verglichen
- compare 1, d.h. es wird nur die letzte Testreihe ausgewertet

Beispiel für ein Reportscrip für Unix

```
#!/bin/bash
java -cp $CLASSPATH:../lib/msbase.jar:../lib/mssqlserver.jar:../lib/msutil.jar:../lib/mysql-
connector-java-3.0.15-ga-bin.jar:../lib/ojdbc14.jar:../lib/sos.connection.jar:
../lib/sos.util.jar:../lib/sos.stresstest.jar:../lib/sos.stresstest.jar
sos.stresstest.StresstestReport $*
```

Beispiel für ein Reportscrip für Windows

```
java -cp %CLASSPATH%;../lib/msbase.jar;../lib/mssqlserver.jar;../lib/msutil.jar;../lib/mysql-
connector-java-3.0.15-ga-
bin.jar;../lib/ojdbc14.jar;../lib/sos.connection.jar;../lib/sos.util.jar;../lib/sos.stresstest
.jar sos.stresstest.StresstestReport %*
```


Beispiel eines Reports

time	function	min ms	max ms	avg ms	threads	+ - min	+ - max	+ - avg	% min	% max	% avg
2004-08-25 19:06:01	artikelstam	630	995	673.3	1	0.0	0.0	0.0	0.0	0.0	0.0
	m.sql										
	artikelsuch	7	24	9.5	1	0.0	0.0	0.0	0.0	0.0	0.0
	e.sql										
	auftragsbes	7	10	8.0	1	0.0	0.0	0.0	0.0	0.0	0.0
	tand.sql										
2004-08-25 18:29:28	artikelstam	625	1003	672.3	1	-5.0	8.0	-1.0	-0.7	0.8	-0.1
	m.sql										
	artikelsuch	7	27	10.6	1	0.0	3.0	1.0	0.0	12.5	11.1
	e.sql										
	auftragsbes	7	21	8.8	1	0.0	11.0	0.8	0.0	109.9	10.8
	tand.sql										