Plugins mit ADITO

AID 004 DE

Autor: FA, MN, MW, KN, JK. Version 10.2. Zuletzt geändert 31.08.2017

Version	Änderungen
10.2	PlugIn-Toolkit (Plugin-spi) jetzt über Maven Central zugänglich
10.1	Anpassung der Formatierungen
10.0	Anpassung an ADITO4.6
2.1	Ablagepfade der Plugins hinzugefügt
2.0	Letzter Stand vor Übernahme in Versionierung

Inhaltsverzeichnis

1. Voraussetzungen

1.1. Java Software Development Kit

Um PlugIns für ADITO4 schreiben zu können, müssen Sie das Java SDK in der von ADITO freigegebenen Version installiert haben.

Welche Version von Java für ADITO4 frei gegeben wurde, steht im ADITO4 Informationsdokument AID017-DE - ADITO Technikbroschüre.

1.2. Editor oder IDE

Zum Schreiben des Quellcodes benötigen Sie einen Text-Editor. Programmieren Sie öfter mit Java, können Sie natürlich eine entsprechende Entwicklungsumgebung (Eclipse, NetBeans, IntelliJ, o.ä.) verwenden.

1.3. ADITO PlugIn-Toolkit

Das ADITO PlugIn-Toolkit besteht aus den beiden Klassen `AbstractClientPlugin` und `AbstractPlugin`, die Ihre Plugins erweitern müssen.

Das ADITO PlugIn-Toolkit (Plugin-spi) ist auf der Maven Central öffentlich mit folgenden Koordinaten zugänglich:

```
<groupId>de.adito.aditoweb</groupId>
<artifactId>plugin-spi</artifactId>
<version>1.0.0</version>
```

Außerdem benötigen Sie zum Entwickeln von ADITO PlugIns die Bibliothek `serversplash.jar` für Server-PlugIns und die Bibliothek `client.jar` für Client-PlugIns. Diese beiden Bibliotheken finden Sie im Verzeichnis `lib/server` bzw. `lib/client` Ihrer ADITO4 Distribution. Sie enthalten die PluginFacade und die ClientPluginFacade, deren Methoden Sie im PlugIn nutzen können.

2. ADITO4 PlugIns

2.1. Server-PlugIns

Server-PlugIns werden am Server ausgeführt. Damit unterliegen sie gewissen Beschränkungen. So können Server-PlugIns auch nur auf das Dateisystem am Server zugreifen.

2.2. Client-PlugIns

Client-PlugIns werden am Client ausgeführt. Das impliziert, dass die Jar-Datei, die das PlugIn enthält und weitere benötigte Bibliotheken und Ressourcen, auf jedem Client, auf dem das PlugIn ausgeführt werden soll, vorhanden sein müssen.

3. PlugIns schreiben

3.1. Server-PlugIns

Ihr Server-PlugIn besteht im Wesentlichen aus einer Java-Klasse, die `AbstractPlugin` erweitert. In dieser Klasse müssen Sie die beiden Methoden `getDescription` und `execute` von `AbstractPlugin` überladen.

Die Signaturen dieser beiden Methoden lauten wie folgt:

- `public String getDescription()`
- `public Object[] execute(Object[] pParameters, IPluginFacade pPluginFacade) throws PluginException`

Das Ergebnis der Methode `getDescription` wird im Serverlog mit angezeigt, wenn beim Initialisieren, Laden oder Ausführen des PlugIns ein Fehler aufgetreten ist, der nicht abgefangen wurde. Damit kann das PlugIn identifiziert werden.

In der Methode `execute` können Sie weitere Klassen und Bibliotheken aufrufen. Außerdem haben Sie in dieser Methode Zugriff auf das Logging-Backend und die DB-Schicht und können die Methoden `log`, `doDBRequest` und `runJDitoProcess` der PluginFacade nutzen (siehe unten).

Beispiel

```
package de.adito.plugin;

import de.adito.aditoweb.common.jdito.plugin.*;
import de.adito.aditoweb.common.jdito.plugin.impl.AbstractPlugin;



/**
 * Demo-Plugin
 *
 * @author ADITO Software GmbH, 2007
 */



public class DemoPlugin extends AbstractPlugin
{
  static
  {
```

```
 System.out.println("************************");
 System.out.println("************************");
 System.out.println("************************");
 System.out.println("*PlugIn zur Demonstration der PlugIn-
Funktionalität*");
 System.out.println("************************");
 System.out.println("************************");
 System.out.println("************************");
  }


  public String getDescription()
  {
    return "DemoPlugin: PlugIn zur Demonstration der PlugIn-
Funktionalität";
  }


  public Object[] execute(Object[] pParameters, IPluginFacade
pPluginFacade)
      throws PluginException
  {
    try
    {
     // Ergebnis der Methode zurückliefern
 return new String[]{"Ergebnis des PlugIns: " +
pParameters[0].toString()};


    }
    catch (Exception ex)
    {


    }
  }
}
```

Der static-Teil des PlugIns wird beim Laden des PlugIns im Serverlog angezeigt. Ist ein PlugIn einmal im Server geladen, kann ein erneutes Laden des PlugIns nur durch Neustart des Servers erzwungen werden!

Haben Sie den Quellcode Ihres PlugIns erstellt, übersetzen Sie den Quellcode mit dem Java-Compiler in eine Class-Datei. Diese packen Sie dann zusammen mit der Klasse `AbstractPlugin` in eine Jar-Datei. Diese Jar-Datei legen Sie im Ordner lib\server\plugin in Ihre ADITO4 Distribution ab.

Bitte achten Sie darauf, dass die Klasse `AbstractPlugin` in der Jar-Datei im Pfad de/adito/aditoweb/common/jdito/plugin/impl liegt. Dieser Pfad spiegelt die Package-Struktur wider und darf nicht verändert werden.

Damit ist ihr PlugIn einsatzfähig.

Die Klasse `AbstractPlugin` muss in der Jar-Datei mitgepackt werden. Dadurch kann der ADITO Server beim Laden des PlugIns überprüfen, ob dieses noch versionskompatibel ist.

Nutzen Sie in Ihrem PlugIn externe Bibliotheken, die auch bereits in der ADITO4-Distribution enthalten sind, müssen diese Bibliotheken in der von ADITO mitgelieferten Version verwendet werden. Die Verwendung anderer Versionen dieser Bibliotheken ist nicht möglich!

Weitere zusätzlich verwendete Bibliotheken müssen in den Klassenpfad aufgenommen werden.

3.2. Client-PlugIns

Ihr Client-PlugIn besteht im Wesentlichen aus einer Java-Klasse, die `AbstractClientPlugin` erweitert. In dieser Klasse müssen Sie die beiden Methoden `getDescription` und `execute` von `AbstractClientPlugin` überladen.

Die Signaturen dieser beiden Methoden lauten wie folgt:

- `public String getDescription()`
- `public String[] execute(String[] pParameters, IClientPluginFacade pPluginFacade) throws PluginException`

Das Ergebnis der Methode `getDescription` wird im Clientlog mit angezeigt, wenn beim Initialisieren, Laden oder Ausführen des PlugIns ein Fehler aufgetreten ist, der nicht abgefangen wurde. Damit kann das PlugIn identifiziert werden.

In der Methode `execute` können Sie weitere Klassen und Bibliotheken aufrufen. Außerdem haben Sie in dieser Methode Zugriff auf das Logging-Backend und können die Log-Methoden der ClientPluginFacade nutzen (siehe unten).

Beispiel

```
package de.adito.plugin;


import de.adito.aditoweb.common.jdito.plugin.*;
import
de.adito.aditoweb.common.jdito.plugin.impl.AbstractClientPlugin;
```

```
/**
 * Demo-Clientplugin
 *
 * @author ADITO Software GmbH, 2007
 */


public class DemoClientPlugin extends AbstractClientPlugin
{
  static
  {
 System.out.println("*********************************");
 System.out.println("*********************************");
 System.out.println("*********************************");
 System.out.println("*Client-PlugIn zur Demonstration der PlugIn-
Funktionalität*");
 System.out.println("*********************************");
 System.out.println("*********************************");
 System.out.println("*********************************");
  }


  public String getDescription()
  {
    return "DemoClientPlugin: Client-PlugIn zur Demonstration der
PlugIn-Funktionalität";
  }


  public String[] execute(String[] pParameters, IClientPluginFacade
pPluginFacade)
      throws PluginException
  {


    try
    {
      return new String[]{"Ergebnis des Client-PlugIns: " +
pParameters[0].toString()};
```

```
    }
    catch (Exception ex)
    {
      return new String[]{"Client-PlugIn: No Message"};
    }
  }
}
```

Der static-Teil des PlugIns wird beim Laden des PlugIns im Clientlog angezeigt. Ist ein PlugIn einmal im Client geladen, kann ein erneutes Laden des PlugIns nur durch Neustart des Clients erzwungen werden!

Haben Sie den Quellcode Ihres PlugIns erstellt, übersetzen Sie den Quellcode mit dem Java-Compiler in eine Class-Datei. Diese packen Sie dann zusammen mit der Klasse `AbstractClientPlugin` in eine Jar-Datei.

Bitte achten Sie darauf, dass die Klasse `AbstractClientPlugin` in der Jar-Datei im Pfad de/adito/aditoweb/common/jdito/plugin/impl liegt. Dieser Pfad spiegelt die Package-Struktur wider und darf nicht verändert werden.

Damit ist ihr PlugIn einsatzfähig.

Die Klasse `AbstractClientPlugin` muss in der Jar-Datei mitgepackt werden. Dadurch kann der ADITO Server beim Laden des PlugIns überprüfen, ob dieses noch versionskompatibel ist. Die erstellte Jar-Datei legen Sie in den Ordner lib\client\plugin in Ihre ADITO4 Distribution.

Nutzen Sie in Ihrem PlugIn externe Bibliotheken, die auch bereits in der ADITO4-Distribution enthalten sind, müssen diese Bibliotheken in der von ADITO mitgelieferten Version verwendet werden. Die Verwendung anderer Versionen dieser Bibliotheken ist nicht möglich!

Client-PlugIns, die über Java Webstart verteilt werden sollen, müssen signiert werden. Das kann entweder mit einem Zertifikat geschehen, das Ihr Unternehmen bereits im Einsatz hat oder auch mit einem selbst erstellten Zertifikat. Weitere Informationen zum Signieren von Jar-Dateien finden Sie in den unter "Verweise, Literatur, Links" angegebenen Quellen.

Da eine JNLP-Datei immer nur Jar-Dateien enthalten darf, die alle mit dem gleichen Zertifikat signiert wurden, brauchen Sie für Ihr PlugIn eine zweite JNLP-Datei, die Sie als Extension einbinden. Auch hierzu finden Sie Informationen in der Java WebStart-Dokumentation, die unter "Verweise, Literatur, Links" aufgelistet ist.

Weitere zusätzlich verwendete Bibliotheken müssen in den Klassenpfad (und damit ggf. in die JNLP-Datei) aufgenommen werden.

4. PlugIns aufrufen

4.1. Server-PlugIns

Server-PlugIns rufen Sie aus JDito mit der Methode `run` des Moduls `plugin` auf.

```
Object[] run(String jarURL, String className, Object[] arguments);
```

jarURL

URL des Plugin-Jar-Files, kann null sein, wenn das Jar-File bereits im Klassenpfad enthalten ist. Die Aufnahme des PlugIns in den Klassenpfad ist der bevorzugte Weg. Die direkte Angabe beim Aufruf des PlugIns funktioniert z.B. unter Linux nicht.

className

Der vollständige Klassenname des PlugIns.

arguments

Kommandozeilenparameter für das PlugIn als String-Array.

return

Die vom Plugin zurückgegebenen Ergebnisse.

throws

PluginException bei Fehlern.

Code-Beispiel:

```
// run the plugin contained in testplugin.jar
var ret = plugin.run("file:///c:/plugin/testplugin.jar",
                 "de.adito.aditoweb.plugin.test.TestPlugin",
                 new Array("arg1", "arg2", "arg3") );

// display the number of array elements returned
question.showMessage(liste.length);
```

4.2. Client-PlugIns

Client-PlugIns rufen Sie aus JDito mit der Methode `swing.doClientIntermediate` mit der Option `CLIENTCMD_RUNPLUGIN` auf:

```
Object doClientIntermediate(int type, String[] details);
```

type

Der Typ des Client-Kommandos, hier: `CLIENTCMD_RUNPLUGIN`.

details

details[0]: Pfad zum PlugIn-Jar am Client, optional, ist die PlugIn-Klasse im Klassenpfad enthalten kann hier `null` übergeben werden.

details[1]: Der vollständige Klassenname des PlugIns.

details[2] – details[x]: Parameter für das PlugIn.

return

Die Ergebnisse des PlugIns als String[].

throws

PluginException bei Fehlern.

Code-Beispiel:

```
// run the plugin contained in testplugin.jar
var details = new Array();
details[0] = "c:/plugin/testplugin.jar";
details[1] = "de.adito.aditoweb.plugin.test.TestPlugin";
details[2] = "arg1";
details[3] = "arg2";
var liste = swing.doClientIntermediate(swing.CLIENTCMD_RUNPLUGIN,
details);

// display the number of array elements returned
question.showMessage(liste.length);
```

5. Zugriff auf weitere Methoden der PluginFacade

In der Methode `execute` des PlugIns wird eine Instanz der PluginFacade als Parameter übergeben.

5.1. Logging

Auf der PluginFacade-Instanz können Sie drei Methoden zum Loggen von Informationen in die Log-Dateien von Server und Client aufrufen.

Diese Methode existiert mit folgenden Signaturen:

- `log(Throwable pException, int pID, Object pDetails, boolean pShowDialog)`
- `log(Throwable pException, Object pDetails, boolean pShowDialog)`
- `log(Object pDetails, boolean pShowDialog)`

pException

Das Exception-Objekt, das geloggt werden soll.

pID

Die ID der Log-Meldung.

Die Übergabe einer ID mit dem Parameter `pID` wird derzeit in der PlugIn-Schnittstelle nicht ausgewertet. Benutzen Sie daher immer die Methoden mit zwei oder drei Parametern!

pDetails

Ein Object mit den Details für die Log-Meldung. Für die Ausgabe wird die Funktion `toString` aufgerufen. Hier kann auch ein Object-Array übergeben werden. Ist dies der Fall wird für die Ausgabe die Methode `toString` auf alle Elemente des Arrays aufgerufen. Werden keine Details angegeben, kann hier `null` übergeben werden.

pShowDialog

`true` oder `false`, gibt an, ob eine Fehlermeldung am Client angezeigt werden soll oder nicht. Wird am Client eine Fehlermeldung angezeigt werden die in `pDetails` übergebenen Informationen in der Detailansicht der Fehlermeldung angezeigt.

Der Zugriff auf diese Logging-Methoden ist sowohl über die Server- als auch über die Client-PluginFacade möglich. Der Eintrag erfolgt je nach Art des PlugIns in die Server-Logdatei oder in die Client-Logdatei.

Die Log-Meldungen der PlugIn-Schnittstelle werden in den Log-Dateien auf dem Log-Level 'R' geloggt.

Beispiel 1

PlugIn-Code:

```
(...)


public class DemoPlugin extends AbstractPlugin
{


  (...)


  public Object[] execute(Object[] pParameters, IPluginFacade
pPluginFacade)
      throws PluginException
  {
    try
    {
 (...)
     throw new Exception("Exception-Message");
    }
    catch (Exception ex)
    {
     // Logging
     pPluginFacade.log(ex, new String[]{"Detail1", "Detail2",
"Detail3"}, true);


     return new String[]{"Exception"};
    }
  }
}
```

Ergebnis:

```
Z-00-N-0011-S Ursprüngliche Java-Exception die den Fehler auslöpste.
[ID 1]  [->]
 Exception: java.lang.Exception  [->] Message: Exception-Message
   J-39-R-0000-S Fehler im Plugin  [->] Detail1  [->] Detail2  [->]
Detail3
```

Beispiel 2

```
(...)
pPluginFacade.log(ex, "Detail", true);
(...)
```

Beispiel 3

```
(...)
pPluginFacade.log("Detail", true);
(...)
```

Beispiel 4

```
(...)
pPluginFacade.log(new String[]{"Detail1", "Detail2", "Detail3"},
true);
(...)
```

5.2. Datenbankzugriff

Der Datenbankzugriff über PlugIns ist nur mit Server-PlugIns möglich!

Hierfür steht Ihnen die Methode `doDBRequest` der PluginFacade mit folgender Signatur zur Verfügung:

```
ArrayList doDBRequest(String pAlias, String[] pSQLStatements, int
pStart, int pCount, boolean pGetMetaData, boolean pWithLobContent,
boolean pWithRowCount)
```

pAlias

Der DB-Alias, auf dem die Statements ausgeführt werden sollen.

pSQLStatements

Die auszuführenden Statements als String-Array.

pStart

Für ein Select-Statement. Der Datensatz, ab dem Datensätze zurückgegeben werden sollen.

pCount

In Verbindung mit `pStart`. Die Anzahl Datensätze, die zurückgegeben werden soll. Hier kann das Feld `DBREQUEST_EOF` genutzt werden. Dies steht für alle Datensätze.

pGetMetaData

Geben Sie hier immer `false` an. Die ermittelten Informationen sind im Moment im PlugIn nicht lesbar!

pWithLobContent

Geben Sie hier immer `false` an. Derzeit wird im PlugIn sonst nur ein BlobHandler-/ClobHandler-Objekt zurückgegeben, mit dem Sie nicht weiterarbeiten können!

pWithRowCount

Geben Sie hier immer `false` an. Derzeit können Sie mit diesem Parameter nichts anfangen!

return

Das Ergebnis des letzten Statements aus `pSQLStatements`. Bei einem Select-Statement wird als Ergebnis das RecordSet als `ArrayList` (mit Object[] für jede Zeile) zurückgegeben. Bei Update-, Insert- und Delete-Statements wird die Zahl der betroffenen Datensätze zurückgeliefert. Bei Fehlern wird eine `PluginException` geworfen.

Beispiel

PlugIn-Code:

```
package de.adito.plugin;

import java.util.ArrayList;

(...)

/**
 * Demo-Plugin
 *
 * @author ADITO Software GmbH, 2007
 */



public class DemoPlugin extends AbstractPlugin
{
  (...)

  public Object[] execute(Object[] pParameters, IPluginFacade
pPluginFacade)
```

```
      throws PluginException
  {
    try
    {
     // DB-Zugriff
 String[] stmts = new String[2];
 stmts[0] = "select count(*) from org";
 stmts[1] = "select orgid, orgname from org where orgid < 1010";
 ArrayList ergebnis = pPluginFacade.doDBRequest("AO_DATEN", stmts,
0, -1, false, false, false);

 return ergebnis.toArray();
    }
    catch (Exception ex)
    {
     (...)
    }
  }
}
```

PlugIn-Aufruf:

```
// Kann null sein, wenn das Plugin im Classpath enthalten ist
var url = "file:///E:/java/DemoPlugin/DemoPlugin.jar";
var klasse = "de.adito.plugin.DemoPlugin";

var pluginAntwort = plugin.run(url, klasse, null)

var string = "";
for (var i=0; i<pluginAntwort.length; i++)
{
 string = string + pluginAntwort[i].join("|") + "\n";
}

question.showMessage(string);
```

5.3. Aufruf von JDito-Prozessen

Der Aufruf von JDito-Prozessen über PlugIns ist nur mit Server-PlugIns möglich!

Hierfür steht Ihnen die Methode `runJDitoProcess` der PluginFacade mit folgender Signatur zur Verfügung:

```
String runJDitoProcess(String pProcessName, Map pLocalVariables)
```

pProcessName

Der Name des JDito-Prozesses der ausgeführt werden soll.

pLocalVariables

Eine Map (Key-Value-Paare) mit den Parametern für den JDito-Prozess. Die Keys können im JDito-Prozess über `$local.<key>` als lokale Variablen verwendet werden.

return

Der Rückgabewert des JDito-Prozesses. Wird im JDito-Prozess mit der JDito-Methode `result.string` o.ä. zurückgegeben.

Beispiel

PlugIn-Code:

```
(...)
import java.util.HashMap;

(...)

/**
 * Demo-Plugin
 *
 * @author ADITO Software GmbH, 2007
 */


public class DemoPlugin extends AbstractPlugin
{
   (...)
```

```
  public Object[] execute(Object[] pParameters, IPluginFacade
pPluginFacade)
      throws PluginException
  {
    try
    {
     // JDito-Prozess ausführen
     HashMap map = new HashMap();
     map.put("Param1", "Wert1");
     map.put("Param2", "Wert2");
     String ergebnis =
pPluginFacade.runJDitoProcess("pluginprozess", map);

     return new String[]{ergebnis};

    }
    catch (Exception ex)
    {
     (...)
    }
  }
}
```

JDito-Prozess:

```
var p1 = vars.getString("$local.Param1");
var p2 = vars.getString("$local.Param2");

var ergebnis = "Erster Parameter: " + p1 + ", Zweiter Parameter: " +
p2;

result.string(ergebnis);
```

PlugIn-Aufruf:

```
// Kann null sein, wenn das Plugin im Classpath enthalten ist
var url = "file:///E:/java/DemoPlugin/DemoPlugin.jar";
var klasse = "de.adito.plugin.DemoPlugin";
```

```
var pluginAntwort = plugin.run(url, klasse, null);

var string = pluginAntwort[0];

question.showMessage(string);
```

6. Webservice Plugins

Über den ADITO4 Manager ist es möglich, Server-Plugins zu erstellen (siehe 2.1), welche eine Schnittstelle zu externen Webservices herstellen.

Wie Sie diese Plugins erstellen können und damit arbeiten finden Sie im ADITO-Informationsdokument AID059-DE - Webservices in ADITO.