1 Einleitung

In den folgenden Abschnitten finden Sie einige Hinweise zu der clientseitigen, in Javascript realisierten Template-Engine, die Sie im Praktikum einsetzen.

1.1 Überblick

Mit den beiden Scripts `te.js` und `tm.js` steht Ihnen eine einfache clientseitige Template-Engine zur Verfügung. Die Templates werden als Dateien geladen und bei Bedarf zu Javascript-Funktionen kompiliert, ähnlich der Vorgehensweise der Python-Template-Engine `mako`.

Beide Scripts verwenden die neuere `class`-Schreibweise, die bei Javascript die Nutzung objektorientierter Konzepte im Vergleich zu `prototype`-Schreibweise vereinfacht.

Die beiden Script stammen vom Autor dieser Hinweise.

1.2 Verwendung im Webbrowser

Zur Nutzung der Komponenten wird die Javascript-Bibliothek `jquery` **benötigt**. Eine spezielle Versionsabhängigkeit besteht nicht.

`jquery` und den Quellcode der Komponenten müssen Sie im `head`-Bereich mit Hilfe des `<script>`-Tag einbinden, z.B. so:

```
<script type="text/javascript" src="/jquery.js"></script>
<script type="text/javascript" src="/te.js"></script>
<script type="text/javascript" src="/tm.js"></script>
```

Serverseitig müssen die Anfragen durch eine geeignete Konfiguration und Ordnerstruktur auflösbar sein.

1.3 Template-Compiler `te.js`

1.3.1 Compiler

In der Klasse `TemplateCompiler_cl` ist der Template-Compiler implementiert. Der Methode `compile_px` wird der Quellcode eines Templates übergeben. Dieser Quellcode wird durch einen Scanner zeichenweise verarbeitet, die zugrunde liegende Syntax wird überprüft. Bei der Erkennung der einzelnen syntaktischen Einheiten wird fortwährend ein Generator-Objekt gefüllt, dass dann nach der Verarbeitung des Quellcodes zur Erzeugung des Zielcodes ausgeführt wird.

1.3.2 Generator

Der Generator, implementiert in der Klasse `Generator_cl`, ist das Backend des Compilers; er erzeugt anhand der ihm übergebenen Information den Javascript-Quellcode für das ausführbare Template. Schließlich wird mit Hilfe des vordefinierten `Function`-Object eine Javascript-Funktion erzeugt und als Ergebnis zur weiteren Verwendung geliefert.

Die erzeugte Javascript-Funktion verfügt über den **Formalparameter** `context`, dem normalerweise ein Objekt mit den auszuwertenden Daten als Aktualparameter übergeben wird.

1.4 Management `tm.js`

Das Script ermöglicht mit der Klasse `TemplateManager_cl` eine einfache Verwendung der Template-Engine:

- mit der Erzeugung einer Instanz der Klasse wird im constructor per Ajax-Request im Hintergrund an den Server die Anforderung `/templates/` übertragen; erwartet werden *alle* Templates in Form von `JSON`-formatierten Daten:
  - Objekt mit der Eigenschaft 'templates'
  - diese Eigenschaft als Objekt mit Eigenschaften, die den Namen des jeweiligen Template und den Quelltext enthalten
  - Sie müssen dafür beim Server ein Modul zur Bereitstellung dieser Ressourcen anhand von Template-Quelltextdateien vorsehen
- die gelieferten Template-Quelltexte werden im Manager-Objekt gespeichert
- geladene Templates werden mit der Methode `execute_px` ausgeführt:
  - der Methode werden der Template-Name und die Daten übergeben
  - die Methode wird zunächst die Kompilierung des Templates veranlassen, wenn dies noch nie benutzt wurde
  - das erzeugte Funktionsobjekt für das Template wird dann mit den übergebenen Daten ausgeführt; diese Daten werden dem Formalparameter `context` zugewiesen
  - Ergebnis ist der ausgewertete Text, der als String geliefert wird.

1.5 Anwendung

Zur Nutzung der Template-Engine muss zunächst ein Manager-Objekt als Instanz der Klasse `TemplateManager_cl` erzeugt werden. Dieses lädt unmittelbar die Template-Quelltexte (beachten Sie: dennoch asynchron!).

Anschließend können die Templates durch den Aufruf der `execute_px`-Methode des Manager-Objekts ausgewertet werden.

1.5.1 Syntax der Templates

Die Zeichen der Template-Quelltexte werden ohne Änderung in den Zieltext übernommen mit zwei Ausnahmen:

- Markierung von Textabschnitten mit dem Zeichen `@` zu Beginn und zu Ende des Textabschnitts
  - damit werden Javascript-Code-Abschnitte gekennzeichnet, d.h. dieser Text wird ungeändert in den Javascript-Code, der für das Template erzeugt wird, übernommen
  - dabei werden zwei syntaktische Vereinfachungen / Änderungen angeboten bzw. erforderlich
  - `for`-Schleifen können ohne umgebende Klammern geschrieben werden; auch wird der Schleifenkörper nicht in geschweifte Klammern gesetzt
  - `for`-Schleifen müssen mit `endfor` abgeschlossen werden
  - entsprechendes gilt für die Fallunterscheidung mit `if`; hier ist die Verwendung von (ggf.) `else` und `endif` erforderlich
- Markierung von Textabschnitten mit dem Zeichen # zu Beginn und zu Ende des Textabschnitts
  - dies sind Abschnitte, die als Javascript-*Ausdrücke* ausgewertet werden
  - im einfachsten Fall als Bezeichner einer Variable
  - oder als Zugriff auf den Formalparameter `context` mit entsprechenden Indizes
  - oder als komplexer, in Javascript zugelassener Ausdruck; es ist erforderlich, dass alle Daten / Strukturen zur Auswertung des Ausdrucks zu diesem Zeitpunkt vorliegen!

Sollen die Zeichen # und `@` nicht als Markierung verwendet werden, müssen Sie wiederholt werden: ## ergibt ein einzelnes # im ausgewerteten Template-Text, `@@` entsprechend ein einzelnes `@`.

1.5.2 Beispiel

Im folgenden Template wird eine Datenstruktur ausgewertet, um eine Liste von Hyperlinks zu erzeugen, die ggf. durch `hr`-Elemente getrennt werden:

```
<!-- Template -->
<!-- Navigationsbereich -->

@var entry_a;@
@var loop_i;@
@for loop_i = 0; loop_i < context.length; loop_i++@
   @entry_a = context[loop_i];@
   @if entry_a[0] == 'divider'@
   <hr />
   @else@
   <a href="##" data-action="#entry_a[0]#">#entry_a[1]#</a>
   @endif@
@endfor@
<!-- EOF -->
```

Die zu übergebenden Daten könnten so aussehen:

```
[
   ['home', 'Startseite'],
   ['divider', ''],
   ['form1', 'Formular 1'],
   ['form2', 'Formular 2']
]
```

1.5.3 Serverseitige Bereitstellung

Bei einem mit `cherrypy` in Python realisierten Webserver könnte die Bereitstellung der Template-Quelltexte etwa so vorgenommen werden:

```
   # Methode z.B. in application.py
   @cherrypy.expose
   #-------------------------------------------------------
   def templates(self):
   #-------------------------------------------------------
      retVal_o = {
         'templates': {}
      }

      files_a = os.listdir('templates') # relativ zum Startverzeichnis des Server
      for fileName_s in files_a:
         file_o = codecs.open(os.path.join('templates', fileName_s), 'rU', 'utf-8')
         content_s = file_o.read()
         file_o.close()
         retVal_o["templates"][fileName_s] = content_s

      return json.dumps(retVal_o)
```