kurz & bündig	
Inhalt	Continuations mit RIFE
Zusammenfassung	Continuations in Web-Anwendungen mit RIFE
Quellcode mit angeliefert	Nein

Continuations mit dem Web-Framework RIFE

Web-Anwendungen einmal anders

von klaus meffert

RIFE ist ein innovatives Framework zur eleganten Erstellung von vollwertigen Web-Anwendungen. Zu den Features von RIFE gehören von Logik befreite HTML-Templates, das Paradigma *Convention over Configuration*, intuitive Workflow-Definitionen und eine agile Systemarchitektur, die schnelle Ergebnisse verspricht. Dieser Artikel diskutiert die Fähigkeit von RIFE, sogenannte Continuations für Web-Anwendungen zu realisieren.

Einleitung

Der Markt der Web-Frameworks im Java-Segment ist groß. Warum also ein Produkt wie RIFE [1]? Weil RIFE die Möglichkeit bietet, ohne große Einarbeitung in kurzer Zeit Web-Anwendungen zu entwickeln und dabei auf eine Vielzahl vorgefertigter Diensten zurückzugreifen. Zu diesen Diensten zählen insbesondere Datenbankerstellung mit einem Befehl aus POJOs heraus, ein Content Management System, eingängiges Verwaltung von Konfigurationen und, neben weiteren, auch Continuations.

Diese Dienste sind in RIFE integriert und stehen ohne Umschweife zur Verfügung. Das gelebte Konzept des Frameworks basiert darauf, dass der Entwickler so wenig wie möglich programmieren muss und so viel wie möglich vom Framework unterstützt wird. Während einige andere Software-Pakete dies nur bedingt schaffen, zeigt dieser Artikel anhand des Aspekts der Continuations, dass es mit RIFE tatsächlich funktioniert und zwar ohne großen Schulungsaufwand.

Continuations

Mit „Continuations“ wird ein Mechanismus bezeichnet, der es erlaubt, eine Web-Anwendung quasi sequentiell abzuarbeiten, ähnlich wie eine klassische Anwendung. Das ist insbesondere deswegen bemerkenswert, als dass eine Web-Anwendung immer aus Server- und Client-Seite besteht. Die Aufteilung einer Web-Anwendung in die Ausgabe einer Seite, Erwarten von Benutzeraktionen auf der Seite sowie Verarbeitung dieser Aktionen wird – oberflächlich betrachtet – sequentialisiert. Es gibt im Quelltext einer Web-Anwendung mit RIFE keine Brüche mehr, der Quelltext zur Ausgabe einer Seite bis hin zur Verarbeitung einer Benutzereingabe lässt sich in einer einzigen Methode kodieren. Eine Unterbrechung im Programmfluss durch eine notwendige Benutzeraktion im Browser fällt somit nicht mehr ins Gewicht. Außerdem ist es mit Continuations möglich, ein Zurückspringen zu einer schon prozessierten Seite zu ermöglichen und dabei den Zustand der wieder angesprungenen Seite vom Framework automatisch restaurieren zu lassen. Für die hierbei verlassene Seite wird der Zustand für ein eventuelles Wiederaufrufen ebenfalls automatisch konserviert. Eine tiefer geschachtelte Aufrufkette bereits besuchter Seiten ist auch möglich.

Basiskonzepte von RIFE

Zum besseren Verständnis des Artikels ist ein kurzer Anriss einiger Basiskonzepte von RIFE notwendig. Die zentralen Klassen in RIFE-Anwendungen sind die für den Aufbau einer Seite oder eines Seitenabschnitts zuständigen. Diese Klassen werden als Elemente bezeichnet und erben von *com.uwyn.rife.engine.Element*. Sie

stellen den Controller dar für Seiten oder Abschnitte, für die sie zuständig sind. In WebWork [4] wird das Pendant als *Action* bezeichnet. Ein Element besitzt zwei zugehörige Komponenten. Die erste ist die Definition der Ein- und Ausgabeparameter der Seite (meist über Formroutinen) sowie der Folgeseiten und einigem anderen mehr. Die zweite Komponente ist ein HTML-Template, in dem auf die Attribute der Elementklasse lesend oder schreibend (über Eingabefelder) zugegriffen wird. Elementklasse, Elementdefinition und Template werden vom Framework automatisch verknüpft und zusammengeführt. Ein in der Elementklasse deklariertes Attribut wird über dessen Getter automatisch ausgelesen, wenn im Template eine Variable mit demselben Namen vorhanden ist. Bei einem im Template definierten Eingabefeld, das denselben Namen besitzt wie ein Attribut der Elementklasse, wird der Setter des Attributs der Elementklasse gerufen. In der Elementklasse wiederum kann eine Variable im Template mit der Anweisung *setValue("variablenname", wert)* gesetzt werden.

Die Navigation zwischen zwei Seiten wird durch Exits definiert. Ein Exit hat einen Namen, ist in eine Elementdefinition eingebettet und besitzt ein Zielelement. Wird der Exit durch Aufruf der Methode *exit("exitname")* gefeuert, dann wird die Ereignisbehandlung des Zielelements angestoßen. Letztere sorgt für den Aufbau der Zielseite.

Eingebauter Support

Damit der Entwickler nicht nur von Routine-Aufgaben entlastet wird, sondern sie fast vergessen kann, bietet RIFE eine ganze Reihe von Ideen an. Das moderne Konzept der *Convention over Configuration* ermöglicht es dem Entwickler, unter Einhaltung einer simplen Nomenklatur dem Framework eigene Bestandteile anzumelden und somit hinzuzufügen. Durch die strikte Trennung von Logik und Inhalt entsteht das von JSP bekannte Dilemma der vermischten HTML-Seiten nicht. Lästige Arbeiten wie das programmatische Auslesen von Cookies sind obsolet. Es reicht die Deklaration einer Cookie-Variable im Template und ein einziger Methodenaufruf zum späteren Auslesen (falls notwendig und nicht eine von RIFE angebotene Logik verwendet wird, etwa zur Benutzerverwaltung).

Die Funktionsfähigkeit einer RIFE-Anwendung lässt sich durch Unit Tests, die außerhalb des Containers laufen können, ohne zeitaufwändiges Anstarten des Web-Servers bestens (ohne Selenium) sicherstellen.

Wie funktionieren Continuations in RIFE?

Von Hause aus ist das Übertragungsprotokoll HTTP zustandslos. Das macht die Verwaltung von Zuständen nicht einfacher. Auch der asynchrone Programmfluss von Web-Anwendungen wirkt sich erschwerend für die manuelle Implementierung von Continuations aus. Es wäre sehr mühselig, den Zustand einer Anwendung manuell zu konservieren und das Laden dieses Zustands bei Bedarf mittels eigener Routinen durchzuführen. Continuations in RIFE nehmen dem Entwickler nahezu sämtliche Arbeit ab. Mit einer einzelnen Anweisung kann der Anwendungszustand weggeschrieben werden, so dass er nach einer Unterbrechung wieder zur Verfügung steht. Das Framework merkt sich dazu relevante Teile des Stack-Inhalts. Dazu gehören insbesondere lokale Variablen und der Aufrufpfad.

Continuations werden in RIFE konkret durch Bytecode-Manipulation von Elementklassen mit Hilfe des Frameworks ASM [3] realisiert. Denn mit proprietären Java-Mitteln kann der Zustand einer beliebigen Anwendung zu einem willkürlichen Zeitpunkt nicht ermittelt werden. Durch die Bytecode-Manipulation wird ein Mechanismus in die Anwendung injiziert, der eben genau das doch erlaubt. Bestandteil der Zustandsverwaltung ist das Festhalten der Werte aller im Kontext der Continuations-Klassen deklarierten Variablen.

Listing 1 zeigt, wie einfach die Umsetzung von Continuations mit RIFE ist. Im wesentlichen handelt es sich um eine intuitiv anmutende *while*-Schleife, in der die ganze Anwendungslogik abläuft. Eine einzige Anweisung ist letztendlich notwendig, um eine Continuation zu ermöglichen. Diese Anweisung lautet *pause()*. Mehr ist es nicht. RIFE durchsucht vor dem Ausführen einer Elementklasse deren Bytecode nach dem *pause()*-Befehl. Wird

er gefunden, ist die Continuation identifiziert, der oben genannte Mechanismus der Bytecode-Manipulation wird einmalig durchgeführt und ein anderer *Class Loader* wird gezogen. Im anderen Fall verhält sich RIFE wie ein konventionelles Web-Framework.

Multiple Continuations

RIFE bietet prinzipiell die Möglichkeit, auf einer Seite mehrere Unterseiten (Elemente) zu verwirklichen. Diese werden auch von RIFE Continuations unterstützt. Wichtig zu wissen ist hierbei, dass ein Browser-Request (ausgelöst etwa durch das Betätigen eines Buttons durch den Anwender) die Ereignisbehandlung aller Unterseiten von Continuations-Elementen aktiviert. Jedes Element sollte also herausfinden kann, ob der Request für sich oder statt dessen für ein anderes Element relevant ist.

Dazu kann im Ereignisbehandler eines Elements (Methode *processElement*) die Methode *hasSubmission("requestname")* befragt werden. Anstelle von *requestname* ist der Name des Requests einzusetzen, für den überprüft werden soll, ob er vorliegt. Die Idee dahinter ist, dass jede Unterseite unterschiedlich benannte Requests hat. Wäre das nicht so, könnte eine Ermittlung der Zuständigkeit für einen vorliegenden Request auf die beschriebene Weise nicht durchgeführt werden.

Ein unschöner Nebeneffekt tritt bei mehrfachen Continuations-Elementen auf, wenn Eingabefelder dargestellt werden. Eingabefelder anderer, von einem Request nicht betroffener Elemente werden initialisiert, wenn ein Request über den Browser abgesetzt wird. Um das zu verhindern, muss zusätzliche Logik in die beteiligten Elemente gesteckt werden. Das artet allerdings etwas in Bastelei aus und kann hier nicht weiter diskutiert werden.

Continuations über mehrere Seiten hinweg

Genauer gesagt, müsste die Überschrift dieses Abschnitts heißen „Continuations über mehrere Elemente hinweg“. Denn ein Element stellt eine Seite oder einen Abschnitt dar und es ist auch möglich, dass auf einer Seite ein Element ein anderes Element derselben Seite aufruft.

Listing 2 zeigt, wie einfach der Interelementaufruf in RIFE möglich ist. In diesem Kontext sind nur zwei Anweisungen wichtig. Die erste Anweisung, *call*, ruft über einen sogenannten Exit dessen Name als Parameter übergeben wird, das zweite Element auf. Das zweite Element verrichtet nun seine Arbeit und übergibt mittels der Anweisung *answer* die Kontrolle zurück an den Aufrufer. Ein optionaler Übergabeparameter erlaubt die Übermittlung einer Information an den Aufrufer, im Beispiel ist das ein Text. Nach Ausführen des Beispiels lautet die Ausgabe über die *print*-Befehle:

```
Start
Ziel
Antwort: Meine Antwort lautet 42
```

Würde man ohne Continuations arbeiten, wären *call* und *answer* durch den *exit*-Befehl zu ersetzen. Im Falle von *answer* könnte allerdings über diesen Weg kein Rückgabewert mitgegeben werden. Stattdessen müsste der Name des Ausgangs, der das Zielelement definiert, spezifiziert und ein Rückgabewert manuell durchgereicht werden.

Step-back Continuations

Eine sogenannte Step-back Continuation ist in gewisser Hinsicht das Gegenteil einer Continuation im herkömmlichen Sinne. Sie erlaubt es, anstatt wie gewöhnlich vorwärts zu navigieren, Seiten wieder zu besuchen, die schon verarbeitet worden sind. Ein schönes veranschaulichendes Online-Beispiel hierfür ist in [2] zu finden. Der Nutzen von Step-back Continuations offenbart sich besonders bei mehrseitigen Dialogschritten, wie im Beispiel zu sehen. Dann nämlich kann der Anwender zurück zu seinen zuvor gemachten Eingaben springen, die von RIFE automatisch konserviert werden. Außerdem ist ein Weiterspringen von einer durch

Rückwärtsnavigieren erreichten Seite derart möglich, dass die nun erneut aufgerufene Seite ebenfalls ihren bisherigen Zustand besitzt. Alle Benutzereingaben sind also wieder verfügbar!

Der Zustand von schon längst abgearbeiteten Seiten ist mit Hilfe von RIFE Continuations immer noch präsent, sofern der weiter oben beschriebene Mechanismus des Klonens umgesetzt wird. Dann nämlich merkt sich RIFE alle Anwendungszustände bis hin zum aktuellen (im nächsten Abschnitt ist beschrieben, wie der Continuations-Speicher geleert werden kann).

Um eine Schritt zurück im Anwendungsfluss zu gehen, muss lediglich der Befehl *stepBack()* aufgerufen werden. Der Zurück-Button des Browsers ist hier allerdings nicht relevant. Eine Elementklasse erkennt einen Rücksprung durch Abfrage der Methode *duringStepBack()*, die *true* im Rücksprungfall zurückliefert.

Continuations begrenzen

Innerhalb einer Anwendung, die Continuations unterstützt, kann es erforderlich sein, dass eine Seite nicht mit mehrmals mit demselben Zustand aufgerufen wird. Beispielsweise dann, wenn eine Zahlung durch Betätigen eines Buttons angestoßen wird. Ein doppeltes Ausführen dieser Funktion wäre fatal. Um das im Rahmen von Continuations zu verhindern, muss nur der gemerkte Kontext der Continuations entfernt werden. Das geschieht durch Aufrufen des Befehls *ContinuationContext.getActiveContext().removeContextTree();*. Es können auch Vaterkontexte statt des aktiven entfernt werden. Der Entwickler hat also eine feinstufige Kontrolle über die Gültigkeit von Continuations,

Debugging mit Continuations

Ein angenehmer Nebeneffekt von Continuations ist das Debug-Verhalten der Web-Anwendung. Die Anwendung lässt sich in den durch Continuations abgebildeten Teilen genauso debuggen wie eine klassische Anwendung. Das bedeutet, im Debugger kann über die Logik, die den Aufbau einer Webseite und die Verarbeitung der Benutzerantwort vornimmt, Schritt für Schritt gesprungen werden. Der Quelltext aus Listing 1 kann also ganz klassisch, nämlich sequentiell im Debugger ausgeführt werden. Das Debug-Verhalten einer Continuations-Anwendung ist vergleichbar mit dem Debugging einer gewöhnlichen Java-Anwendung. Unterschiede sind festzustellen, wenn mehrere Continuations-Elemente auf einer Seite im Browser dargestellt werden. Dann nämlich werden bei einem Request alle Elemente hintereinander aufgerufen. Im Debugger können die einzelnen Anweisungen jedes Elementes ausgeführt werden, bis irgendwann das für den Request zuständige Element an der Reihe ist.

Arbeit für den Entwickler

Damit Continuations den Zustand von lokalen Variablen der Ereignisbehandlung intern zwischenspeichern kann, müssen diese das unterstützen (Klonen oder Serialisierung). Zur Vereinfachung für den Entwickler stellt RIFE mit *com.uwyn.rife.tools.ObjectUtils* eine Hilfsklasse zur Verfügung. Sie bietet die Methoden *deepClone(Object)* und *genericClone(Object)*. Die erste Methode erstellt tiefe Kopien von Standardklassen und Arrays, die zweite Methode unterstützt Implementierungen von *java.lang.Cloneable*. Ein anderer Weg ist das Überschreiben der *clone()*-Methode der Element-Klasse, in der die Ereignisbehandlung stattfindet. In einer eigenen Implementierung können die Felder ausgespart werden, die für Continuations nicht relevant sind, also nicht geklont werden müssen. Anstelle dessen kann aus der Ereignisbehandlung eine Methode aufgerufen werden, die das Objekt instantiiert bzw. dessen Instanz zurückliefert, anstatt dass das Objekt direkt in der *processElement()*-Methode anzusprechen. Denn nur innerhalb der letztgenannten Methode deklarierte Objekte werden für das Klonen berücksichtigt.

Gesetzt den Fall, dass nur die aktuelle Continuations-Instanz verfügbar sein muss und Step-back Continuations nicht möglich sein sollen, gibt es eine andere, elegante Möglichkeit. In der Continuations

Elementklasse wird eine neue Methode *cloneContinuations()* eingeführt, die den Wert *false* zurückliefert und sonst nichts tut. Somit weiß RIFE, dass es nicht gewünscht/notwendig ist, den aktuellen Anwendungszustand zu klonen. In diesem Fall gibt es auch keinen Zugriff auf vergangene Anwendungszustände, sondern nur auf den aktuellen. Sichtbar wird das bei einem Refresh im Browser: Liefert *cloneContinuations()* den Wert *false* zurück, wird – wie das ohne Continuations auch der Fall wäre – eine neue Instanz der Seite aufgebaut. Ist der Rückgabewert *true* wird die vor dem Refresh gezeigte Instanz wieder hervorgezaubert.

Fazit

Das vorgestellte Framework RIFE ist durch die Vielzahl integrierter Dienste und Funktionen sehr leistungsfähig. Darüber hinaus sind spannende Erweiterungen wie besserer Ajax-Support zu erwarten. Es stellt eine echte Alternative zu populären Frameworks dar, die sicher komplizierter in der Handhabung sind. Das ausgeklügelte Zusammenspiel mit anderen Frameworks wie OpenLaszlo erweitert den Wirkungskreis von RIFE. Hochwertige Webanwendungen mit einer Vielzahl vorfabrizierter Dienste sind möglich. Der Entwickler kann sich auf seine Kernarbeit konzentrieren, lästige Routinearbeiten werden obsolet. Eine Reihe von öffentlich zugänglichen Anwendungen zeigt das Potential von RIFE. Erfreulich, dass Rapid Prototyping und testgetriebene Entwicklung von Webanwendungen durch RIFE gefördert werden. Continuations mit RIFE ermöglichen eine intuitive Entwicklung von Web-Anwendungen, wenngleich nicht davon auszugehen ist, sehr lastintensive Produkte damit in wichtigen Teilen betreiben zu können.

Es bleibt abzuwarten, wie die Weiterentwicklung etablierter Frameworks durch die Vorzüge von RIFE beeinflusst wird. Momentan ist es so, dass andere Produkte RIFE bereits integrieren oder unterstützen. WebWork [4] etwa nutzt Continuations von RIFE bereits. Das RIFE-Team ist zudem bestrebt, bewährte Technologien zu assimilieren. So sollen Continuations in Zukunft durch Terracotta DSO abgebildet werden.

Klaus Meffert ist als Organisationsberater der Brandt & Partner GmbH im SAP-Bereich tätig. Er arbeitet parallel an seiner Doktorarbeit zum Thema Software-Entwicklung. Weiterhin engagiert er sich im Open-Source Bereich sowie als Fach- und Buchautor. www.klaus-meffert.de

Links & Literatur

[1] RIFE: http://rifers.org/

[2] Informationen und Links zu RIFE: http://www.klaus-meffert.de/rife.html

[3] ASM: http://asm.objectweb.org/

[4] WebWork: http://www.opensymphony.com/webwork/

Listing 1: Einfaches Beispiel für Continuations

```
public class ContinuationDemo extends Element {
  // Ereignisbehandlung
  public void processElement(){
    Template template = getHtmlTemplate("mytemplate");
    ...
    // Es folgt eine sequentiell anmutende Schleife!!!
    while (!finished) {
      ...
      print(template);//Ausgabe HTML-Seite
      pause(); //Continuations-Befehl
      int answer = ... //aus Benutzereingabe ermitteln
      if (answer == solution) {
        finished = true;
      }
      else {
        tries++;
      }
    }
  }

}
```

Ende Listing

Listing 2: Continuations über mehrere Seiten hinweg

```
// Erstes Element
public class Call extends Element {
  public void processElement() {
    print("Start");
    // Aufruf von zweitem Element
    String antwort = (String)call("callexit");
    print("Antwort: " + antwort);
  }
}

// Zweites Element
public class CallTarget extends Element {
  public void processElement() {
    print("Ziel");
    // Rücksprung zu erstem Element
    answer("Meine Antwort lautet 42");
  }
}
```

Ende Listing