

Anleitung Transit

Ulrich Hilger

Version 3.1, 16. März 2021

Inhalt

1. Java-Server	1
1.1. Anwendungsfälle	2
1.2. Einfacher Rückgabewert	3
1.3. Einfacher Parameter	3
1.4. Rückgabe eines Objekts	4
1.5. Objekt als Parameter	4
2. Webclient	6
3. Voraussetzungen	8
4. Installation	8
5. Konfiguration	8
6. Sicherheit	9
6.1. Berechtigungen via Filter	10
7. Dokumenthistorie	10

Transit macht die Funktionen einer Webanwendung für Aufrufe über HTTP zugänglich. Java-Code auf einem Server lässt sich mit Transit ohne statische Annotationen in Webclients nutzen. Dieses Dokument beschreibt die Verwendung von Transit.

1. Java-Server

Für eine Ausführung von Java-Code auf einem Server wie z.B. Tomcat, Jetty, Glassfish, Wildfly, Undertow, Payara, usw. muss dieser Java-Code in Form von Methoden einer Java-Klasse bereit stehen. Eine solche Klasse kann zum Beispiel als Datei `TransitDemo.java` wie folgt angelegt sein.

Der Quellcode der Java-Klasse TransitDemo

```
package de.uhilger.demo.api;  
  
public class TransitDemo {  
  
    public String hallo() {  
        return "Hallo Welt!";  
    }  
  
}
```

Der obige Quellcode der Klasse wird zu `TransitDemo.class` kompiliert und in einen Ordner namens `TransitDemo` mit folgendem Inhalt überführt.

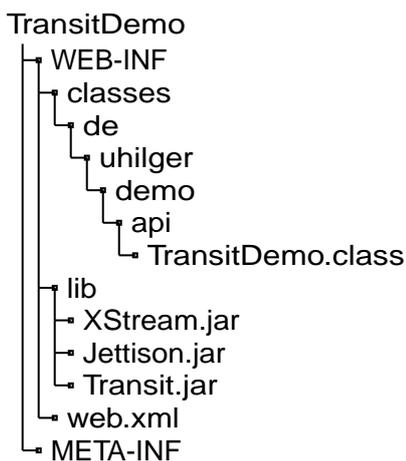


Abb. 1. Web-Archiv-Ordner

Der Ordner `lib` des Web-Archivs enthält die Klassenbibliotheken `XStream`, `Jettison` und `Transit` (vgl. [Voraussetzungen](#)). Im Deployment Descriptor, der Datei `web.xml`, wird folgendes eingetragen:

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app version="3.0" xmlns="http://java.sun.com/xml/ns/javaee"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
http://java.sun.com/xml/ns/javaee/web-app_3_0.xsd">
  <description>Demo-Anwendung fuer Transit</description>
  <display-name>transit-demo</display-name>
  <servlet>
    <servlet-name>Transit</servlet-name>
    <servlet-class>
      de.uhilger.transit.web.TransitServlet
    </servlet-class>
    <init-param>
      <param-name>klassen</param-name>
      <param-value>
        de.uhilger.demo.api
      </param-value>
    </init-param>
  </servlet>
  <servlet-mapping>
    <servlet-name>Transit</servlet-name>
    <url-pattern>/api</url-pattern>
  </servlet-mapping>
</web-app>
```

Der Ordner `TransitDemo` wird als ZIP-Archiv zur Datei `TransitDemo.war` gepackt. Das so entstandene Webarchiv kann nun auf jedem Java-Server wie z.B. Tomcat, Jetty, Glassfish, Wildfly, Undertow, Payara, usw. ausgeführt werden. Bei Tomcat gelingt die Ausführung beispielsweise mit Ablage der Datei `TransitDemo.war` im Ordner `webapps` von Tomcat.

Wenn Tomcat beispielweise auf der lokalen Maschine über Port 8080 läuft, kann die Methode `hallo` der Klasse `TransitDemo` im Browser mit folgendem URL gerufen werden:

```
http://localhost:8080/TransitDemo/api?c=de.uhilger.demo.api.TransitDemo&m=hallo&f=JSON
```

Die Antwort erscheint im Browser:

```
Hallo Welt!
```

Nach diesem Schema erstellter Java-Code kann in den Varianten der nachfolgend beschriebenen Anwendungsfälle verwendet werden.

1.1. Anwendungsfälle

Transit kann auf zwei Arten verwendet werden, mit dem `TransitServlet` und dem `TransitServletRS`.

Jedes der Servlets wird auf dem Webserver über einen Eintrag im Deployment Descriptor eingeschaltet wie es im vorangegangenen Kapitel beschrieben ist (vgl. auch [Konfiguration](#)).

Aufruf des `TransitServlet`

Das `TransitServlet` verarbeitet Aufrufe, bei denen Parameter im Query-Teil eines URL angegeben sind.

```
http://example.com/anwendung/api?c=package.Klasse&m=methodenname&f=JSON&p=param1&p=param2
```

Aufruf des `TransitServletRS`

Beim `TransitServletRS` werden die Parameter als Teil des Pfads ausgedrückt.

```
http://example.com/anwendung/api-rs/package.Klasse/methodenname/format/param1/param2
```

Varianten des Aufrufes

Transit unterscheidet hierbei die folgenden Arten von Aufrufen:

- einfacher Rückgabewert
- einfacher Parameter
- Rückgabe eines Objekts
- Objekt als Parameter

Für jeden der obigen Fälle ist nachfolgend ein Beispiel erläutert, das mit Aufrufen der beiden Klassen `TestKlasse` und `TestDatenKlasse` veranschaulicht, wie die Verwendung von Transit funktioniert. Allen Fällen gemeinsam ist, dass sowohl Parameter als auch Rückgabewert in Textform übertragen werden. Unabhängig von ihrem Datentyp wie beispielsweise Zahl, Text, Datum, Objekt werden sie also stets als Text ausgetauscht.

1.2. Einfacher Rückgabewert

Die einfachste Form der Nutzung von Transit ist der Aufruf einer Java-Methode ohne Parameter. Ein Beispiel für diese Art der Nutzung ist die Methode `TestKlasse.halloWelt`. Sie erwartet keine Parameter und gibt einfach die Zeichenkette `Hallo Welt` zurück.

URL für Aufruf mit `TransitServlet`

```
http://example.com/anwendung/api?c=de.uhilger.transit.TestKlasse&m=halloWelt
```

URL für Aufruf mit `TransitServletRS`

```
http://example.com/anwendung/api-rs/de.uhilger.transit.TestKlasse/halloWelt/JSON/
```

Rückgabewert

```
Hallo Welt
```

1.3. Einfacher Parameter

Ein anderes Beispiel ist die Methode `TestKlasse.gruss`. Sie erwartet einen Parameter, mit dem der

Name eines Benutzers angegeben werden kann. Als Beleg, dass der Parameter verarbeitet wurde sendet der Server als Rückgabewert die Zeichenkette "Hallo [name]!". Bei der Angabe des Parameters "Fred" lautet die Antwort des Servers "Hallo Fred!".

URL für Aufruf mit TransitServlet

```
http://example.com/anwendung/api?c=de.uhilger.transit.TestKlasse&m=gruss&p=Fred
```

URL für Aufruf mit TransitServletRS

```
http://example.com/anwendung/api-rs/de.uhilger.transit.TestKlasse/gruss/JSON/Fred
```

Rückgabewert

```
Hallo Fred!
```

1.4. Rückgabe eines Objekts

Sieht eine Java-Methode als Rückgabewert ein Java-Objekt vor, wird dieses von Transit als Text im JSON- oder XML-Format übermittelt. Ein Beispiel für diesen Anwendungsfall liefert die Methode `TestKlasse.getTestDaten`. Ihr werden drei einfache Werte als Parameter übermittelt, die der Server in ein Objekt der Klasse `TestDatenKlasse` verpackt welches an den Browser zurückgesendet wird.

URL für Aufruf mit TransitServlet

```
http://example.com/anwendung/api?c=de.uhilger.transit.TestKlasse&m=getTestDaten&p=16&p=testText&p=testName
```

URL für Aufruf mit TransitServletRS

```
http://example.com/anwendung/api-rs/de.uhilger.transit.TestKlasse/getTestDaten/JSON/16/testText/testName
```

Rückgabewert

```
{"TestDatenKlasse":{"id":16,"name":"testName","text":"testText"}}
```

Der Rückgabewert kann auf der Seite des Browsers mit der Funktion `JSON.parse` in ein JavaScript-Objekt verwandelt werden, dessen Typbezeichnung, Struktur und Inhalt mit denen des Java-Objekts identisch sind. Frameworks wie z.B. jQuery enthalten zudem Funktionen für Ajax-Aufrufe, die für Antworten im Format JSON direkt ein JSON-Objekt liefern (vgl. `getJSON`).

1.5. Objekt als Parameter

Erwartet die Methode einer Java-Klasse ein Objekt oder mehrere Objekte als Parameter, werden diese als Texte im JSON-Format übermittelt. Klasse und Struktur eines solchen Objekts sind gewöhnlich in der API-Dokumentation ersichtlich, wie es z.B. für die Klasse `TestDatenKlasse` von Transit der Fall ist. Sie wird von der Methode `TestKlasse.testObjektVerarbeiten` als Parameter erwartet. Mit den Informationen aus der Dokumentation kann das Objekt in JavaScript identisch angelegt werden, wie das folgende Beispiel zeigt.

die Klasse `TestDatenKlasse` in Javascript

```
function TestDatenKlasse(i, n, t) {
  this.id = i;
  this.name = n;
  this.text = t;
}
```

Die obige Deklaration der Klasse `TestDatenKlasse` kann verwendet werden, um ein Objekt dieser Klasse in Javascript zu erzeugen:

```
var testObjekt = new TestDatenKlasse(16, 'Fred', 'einText');
```

Das `testObjekt` wird dann als Parameter in einem Methodenaufruf über Transit verwendet. Es wird zu diesem Zweck zuvor in einen JSON-Ausdruck umgewandelt.

Die Hilfsfunktion `serialisieren` verwandelt ein Javascript-Objekt nach JSON

```
function serialisieren(obj) {
  return '{"' + obj.constructor.name + "':" + JSON.stringify(obj) + '"}';
}
```

Der anschließende Aufruf der Methode `TestKlasse.testObjektVerarbeiten` erfolgt ähnlich wie in der Beschreibung im Kapitel [Webclient](#) wie folgt

Aufruf einer Methode, die ein Objekt als Parameter erwartet

```
var xhr = new XMLHttpRequest();
xhr.onreadystatechange = function() {
  if (this.readyState === 4 && this.status === 200) {
    console.log(this.responseText);
  }
};

var url = 'http://example.com/anwendung/api';
var klasse = '?c=de.uhilger.transit.TestKlasse';
var methode = '&m=testObjektVerarbeiten';
xhr.open("POST", url + klasse + methode, true);
xhr.setRequestHeader('Content-type', 'application/x-www-form-urlencoded');

var testObjekt = serialisieren(new TestDatenKlasse(16, 'Fred', 'einText'));
xhr.send('p=' + testObjekt);
```

Im obigen Beispiel wird ein Javascript-Objekt `TestDatenKlasse` erzeugt und in die Variable `testObjekt` serialisiert. Die so serialisierte `TestDatenKlasse` wird per HTTP-POST als Parameter der Server-Methode `testObjektVerarbeiten` an den Server geschickt.

1.5.1. Einschränkungen

Beim Aufruf von Methoden, die Objekte als Parameter erwarten werden keine eindeutigen Klassennamen bestehend aus Package und Klasse verwendet. Es können daher nur Methoden gerufen werden, die als Parameter Objekte erwarten, deren Klassenname auch ohne Angabe der Package eindeutig ist.

2. Webclient

Auf der Seite eines Webclients werden Funktionen, die von einem Server bereitgestellt werden, über [asynchrone Funktionsaufrufe](#) genutzt. So ist es möglich, HTTP-Anfragen durchzuführen, während eine HTML-Seite angezeigt wird. Die Seite kann verändert werden ohne sie komplett neu zu laden. Das folgende Beispiel nutzt diese Möglichkeit zum Abruf der aktuellen Zeit von einem Server.

die Javascript-App `serverzeit.js` zum Abruf der Zeit von einem Server

```
function Serverzeit() {
    var self = this;

    this.abrufen = function(elem_id) {
        self.http_get("http://example.com/test/api/uhrzeit", self.antwort_zeigen,
elem_id);
    };

    this.antwort_zeigen = function (antwort, id) {
        var elem = document.getElementById(id);
        elem.textContent = antwort;
    };

    this.http_get = function (url, callback, id) {
        var xhr = new XMLHttpRequest();
        xhr.onreadystatechange = function() {
            if (this.readyState === 4 && this.status === 200) {
                callback(this.responseText, id);
            }
        };
        xhr.open("GET", url, true);
        xhr.send();
    };
}
```

Die obige App wird wie folgt in einer HTML-Seite verwendet.

eine HTML-Seite zur Anzeige der Zeit von einem Server

```
<html>
  <head>
    <title>Serverzeit-Demo</title>
  </head>
  <body>
    <p>Zeit: <span id="zeitanzeige">wird abgerufen...</span></p>
    <script src="serverzeit.js"></script>
    <script>
      var app;
      document.addEventListener('DOMContentLoaded', function () {
        app = new Serverzeit();
        app.abrufen('zeitanzeige');
      });
    </script>
  </body>
</html>
```

In manchen Fällen ist es nötig, die Methode HTTP-POST anstelle von HTTP-GET zu verwenden, beispielsweise, wenn Daten zum Server geschickt werden, deren Umfang zu groß ist, um als Parameter im Query-Teil des URL übergeben zu werden. In diesem Fall würde auf der Seite des Webclients ein Code wie folgt verwendet werden.

Variante eines Funktionsaufrufes mit HTTP-POST

```
this.http_post = function (url, data, callback, id) {
  var xhr = new XMLHttpRequest();
  xhr.onreadystatechange = function() {
    if (this.readyState === 4 && this.status === 200) {
      callback(this.responseText, id);
    }
  };
  xhr.open("POST", url, true);
  xhr.setRequestHeader('Content-type', 'application/x-www-form-urlencoded');
  xhr.send(data);
};
```

Im zusätzlichen Parameter `data` werden die Parameter übergeben, die die Server-Methode erwartet. Die Reihenfolge muss ihrer Deklaration entsprechen und zum Beispiel für eine Methode, die drei Parameter erwartet, wie folgt angelegt sein:

```
var data = "p=Parameter 1&p=Parameter 2&p=Parameter 3";
```

Wichtig ist zudem, dass im Header der Content Type wie im Beispiel gesetzt wird, sonst werden die Inhalte von `data` nicht richtig verarbeitet.

In den vorangegangenen Kapiteln ist beschrieben, wie Java-Methoden auf dem Server mit Hilfe von Transit für die hier gezeigten Aufrufe von Webclients zugänglich gemacht werden können.

3. Voraussetzungen

Transit setzt auf die Klassenbibliotheken XStream und Jettison zur dynamischen Serialisierung und Deserialisierung von Java-Objekten auf. Die Klassenbibliotheken müssen im Classpath enthalten sein wie nachfolgend beschrieben.

4. Installation

Damit eine Webanwendung die Funktionen von Transit nutzen kann müssen die folgenden Schritte ausgeführt werden

- Herunterladen [XStream](#) und [Jettison](#)
- Herunterladen von [Transit](#)
- Kopieren der Dateien `transit.jar`, `xstream.jar` und `jettison.jar` ins Verzeichnis `WEB-INF/lib` der Webanwendung

Die Klassenbibliotheken werden so mit der Webanwendung verteilt und stehen zur Laufzeit zur Verfügung.

5. Konfiguration

Mit einem Eintrag im Deployment Descriptor, der Datei `WEB-INF/web.xml` einer Webanwendung werden beliebige Java-Methoden für Webclients nutzbar. Der Eintrag lautet wie folgt

```
<servlet>
  <servlet-name>Transit</servlet-name>
  <servlet-class>
    de.uhilger.transit.web.TransitServlet
  </servlet-class>
</servlet>

<servlet>
  <servlet-name>TransitRS</servlet-name>
  <servlet-class>
    de.uhilger.transit.web.TransitServletRS
  </servlet-class>
</servlet>

<servlet-mapping>
  <servlet-name>Transit</servlet-name>
  <url-pattern>/api</url-pattern>
</servlet-mapping>

<servlet-mapping>
  <servlet-name>TransitRS</servlet-name>
  <url-pattern>/api-rs/*</url-pattern>
</servlet-mapping>
```

Damit werden Methodenaufrufe über die Endpunkte `/api` und `/api-rs` der Webanwendung eingerichtet. Läuft die Webanwendung beispielsweise auf dem Server `example.com` unter dem Kontext `anwendung`, gelangen mit obigem Eintrag Methodenaufrufe über die folgenden Uniform Resource Locators (URL)

<http://example.com/anwendung/api>

<http://example.com/anwendung/api-rs/>

6. Sicherheit

Zur sicheren Verwendung von Transit müssen Aufrufe beschränkt werden, damit nicht alle Klassen und Methoden auf einem Server zugänglich sind. Dem Transit-Servlet wird dazu ein Parameter im Deployment Descriptor mitgegeben, der Packages oder einzelne Klassen benennt. Nur die Methoden dieser Packages oder Klassen sind dann von außen verwendbar.

```
<init-param>
  <param-name>klassen</param-name>
  <param-value>
    com.example.anwendung.api;
    com.example.anwendung.web
  </param-value>
</init-param>
```

Mit der Trennung durch Semikolon können mehrere Packages oder Klassen angegeben werden. Das obige Beispiel lässt nur Aufrufe von Methoden zu, die sich in Klassen der Packages `com.example.anwendung.api` oder `com.example.anwendung.web` oder deren Subpackages befinden.

6.1. Berechtigungen via Filter

In vielen Fällen genügt es, Aufrufe von Transit auf bestimmte Java-Klassen oder -Packages zu beschränken, wie es im vorigen Abschnitt beschrieben ist. Sollen darüber hinaus anwendungsspezifische Berechtigungen geprüft werden, kann die Schnittstelle `RechtePruefer` verwendet werden. Eine Klasse muss für Berechtigungsprüfungen lediglich diese Schnittstelle implementieren und über einen Filter einbinden.

7. Dokumenthistorie

Tabelle 1. Änderungen

Nr	Datum	Autor	Beschreibung
1	11. Mai 2020	Ulrich	Neufassung dieser Doku in Version 3
2	16. März 2021	Ulrich	Version 3.1: Kapitel Java-Server und struktutelle Umstellungen