CloudTest Getting Started Guide

1.5.x.Final

jimy_xue@163.com

vrules4j@gmail.com

jordan.xue@hotmail.com

2018-07

Preface

About CloudTest License

CloudTest is an open source project initiated by the individual, which is distributed on Apatche Licenses 2.0 releases. Every organization or individual is fully granted for viewing its source code or modify the source codes to match his/her own special needs; commercial applications have the greatest license under Apatche Licenses 2.0, More detailed specific Licenses see http://www.apache.org/licenses/

CloudTest is free software, which is distributed in the hope that it will be useful not only for the development founder but also for all individuals and organizations who is in need.

Key Values

Maybe you are also looking for a new testing framework can help do following:

1. Generate all test cases automatically in one second with test data.
2. It can test all methods including public protected, as well as private
3. Automation regression is also build in framework
4. None java coding jobs for test case development
5. You can test you cases in every where and without time and geographical restrictions
    a) By mail
    b) By your mobile phone
    c) The only thing you need is internet

CloudTest is a redefined unit testing approach and methodology, which can make your testing jobs become much more easy and efficient. It is a pure java lightweight framework integrated test cases management. Test data management, assert management, automation regression, performance monitor and test report in one.

1. High performance
2. None java code
3. Assert support inside
4. Generate test case and data automatically
5. Distributed and remote supported
6. All methods are testable, including public, protected and private
7. Cloud oriented services, including email and web service and so on
8. Hot deployment supported
9. Automatical regression and report supported
10. Performance monitor and loading testing supported
11. Spring supported

12. Transaction management supported
13. Plugin architecture designed, which is extensible and customized as demand
14. Integrated with JUnit closely (Including JUnit4)
15. Parallel computing supported
16. Hotspots computing and report supported

Obtaining Release Package

https://sourceforge.net/projects/cloudtest/

1.1 Quick Start

1. Download cloudtest release from:https://sourceforge.net/projects/cloudtest/
2. Download dependent third party libraries
   a) vRules4j
   b) Bsh
   c) XStream
3. Set cloudtest core jar to classpath, if using container, you can copy all of them to 'WEB-INF/lib/' directly would be fine.
4. Copy cloudtest root folder to you local, e.g 'd:\cloud_test\cloudtest\'
5. Configure CLOUDTEST_HOME environment variable as 'd:\cloud_test'. You can also set the home path via -Dcloudtest.home = "d:\cloud_test" in runtime.

```
<servlet>
        <servlet-name>CloudTestServlet</servlet-name>
        <servlet-class>com.unibeta.cloudtest.servlet.CloudTestServlet</servlet-class>
        <init-param>
            <param-name>CLOUDTEST_HOME$PathProvider</param-name>
            <param-value>com.unibeta.cloudtest.config.impl.CLOUDTEST_HOME$PathProviderImpl</param-
value>
        </init-param>
        <init-param>
            <param-name>ROOT_FOLDER_NAME</param-name>
            <param-value>cloudtest</param-value>
        </init-param>
        <load-on-startup>2</load-on-startup>
    </servlet>
    <servlet>
        <servlet-name>vRules4jServlet</servlet-name>
        <servlet-class>com.unibeta.vrules.servlets.VRules4jServlet</servlet-class>
```

```
        <load-on-startup>1</load-on-startup>
    </servlet>
```

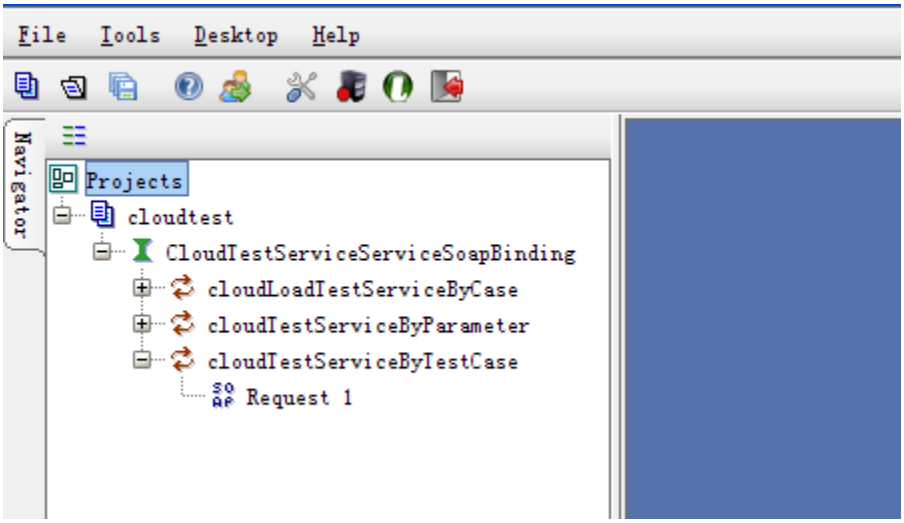
6. Configure servlet in web.xml as below, it will help deploy cloudtest webservice automatically.
7. Enable web service engine in your container, for example CXF or axis2. Make sure the web service engine is available after startup the container.
8. Start the web contain with web service engine enabled,
9. Input http://localhost:8080/website/services in URL, if you get below web service deployed.

10. Conglutination! Enjoy your cloud test journey, where make your testing much more easy/efficient and enjoyable.

1.2 Start To Test By Cloud Service

After cloudtest was installed successfully, you can start to test via web service test tool. SoapUI is recommended.

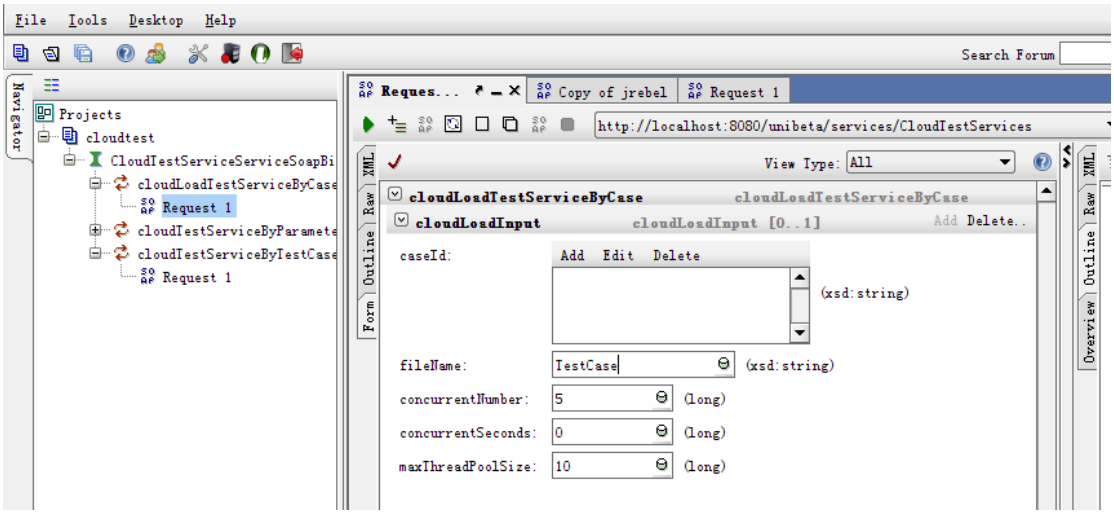
1. Input the cloudtest web service end point and create a SoapUI project



2. You can see three testing services listed below:

a) cloudLoadTestServiceByCase

A loading test service by cases. It can simulate the concurrent sure number and max thread number.

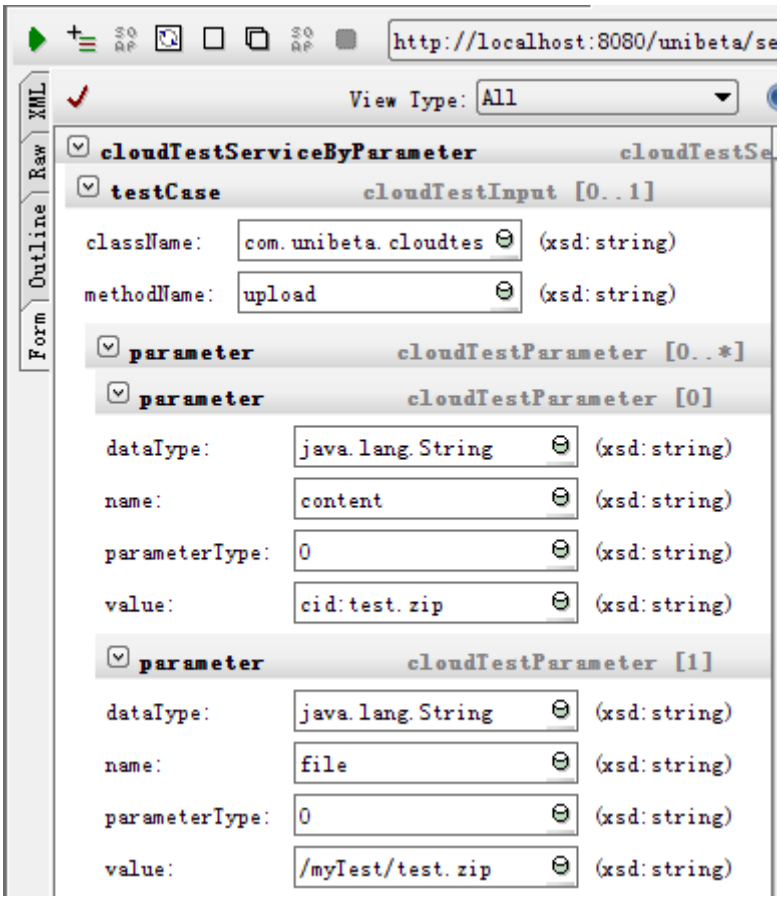


Testing pay-load input:

```
<cloudLoadInput>
    <caseId>?</caseId>
    <fileName>TestCase</fileName>
    <concurrentNumber>100</concurrentNumber>
   <concurrentSeconds>0</concurrentSeconds>
  <maxThreadPoolSize>10</maxThreadPoolSize>
</cloudLoadInput>
```

b) cloudTestServiceByParameter

A general testing service by parameter. it can test all class's methods deployed in web container, including private, protected and public.

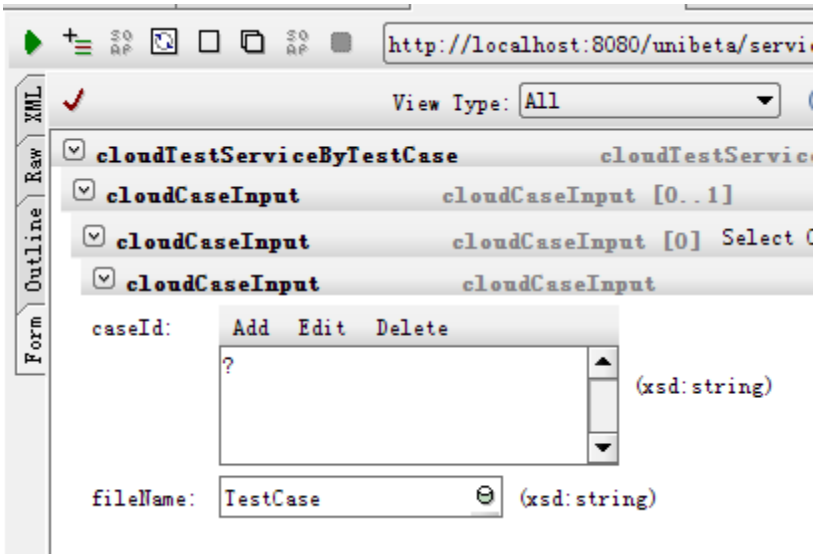


Testing pay-load input:

```
<testCase>
      <className>MyTest</className>
     <methodName>test</methodName>
      <parameter>
          <dataType>java.lang.String</dataType>
          <name>param name</name>
          <parameterType>1</parameterType>
          <value>"this is a string value."</value>
      </parameter>
</testCase>
```

c) cloudTestServiceByTestCase

A service for running test suite from external cased in xml.



Testing pay-load input:

```
<cloudCaseInput>
                <caseId>?</caseId>
                 <fileName>TestCase/</fileName>
</cloudCaseInput>
```

1.3 Cloud Case Definition/Tools

1.3.1 General Configuration Structure

1) ${CLOUDTEST_HOME}/cloudtest
   - Config: Defined general config files, such as PluginConfig.xml
   - TestCase: Stores all test cases in xml managed in folder structure.
   - TestData: Stores all test data in xml.

1.3.2 ParameterType

- 0: primitive data type
  - It can be java statement or Xstream preconize xml pay-load, e.g,
    - ✓ "abc" stands for java.lang.String;
    - ✓ 1L stands for java.lang.Long or long
    - ✓ <string>abc</string> equals to "abc"
- 1: load the test data from external xml file. The xml can be digested by Xstream engine. E.g TestData/myTestData.xml

1.3.3 Java2TestCases

com.unibeta.cloudtest.tool.Java2TestCases is powerful engine to digest java to test cases automatically. E.g

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/" xmlns:com="com.unibeta.cloudtest">
    <soapenv:Header/>
    <soapenv:Body>
        <com:cloudTestServiceByParameter>
            <testCase>
                <className>com.unibeta.cloudtest.tool.Java2TestCases</className>
                <methodName>digestToTestCases</methodName>
                <parameter>
                    <dataType>java.lang.String</dataType>
                    <name>?</name>
                    <parameterType>0</parameterType>
                    <value>"com.unibeta.vrules.engines"</value>
                </parameter>
                <parameter>
                    <dataType>java.lang.String</dataType>
                    <name>file path</name>
                    <parameterType>0</parameterType>
                    <value>""</value>
                </parameter>
                <parameter>
                    <dataType>int</dataType>
                    <name>modifier level</name>
                    <parameterType>0</parameterType>
                    <value>0</value>
                </parameter>
            </testCase>
        </com:cloudTestServiceByParameter>
    </soapenv:Body>
</soapenv:Envelope>
```

1.3.4 Plugins

CloudTest is designed as pluginable for some particular interfaces:

- com.unibeta.cloudtest.config.plugin.elements.ParamConfigServicePlugin
- com.unibeta.cloudtest.config.plugin.elements.ReportGeneratorPlugin
- com.unibeta.cloudtest.config.plugin.elements.SpringBeanFactoryPlugin
- com.unibeta.cloudtest.config.plugin.elements.UserTransactionPlugin

It is configurable in PluginConfig.xml

```
<pluginConfig>
    <plugin
        id="com.unibeta.cloudtest.config.plugin.elements.ParamConfigServicePlugin"
        desc="Global param configuration service plugin">
        <className>com.unibeta.cloudtest.config.plugin.elements.impl.ParamConfigServicePluginImpl
        </className>
    </plugin>
    <plugin
        id="com.unibeta.cloudtest.config.plugin.elements.SpringBeanFactoryPlugin"
        desc="Spring bean factory plugin">
        <className>com.unibeta.cloudtest.config.plugin.elements.impl.SpringBeanFactoryPluginImpl
        </className>
    </plugin>
    <plugin
        id="com.unibeta.cloudtest.config.plugin.elements.UserTransactionPlugin"
        desc="User transaction plugin">
        <className>com.unibeta.cloudtest.config.plugin.elements.impl.UserTransactionPluginImpl
        </className>
    </plugin>
    <plugin
        id="com.unibeta.cloudtest.config.plugin.elements.ReportGeneratorPlugin"
        desc="Report generator">

    <className>com.unibeta.cloudtest.config.plugin.elements.impl.ReportGeneratorPluginImpl</className>
    </plugin>
    <!-- the endpoint address' format is like below "http://localhost:80/myapp/services/CloudTestServices"
        By default, If it is empty, the endpoint is "http://localhost:[port]/[webapp-context-
name]/[webservice-sevlet-name]/CloudTestServices".
        if set it as "null" or "none", system will not publish the web service automatically.
        In this case you can use other methods to deploy the web service. -->
    <param name="cloudtest.WebService.EndpointAddress" value=""/>
    <param name="cloudtest.UserTransaction.JNDI" value="java:comp/UserTransaction"/>
    <param name="cloudtest.AutomationTest.SwitchFlag" value="true"/>
    <param name="cloudtest.AutomationTest.IntervalHours" value="6"/>
    <param name="cloudtest.Email.FromName" value="xxx@company.com"/>
    <param name="cloudtest.Email.HostName" value="mail.company.com"/>
    <param name="cloudtest.Email.SenderUsername" value="xxx"/>
    <param name="cloudtest.Email.SenderPassword" value="@#XX"/>
    <param name="cloudtest.Email.Pop3HostName" value="pop.company.com"/>
    <param name="cloudtest.Email.Pop3Port" value="995"/>
    <param name="cloudtest.Email.StoreProtocal" value="pop3s"/>
    <param name="cloudtest.MailService.DeployedServerName" value="computer name"/>
```

```
    <param name="cloudtest.LoadTest.MaxDetailedResponseAmount" value="100000"/>
    <slave-servers>
      <!-- <server id="slave-server1"
          address="http://localhost1:8080/unibeta/services/CloudTestServices" desc=""/>-->
      <!-- <server  id="slave-server2"
          address="http://localhost2:8080/unibeta/services/CloudTestServices" desc=""/>-->
    </slave-servers>
</pluginConfig>
```

1.3.5 Hotspots Report

Hotspots Report is a powerful report engine to compute all failed test cases with specified sampling rate as below parameter:

```
    <!-- sampling_rate for hotspots report, decimal value is required.
     set as 0.0 to disable hotspots function. By default it is 1.0
     sampling amount is: (history.index.maximum *
hotspots.sampling_rate)-->
    <param name="cloudtest.report.hotspots.sampling_rate" value="1.0"/>
```

The basic functions are below:

1. All test cases that ever failed in local.
2. Illustrates all cases result based on time shaft.
3. Trend report is supported for hotspots report
4. It will be triggered by every report events, including loading test and parallel testing.

Notes:

1. if sampling rate is 1.0, the hotspots report content might be very huge. sampling_rate is recommended as '0.2'~'0.5'.

2. if sampling rate is 0.0, the hotspots report will be disabled.

1.3.6 Default Inner Variables

There are lot default inner variables for convenient usage.

Variable Name	Description	Example
$cache$	System default parameter of cache instance. Value:com.unibeta.cloudtest.config.CacheManagerFactory.getInstance()	$cache$.get("runtime_data",key); $cache$.put("runtime_data",key,value);
$CloudObject$	Used for quick object operation, such as fromJava, fromXMLFile. Evl; Value: com.unibeta.cloudtest.util.ObjectDigester.ObjectDigester()	$CloudObject$.fromJava("return 3+3;");

$RootPath$	A String of cloudtest root path name. Value:com.unibeta.cloudtest.config.ConfigurationProxy.getCloudTestRootPath()	$RootPath$ + "/TestData/xml/my.xml";
$Java2TestCases$	System default parameter of Java2TestCases instance. Value: com.unibeta.cloudtest.tool.Java2TestCases.Java2TestCases()	$Java2TestCases$.digestToMockXmlData(String className, String fileName) $Java2TestCases$.digestToTestCases(String className, String destFileName, int accessLevel)
$beans$	Used for quick java bean operation from Spring. Value: com.unibeta.cloudtest.config.plugin.CloudTestPluginFactory Implements: org.springframework.beans.factory.BeanFactory	$beans$.getBean("springBeanName");
$PluginConfig$	System default parameter of PluginConfig instance. Value: com.unibeta.cloudtest.config.plugin.PluginConfigProxy	$PluginConfig$.getParamValueByName("PARA_NAME");

Runtime user variables can also be defined by below code:
$cache$.put("runtime_data",key,value);
So that, 'key' can be used in anywhere in cloudtest runtime lifecycle. For example,
$cache$.put("runtime_data",no5, 5L);
The result is: 5+no5 = 10;

1.3.7 Automation Case Recorder

CloudTest recorder is AOP oriented, pointcut into particular aspect that has been defined in aop:config and signature regex matches. Before usage recorder function, please do configure below aop:config to enable. For example,

```
<aop:aspectj-autoproxy proxy-target-class="true"/>

<bean name="cloudTestRecorder"
    class="com.unibeta.cloudtest.tool.CloudTestRecorder" />
 <aop:config>
    <aop:aspect id="cloudtestRecorderAop" ref="cloudTestRecorder">
        <aop:pointcut id="target"
            expression="execution(public * com..*.*(..))" />
        <aop:around method="record" pointcut-ref="target" />
    </aop:aspect>
</aop:config>
```

Then, configure case-recorder in PluginConfig.xml

```
<case-recorders>
    <recorder id = "aaa" poweroff = "false" targetCaseFilePath =
"bbb.tc.xml" desc= "">
        <signatureRegex>
            <className>.*</className>
            <modifiers>.*</modifiers>
            <methodName>.*</methodName>
```

```
        </signatureRegex>
        <signatureRegex>
            <className>.*</className>
            <modifiers>.*</modifiers>
            <methodName>.*</methodName>
        </signatureRegex>
    </recorder>
</case-recorders>
```

1.4 Appendix

1.4.1 Can cloudtest framework support all data types?

Yes, all data types are supported, including primitive and complex type, as well as user defined object. The data payload has to be expressed by XStream. E.g, <int>1</int> or <string>my string</string>.

1.4.2 How to generate test case automatically?

com.unibeta.cloudtest.tool.Java2TestCases.digestToTestCases(`String className, String destFileName, int accessLevel`) is the basic utilization tool, which can generate test case by given class name.

className: given class name or package name that are going to generate test cases.
If the value is package name, the engine will digest all class under given package.
destFileName: the target file name, the generate test case files to be saved.
If it is a folder name, test case file will be located in package path under given folder name.
If it is null or empty, the test case file will be located in package path by default.
accessLevel: indicates what kind modify type members' test case will be generated automatically.
0: public
1: protected and public
2: private, protected and pubic
Others: private, protected and pubic

Example:

```
<testCase>
    <className>com.unibeta.cloudtest.tool.Java2TestCases</className>
    <methodName>digestToTestCases</methodName>
    <parameter>
```

```
            <dataType>java.lang.String</dataType>
            <name>?</name>
            <parameterType>0</parameterType>
            <value>"com.unibeta.vrules.engines"</value>
        </parameter>
        <parameter>
            <dataType>java.lang.String</dataType>
            <name>file path</name>
            <parameterType>0</parameterType>
            <value>""</value>
        </parameter>
        <parameter>
            <dataType>int</dataType>
            <name>modifier level</name>
            <parameterType>0</parameterType>
            <value>0</value>
        </parameter>
    </testCase>
```

1.4.3 How to send test report?

Below cloud test statement can auto regress cases and send out the test report.

```
<testCase id="CloudTestReport" assertId="" returnFlag="false"
       desc="Invoke the report service and send mail report">
       <className>
          com.unibeta.cloudtest.tool.CloudTestReportor
       </className>
       <methodName>report</methodName>
       <parameter>
            <name>PorjectName-Module</name>
            <dataType>java.lang.String</dataType>
            <parameterType>0</parameterType>
            <value>"[PorjectName-Module]"</value>
        </parameter>
        <parameter>
            <name>caseFileName</name>
            <dataType>java.lang.String</dataType>
            <parameterType>0</parameterType>
            <value>"TestCase/com"</value>
       </parameter>
       <parameter>
          <name>email address to</name>
```

```
            <dataType>java.lang.String</dataType>
            <parameterType>0</parameterType>
            <value>""</value>
        </parameter>
    </testCase>
```

1.4.4 How to setup automation regression by cloud test?

CloudTest can help manage automation regression, if plugin config was configured correctly as below:

```
<param name="cloudtest.AutomationTest.SwitchFlag" value="true"/>
```

1.4.5 How to configure mail host information?

Email config info can be configure in PluginConfig.xml via below parameters:

```
    <param name="cloudtest.Email.FromName" value="xxx@company.com"/>
    <param name="cloudtest.Email.HostName" value="mail.company.com"/>
    <param name="cloudtest.Email.SenderUsername" value="xxx"/>
    <param name="cloudtest.Email.SenderPassword" value="@#XX"/>
    <param name="cloudtest.Email.Pop3HostName" value="pop.company.com"/>
    <param name="cloudtest.Email.Pop3Port" value="995"/>
    <param name="cloudtest.Email.StoreProtocal" value="pop3s"/>
```

1.4.6 How to deploy Mail Server center?

CloudTest support executing test cases by email, configuration steps:

1. Configure mail account correctly
2. Deploy mail account as mail server center by below parameter config:

```
<param name="cloudtest.MailService.DeployedServerName" value="computer name"/>
```

If current computer user name matches configured "DeployedServerName", current mail account have been deployed as a cloud test mail server center.

For example, if the mail account is abc@xxx.com, if you send below test payload request, the mail server will respond you testing request and send out test report automatically.

```
<cloudTestCase assertRuleFile="">
    <testCase id="CloudTestReport" assertId="" returnFlag="false"
        desc="Invoke the report service and send mail report">
        <className>
```

```
            com.unibeta.cloudtest.tool.CloudTestReportor
        </className>
        <methodName>report</methodName>
        <parameter>
            <name>PorjectName-Module</name>
            <dataType>java.lang.String</dataType>
            <parameterType>0</parameterType>
            <value>"[PorjectName-Module]"</value>
        </parameter>
        <parameter>
            <name>caseFileName</name>
            <dataType>java.lang.String</dataType>
            <parameterType>0</parameterType>
            <value>"TestCase"</value>
        </parameter>
        <parameter>
            <name>email address to</name>
            <dataType>java.lang.String</dataType>
            <parameterType>0</parameterType>
            <value>""</value>
        </parameter>
    </testCase>
</cloudTestCase>
```

1.4.7 How to publish cloud test web service?

By default, cloud test web service is deployed by servlet automatically while the web container startup.

```
<servlet>
        <servlet-name>CloudTestServlet</servlet-name>
        <servlet-
class>com.unibeta.cloudtest.servlet.CloudTestServlet</servlet-class>
        <init-param>
            <param-name>CLOUDTEST_HOME$PathProvider</param-name>
            <param-
value>com.unibeta.cloudtest.config.impl.CLOUDTEST_HOME$PathProviderImpl</par
am-value>
        </init-param>
        <init-param>
            <param-name>ROOT_FOLDER_NAME</param-name>
            <param-value>cloudTest</param-value>
```

```
        </init-param>
        <load-on-startup>2</load-on-startup>
    </servlet>
```

But, it is not the only way to publish clout test web service. You can also deploy it by JBOSS ESB or Spring in demand.

```
@WebService(name = "CloudTestService", targetNamespace = "com.unibeta.cloudtest")
```

com.unibeta.cloudtest.CloudTestService

```
@WebMethod(operationName = "cloudTestServiceByTestCase")
    @WebResult(name = "cloudTestResult")
    public CloudTestOutput doTest(
            @WebParam(name = "cloudCaseInput", mode = Mode.IN)
            CloudCaseInput input)
```

```
@WebMethod(operationName = "cloudTestServiceByParameter")
    @WebResult(name = "cloudTestResult")
    public CloudTestOutput doTest(@WebParam(name = "testCase", mode = Mode.IN)
    CloudTestInput input)
```

```
@WebMethod(operationName = "cloudLoadTestServiceByCase")
    @WebResult(name = "cloudTestResult")
    public CloudTestOutput doLoadTest(
            @WebParam(name = "cloudLoadInput", mode = Mode.IN)
                CloudLoadInput loadTestInput)
```

1.4.8 How to import external cases to running context?

'imports 'attribute under cloudTestCase can import external case file to current runtime life-cycle.

1. './' stands for current folder
2. '../'stands for parent folder
3. '../../' stands for parent's parent folder, just as Linux file operation command
4. Multiple case files can be split by ',' or ';'

E.g:

```
<cloudTestCase assertRuleFile="DataTypeTest.assert.xml"
imports="./DataTypeTest1.tc.xml,../DataTypeTest2.tc.xml">
```

1.4.9 How to be compatible with other unit test framework?

Cloud test can be compatible with other unit test frameworks via implements com.unibeta.cloudtest.config.plugin.elements.CaseRunnerPlugin. It can be configured in PluginConfig.xml

By default, com.unibeta.cloudtest.config.plugin.elements.impl.JUnitCaseRunnerPluginImpl can support JUnit cases smoothly.

1.4.10 How to use parallel computing?

Parallel computing is supported from v1.2 version.
Step1:
Configurate slave-servers in PluginConfig.xml, for example:

```
<slave-servers>
      <!-- <server
id="slave-server1"
address="http://localhost1:8080/unibeta/services/CloudTestServices"
desc=""/>-->
      <!-- <server
id="slave-server2"
address="http://localhost2:8080/unibeta/services/CloudTestServices"
desc=""/>-->
</slave-servers>
```

Step2:
Run parallel job with below service:

```
com.unibeta.cloudtest.parallel.ParallelJob
public static CloudTestOutput run(String caseUri, String mail)
```

Execution Process:
1. cloudtest framework will distribute given case URI related tasks to all slave servers concurrently.
2. Execute sub tasks in multiple servers defined in slave-servers.
3. Collect all testing result to host and send the report to given mail address finally.

Step3:
Check the test report in email.