ColdFusion

Generell

- ColdFusion-Dateien werden generell klein geschrieben - ausser Application.cfm, Application.cfc und OnRequestEnd.cfm.
- CF-Tags werden klein geschrieben, CF-Funktionen in der CamelCase-Notation.
- Bei zusammengesetzten Dateinamen ist die CamelCase-Schreibweise zu verwenden (Bsp: userService.cfc).
- Verzichte auf den "Gartenhag" # innerhalb von ColdFusion-Code. Das Rautenzeichen wird so gut wie nie benötigt.
- Vermeide Zeilen, die länger als 80 Zeichen sind. Wenn ein Ausdruck nicht auf einer Zeile untergebracht werden kann, ist es ratsam, das Statement auf mehrere Zeilen aufzuteilen. Der Umbruch sollte dabei nach einem Operator eingefügt werden. Die folgende Zeile soll um eine weitere Einheit eingerückt werden.
- ColdFusion-Code wird sauber formatiert, das heisst, es werden Einrückungen mit Tabulatoren verwendet.
- Halte den Code einfach - einfacher Code ist einfacher zu verstehen und debuggen.
- Wenn eine vorhandene Datei angepasst wird, soll der vorgefundene Programmierstil übernommen werden, es sei denn, die Applikation wird grundsätzlich überarbeitet.
- Whitespace ist grundsätzlich zu unterdrücken: cfsetting enablecfoutputonly und output=false. Die Aktivierung des Whitespace-Managements im ColdFusion-Administrator ist nicht empfohlen, einerseits aus Performance-Gründen und andererseits zur Erhöhung der Portabilität (Deployment) einer Applikation.
- Bei der Ausgabe von Content, der von Benutzern erstellt worden ist, ist besondere Vorsicht geboten. In JavaScript-Bereichen ist die Funktion JSStringFormat() zu verwenden, in Formularen und allgemein HTML-Attributen wird mit HTMLEditFormat() gearbeitet.

Sicherheit

- Client-seitige Überprüfung mit JavaScript ist nett, ersetzt aber nicht die Server-seitige Überprüfung von Eingaben.
- Nicht mehr verwendeter Code ist zu entfernen, damit er nicht versehentlich ausgeführt wird.
- In Datenbank-Abfragen ist immer der ColdFusion-Tag "<cfqueryparam />" zu verwenden. Bei TOP / LIMIT Einschränkungen funktioniert der Tag nicht, die Werte sind abzusichern mit Int(Val(request.top)).
- Debugging auf Produktivservern ist grundsätzlich nicht aktiviert und darf nur mit IP-Restriktion aktiviert werden. Dabei ist zu beachten, dass Applikationen unterschiedliche Caching-Methoden verwenden - Debug-Informationen dürfen nicht gecacht werden!
- Bei folgenden Tags sind bei den Werte-Parametern CRLF zu entfernen mit ReReplace(value, "[\r\n]", "", "ALL"):
    - cfheader
    - cfcontent (type Attribut)
    - cfmail
    - cfmailparam
    - cfmailpart

- Der cflocation-Tag erhält immer das Attribut addtoken=false
- Der cftoken ist mit einer UUID zu erstellen (Einstellung im CF-Administrator)
- Nach Möglichkeit sind J2EE-Sessions zu verwenden.

Fehlerbehandlung

- Fehlermeldungen dürfen keine Sicherheitsrelevanten Informationen enthalten. Dies gilt generell für Fehler, aber auch für nicht gefundene Seiten und Templates.
- ColdFusion-Fehlermeldungen gelangen nicht 1:1 zum Benutzer der Applikation.
- Auf Stufe Applikation sind Fehler mit dem ColdFusion-Tag "cferror" oder onError() in Application.cfc abzufangen.
- Innerhalb der Applikation sind Try/Catch-Funktionen zur Fehlerflusssteuerung einzusetzen.

Variablen

- Variablen werden immer (mit Ausnahme des Variables-Scopes) mit dem vollständigen Pfad (Scope) aufgerufen.
  Beispiel:

```
<cfoutput>
#client.user.name#
#session.user.name#
#url.username#
#form.username#
</cfoutput>
```

- Variablennamen sind kurz, treffend und sprechend zu wählen. ColdFusion behandelt Variablennamen nicht Case-sensitiv, weshalb der CamelCase-Stil bevorzugt wird.
- Variablennamen beginnen mit einem Zeichen aus [a-zA-Z] und enthalten keine Sonderzeichen.
- Um vom Namen auf die Art der Variable zu schliessen, haben sich folgende Präfixe eingebürgert:

Typ	**Präfix**	**Pflicht**	**Beispiel**
Struktur	stc	*	stcUser
Array	ary	*	aryData
Xml-Date	xml	*	xmlData
Query	qry	*	qryUser
Datum	dte	*	dteEvent
Zeit	tme	*	tmeEvent
Boolean	idx, is, has	*	idxEmail, isConfirmed, hasEmail
Component	obj	*	objCache
Integer, Numeric	int		intAnzahl
String	str		strNewContent

- Allgemein bekannte Abkürzungen werden gross geschrieben (Bsp: ID,UID usw).
- Formularfeldnamen sollten der Bezeichnung für Datenbankfelder oder Objekteigenschaften entsprechen. Dies dient der Übersicht und der einfacheren Verarbeitung von Formulardaten.

Funktionen

- Funktionen erfüllen in der Regel einen einzigen Zweck.
- Funktionen enthalten Methoden zur Validierung von Daten und allfälliger Fehlerbehandlung.
- Funktionen sind möglichst abstrakt zu halten um eine Wiederverwendbarkeit zu ermöglichen.
- Lokale Variablen innerhalb von Funktionen sind immer in den Var-Scope zu setzen. Sind mehrere lokale Variablen von Nöten, wird empfohlen, eine lokale Struktur zu initialisieren:

```
<cffunction name="getUser" access="public" output="false" returntyp
<cfargument name="userID" default="" required="true" />
<cfset var local = StructNew() />
<cfset local.hasUser= false />
<!---  Some code --->
<cfreturn local />
</cffunction>
```

  ab CF9 (Adobe) und Railo 3.2 kann auf den local-Scope zugegriffen werden. Diese Struktur muss nicht explizit initialisiert werden:

```
<cffunction name="hasUser" access="public" output="false" returntyp
<cfargument name="userID" default="" required="true" />
<cfset local.hasUser= false />
<!---  Some code --->
<cfreturn local.hasUser />
</cffunction>
```

- Zu Überprüfung der Scopes wird die frei verfügbare Applikation varScoper eingesetzt.
- In aller Regel sind Funktionen nicht für den direkten Output vorgesehen. Deshalb ist das Attribut "output" explizit auf false zu setzen.
- Zur einfachen Dokumentation werden die hint-Attribute beim cffunction-Tag und sofern vorhanden auch bei cfargument gesetzt.
- Die Verwendung des Type-Attributes beim returntype ist sinnvoll, sofern nicht "any" benutzt wird.
- Von der Verwendung des Type-Attributes bei cfargument ist bei Systemen abzuraten, bei denen die Typen-Validierung nicht explizit deaktiviert werden kann. Die Validierung des Datentyps erfolgt zudem innerhalb der Funktion, um eine optimale Fehlerbehandlung zu ermöglichen.
- Bei der Benennung der Funktion ist nach dem Schema verbObjekt vorzugehen (Bsp: getUser, checkLogin usw). Besteht kein sinnvolles Objekt, das zur Namensbildung herangezogen werden kann, oder kann eine Überschneidung mit ColdFusion-Core-Funktionen nicht ausgeschlossen werden, ist der Präfix fn zu verwenden (Bsp: fnNewLine, fnReFindAll).
- Erfinde das Rad nicht neu: cflib.org bietet eine Vielzahl kleiner Helferlein, sogenannter UDF.

Komponenten

- In aller Regel sind Komponenten nicht für den direkten Output vorgesehen. Deshalb ist das Attribut "output" explizit auf false zu setzen.

Includes

- Der Pfad zum Include-Template sollte nicht mit einem "/" beginnen:

```
<!---  richtig --->
<cfinclude template="../views/header.cfm" />
<cfinclude template="#application.clientMapping#/views/header.cfm"
<!---  falsch --->
<cfinclude template="/views/header.cfm" />
```

Die Pfadangabe wird vom Application-Server ermittelt und nicht vom Webserver selber. Deshalb werden ColdFusion Mapping-Angaben berücksichtigt, welche sich nicht mit den Mappings im Webserver decken müssen (Bsp: CFIDE).

Listentrenner

- Bei numerischen Werten und UIDs ist als Listentrenner das Komma zu verwenden und muss nicht speziell angegeben werden.
- Bei nicht numerischen Werten oder UIDs hat sich als Listentrenner folgendes Zeichen bewährt: |

Datenbank-Abfragen

- Datenbank-Abfragen sind einerseits so performant wie möglich und andererseits so generell wie nötig zu schreiben. Wenn davon ausgegangen werden muss, dass das System nicht verschiedene Datenbank-Typen berücksichtigen muss und wird, soll eine Datenbank-spezifische Schreibweise verwendet werden. Andernfalls sind ORM-Tools (Reactor, transferORM) oder eine allgemeine Schreibweise zu verwenden.
- Aufwändige Datenmanipulationen erledigen Datenbanken oftmals besser.
- Store Procedures, Views und Triggers können verwendet.
- Transaktionskontrolle ist zur Fehlervermeidung zu verwenden.

Testing

Für ColdFusion-Testing bieten sich folgende kostenlosen Tools an:

- mxUnit und cfcUnit: Unit-Testing
- ColdFire: Debugging-Extension für Firebug
- Der Railo-Application-Server bietet die Möglichkeit, ColdFusion-Code zu kompilieren und dabei allfällige Fehler zu entdecken.
- Zu Überprüfung der Scopes in Komponenten kann die frei verfügbare Applikation varScoper eingesetzt werden.

Kommentare

- Einzeilige Kommentare brauchen in der Regel keine eigene Zeile, sondern können im Anschluss an das ColdFusion-Statement geschrieben werden.
- ColdFusion-Kommentare haben die folgende Syntax:

```
<!---  Hier steht ein CF-Kommentar --->
<cfscript>
// Ein ColdFusion-Kommentar innerhalb eines CFScript-Blocks
</cfscript>
```

- Zahlreiche ColdFusion-Tags bieten Selfdocumentation-Features, beispielsweise hint-Attribute. Diese sollen verwendet werden. Oftmals wird dadurch ein normaler ColdFusion-Kommentar überflüssig und die automatische Dokumentationserstellung wird vereinfacht.
- Es ist eine Beschreibung für jede ColdFusion-Datei einzufügen.
  Beispiel:

```
<!---  ------------------------------------------------------------
FileName        : frmLogin.cfm
Author          : samelim
Purpose         : Login-Formular
Modification Log:
Name            Date            Description
================================================================
samelim         21.08.2009      Created
------------------------------------------------------------
```

- Wenn Kommentare zur weiteren Bearbeitung des Dokuments dienen, sollen Standard-Begriffe verwendet werden (TODO, FIXME usw).

```
<!---  TODO: Session bereinigen | samelim | 2009/06/15 11:39:43 AM
```

Interaktion mit JavaScript

Werden Daten zur Weiterverarbeitung an JavaScript übergeben, sind die Werte mit JSStringFormat() abzusichern.Zur Übermittlung komplexer Daten ist das JSON-Format zu verwenden. Die in JSON

aufbereiteten Daten können mit jsonlint validiert werden.

Tipps

Adobe CF9 ORM

- Einführung in ORM mit Hibernate von Adobe ColdFusion
- ORM-Dokumentation für Adobe CF9
- HQL: Dokumentation der Hibernate Query Language

externe Quellen

Dokumentationen

- Learn Modern ColdFusion (CFML) in 100 Minutes
- Learn CF in a week
- Livedocs für Adobe und Railo
- Livedoc für CF9 mit CFlib integriert

Adobe

- Adobe ColdFusion Learn & Support
- Tour de ColdFusion - Dokumentation mit Code-Beispielen als Air-Applikation
- ColdFusion 9 (en)
- CF9 Development Guide PDF-Dokument, 23 MB (en)
- ColdFusion MX 8 (en)
- ColdFusion MX 7 (en)
- ColdFusion MX 6.1 (en)
- ColdFusion MX 6 (en)
- ColdFusion 5 (en)

Lucee/Railo

- Lucee
- Railo 3.1 (en)

CheatSheets

- ColdFusion 9
- ColdFusion allgemein
- CFScript
- DateFormat
- Regular Expressions
- RegEx 2

Foren

- cfml.de (de)
- Railo Yahoo Group (de)
- Railo Google Group (en)
- Adobe ColdFusion Forum (en)

Code

- Funktions-Bibliothek von cflib.org (en)
- OpenSource-Projekte mit ColdFusion bei riaforge (en)
- ColdFusionCookBook (en)
- Adobe ColdFusion Cookbook (en)
- Tutorials bei LearnCF (en)
- Snippets-Sammlungen (en)
- CF9-Tutorials (en)

Sicherheit

- Adobe Product Security Incident Response Team (en)
- Empfehlungen für CF-Administrator Einstellungen (en)
- Security-Tipps zu CF8 von Adobe (en)
- Einstieg in Sicherheitsaspekte bei ColdFusion

Regular Expressions

- RegEx visualieren online
- RegEx-Tool, Air-Applikation

diverses

- Tools and Resources to Consider for CF developers, von Charlie Arehart (en)
- coldfusionmeetup (en)
- cfmeetup Recordings
- Developer-Wiki von Jim Priest mit Links zu diversen Programmier-Themen (en)
- Einführung in OOP - Objekt orientierte Programmierung (en)