1 Einleitung

In den folgenden Abschnitten finden Sie Hinweise zu JavaScript-Komponenten, die Sie bei der Bearbeitung von Praktikumsaufgaben verwenden sollen.

Die Scripts sind Eigenentwicklungen, für deren korrekte Funktionsfähigkeit keinerlei Gewährleistung übernommen werden kann.

1.1 Überblick

Es handelt sich um folgende Komponenten:

- Eventservice:
  - clientseitiger asynchroner Datenaustausch zwischen JavaScript-Komponenten
  - Implementierung in `evs.js`
- Template-Engine:
  - Implementierung in `tco.js` und `tmg.js`
- Wrapper für die Nutzung der Fetch-API:
  - Implementierung in `req.js`.

1.2 Verwendung im Webbrowser

Zur Nutzung der Komponenten werden *keine* anderen JavaScript-Bibliotheken benötigt.

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

```
<script type="text/JavaScript" src="/req.js"></script>
<script type="text/JavaScript" src="/evs.js"></script>
<script type="text/JavaScript" src="/tco.js"></script>
<script type="text/JavaScript" src="/tmg.js"></script>
```

Anhand der Konfiguation des Webservers kann festgelegt werden, auf welche physikalische Datei die jeweilige Anfrage abgebildet wird. Entsprechende Angaben können Sie der Konfigurationsdatei des einfachen Demonstrationsbeispiels entnehmen.

Die genannten Komponenten verwenden durchweg die ES6-Klassenschreibweise. Ältere Webbrowser, die diese Schreibweise nicht unterstützen, können nicht verwendet werden.

2 Allgemeine Vorgaben

2.1 Strict-Mode

Die Scripts verwenden den sog. **Strict-Mode**. Im Strict-Mode werden einige problematische Konstrukte nicht zugelassen. Insbesondere wird nicht zugelassen, Bezeichner ohne vorherige Deklaration mit `var` oder `let` zu erzeugen.

2.2 Namensraum `APPUTIL`

Um die Belegung des globalen Namensraums mit Bezeichnern, z.B. für Funktionen oder Klassen, möglichst gering zu halten und Namenskonflikte bei Bezeichnern zu vermeiden, wird der Namensraum `APPUTIL` definiert. Dieser Namensraum ist nichts anderes als ein einfaches Objekt, in den die speziellen Bezeichner eingetragen werden.

Bei jedem Script wird zu Beginn geprüft, ob der Namensraum bereits definiert ist. Ist dies nicht der Fall, wird der Namensraum definiert. Die Vorgehensweise stellt sicher, dass die Reihenfolge der Verwendung der Scripts hinsichtlich der Verfügbarkeit des Namensraums keine Rolle spielt.

3 Eventservice (evs.js)

3.1 Publish-Subscriber-Muster

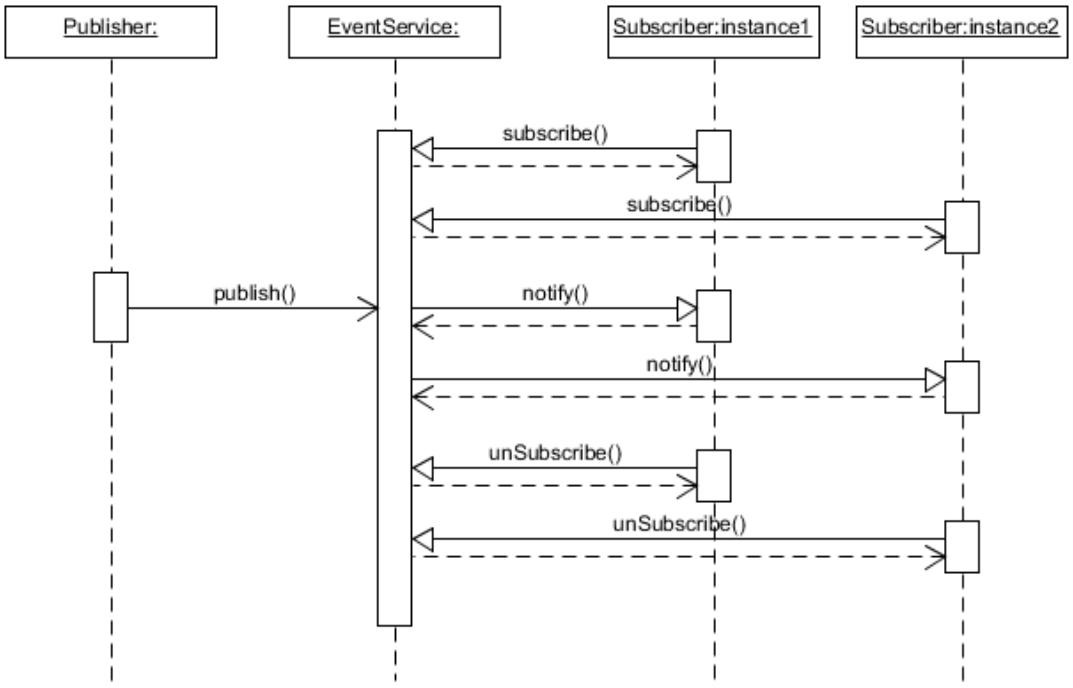
Die Komponente Eventservice implementiert das Publish-Subscriber-Muster:

- `Subscriber` melden sich zum Empfang von Nachrichten an (`subscribe`-Schnittstelle)
- ein `Publisher` veröffentlicht über die `publish`-Schnittstelle des Eventservice eine Nachricht
- die `Subscriber` erhalten dann eine Benachrichtigung über ihre `notify`-Schnittstelle
- `Subscriber` beenden den Empfang durch Aufruf der `unSubscribe`-Schnittstelle.

Mit Hilfe dieses Musters ist es möglich, dass Nachrichten von einem Objekt ohne Kenntnis der tatsächlichen Empfänger versendet werden können.

Im folgenden Sequenzdiagramm sind diese Zusammenhänge dargestellt:

- die Verarbeitung der Veröffentlichung (`publish`-Schnittstelle) erfolgt asynchron, daher ist hier auch eine asynchrone Botschaft zwischen `Publisher` und Eventservice eingetragen
- beispielhaft melden sich 2 `Subscriber` zum Nachrichtenempfang an
- eine Nachricht wird durch den Aufruf der `notify`-Schnittstellen an die `Subscriber` weitergeleitet
- die `Subscriber` beenden den Nachrichtenempfang.



Sequenzdiagramm Publish-Subscriber

Zur Notation:

- durchgezogene Linie mit offener Pfeilspitze: asynchroner Aufruf einer Methode = Nachricht asynchron senden
  - es wird dann auch keine Antwort erwartet

- durchgezogene Linie mit geschlossener Pfeilspitze: synchroner Aufruf einer Methode = Nachricht synchron senden
  - die Antwort wird als gestrichelte Linie mit offener Pfeilspitze dargestellt.

3.2 Implementierung

Der `EventService` wird in der Klasse `EventService_cl` implementiert. Diese enthält die Methoden

- `constructor`
- `subscribe_px`
- `unSubcribe_px`
- `publish_px`.

Beim Registrieren (`subscribe_px`) und Aufheben der Registrierung (`unSubscribe_px`) muss ein `Subscriber` den Verweis auf sich selber sowie die Bezeichnung der Nachricht übergeben. Er erhält dann beim Auftreten einer passenden Nachricht über seine Schnittstelle `notify_px` einen Verweis auf sich, die Bezeichnung der Nachricht sowie Daten zur Nachricht.

Die internen Methoden `defer_p`, `each_p`, `findAll_p` und `compact_p` dienen der Vereinfachung der Implementierung.

In `defer_p` wird die JavaScript-Funktion `setTimeout` verwendet, mit der JavaScript-Funktionen zeitabhängig ausgeführt werden können. Da der JavaScript-Interpreter nur einen (Main-) Thread ausführt, werden zeitabhängige Funktionen *nach* der Beendigung einer Ausführungssequenz ausgeführt. Hier wird durch die Zeitangabe `1 ms` erreicht, dass die Funktion, die als Aktualparameter der JavaScript-Funktion `setTimeout` notiert wir, direkt nach Abschluss der Bearbeitung im Main-Thread ausgeführt wird.

3.3 Beispiel

```
1   var APP = {};
2   APP.es_o = new APPUTIL.EventService_cl();
3
4   class myClass_cl {
5      constructor () {
6         APP.es_o.subscribe_px(this, 'app');
7      },
8      notify_px: function (self_opl, message_spl, data_apl) {
9         switch (message_spl) {
10        case 'app':
11           switch (data_apl[0]) {
12           case 'work.done':
13              console.log(data_apl[1]); // TODO: müsste man vorher auf Existenz prüfen!
14              break;
15           default:
16              console.warning('[Application_cl] unbekannte app-Notification:
'+data_apl[0]);
17              break;
18           }
19           break;
20        default:
21           console.warning('[Application_cl] unbekannte Notification: '+message_spl);
22           break;
23        }
24     }
25  });
26
27  APP.es_o.publish_px('app', ['work.done', 'success']);
```

4 Clientseitige Template-Engine (tco.js, tmg.js)

4.1 Überblick

Mit den beiden Scripts `tco.js` und `tmg.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`.

4.2 Template-Compiler `tco.js`

4.2.1 Compiler

In der Klasse `APPUTIL.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, das dann nach der Verarbeitung des Quellcodes zur Erzeugung des Zielcodes ausgeführt wird.

4.2.2 Generator

Der Generator, implementiert in der Klasse `APPUTIL.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.

4.3 Management `tmg.js`

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

- mit der Erzeugung einer Instanz der Klasse wird per 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
- dann wird über den Eventservice eine Nachricht `templates.loaded` verschickt; i.d.R. wird ein Applikationsobjekt daraufhin die weitere Initialisierung der Anwendung veranlassen
- im Fehlerfall wird über den Eventservice eine Nachricht `templates.failed` versendet
- 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.

4.4 Anwendung

Zur Nutzung der Template-Engine muss zunächst ein Manager-Objekt als Instanz der Klasse APPUTIL.TemplateManager_cl erzeugt werden. Dieses lädt unmittelbar die Template-Quelltexte und sendet dann eine Benachrichtung (siehe oben).

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

4.4.1 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 angeboten und sind nachfolgende Änderungen gegenüber 'normalem' JavaScript notwendig:
    - `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!

Verwendung von `include`

In den mit dem Zeichen `@` gekennzeichneten Textabschnitten kann auch das Ergebnis der Auswertung eines anderen Templates eingefügt werden. Dazu wird folgende Syntax verwendet:

```
@include template-name data-object@
```

`template-name` muss durch den Namen des Templates ersetzt werden, dessen Anwendung hier vorgesehen ist. Beim Platzhalter `data-object` gibt man die Context-Variable an, die bei der Ausführung des Template verwendet wird. Wenn man verschiedene Werte übergeben will, muss die Context-Variable ein Array oder Object sein.

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

Das müssen Sie vor allem bei Links beachten: statt `href=#` müssen Sie dann `href=##` schreiben!

4.4.2 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']
]
```

4.4.3 Serverseitige Bereitstellung

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

```
1        # Methode z.B. in application.py
2        @cherrypy.expose
3        #-------------------------------------------------------
4        def templates(self):
5        #-------------------------------------------------------
6           retVal_o = {
7              'templates': {}
8           }
9
10          files_a = os.listdir('templates') # relativ zum Startverzeichnis des Server
11          for fileName_s in files_a:
12             file_o = codecs.open(os.path.join('templates', fileName_s), 'rU', 'utf-8')
13             content_s = file_o.read()
14             file_o.close()
15             retVal_o["templates"][fileName_s] = content_s
16
17          return json.dumps(retVal_o)
```

5 Demonstrator

Als sehr einfaches Anwendungsbeispiel zur Nutzung der Scripts `evs.js`, `tco.js`, `tmg.js` und `req.js` dient der “Demonstrator”, der Ihnen im Rahmen der Unterlagen zur Veranstaltung WEB mit Quellcode zur Verfügung gestellt wird.

Wesentliche Eigenschaften sind:

- Single-Page-Applikation
- Layout wird mit absoluter Positionierung erzielt
- die Inhalte aller Bereiche wird anhand der fest vorgegebenen Beispieldaten und der Templates erzeugt
- Reaktionen auf die Bedienungen im Sidebar und im Content-Bereich (“anzeigen” bzw. “zurück”) sind interne Nachrichten per Eventservice, die im Applikationsobjekt ausgewertet werden
- serverseitig wird *Method-Dispatching* verwendet, d.h. es wird eine an die REST-Prinzipien angelehnte Kommunikation eingesetzt (allerdings ohne Veränderung von Daten)
- die Konfiguation des Webservers befindet sich in einer Konfigurationsdatei; dort ist auch erkennbar, wie die Kurzbezeichnungen für die Scripts auf die tatsächlichen Dateinamen abgebildet werden.