EBERHARD KARLS
UNIVERSITÄT
TÜBINGEN

Mathematisch-Naturwissenschaftliche Fakultät
Fachbereich Informatik
Informationsdienste – Prof. Dr. Thomas Walter

Einführung Internet-Technologien

Sommersemester 2020

Übungsblatt 3

Theorie

Die Lösungen zu den Theorieaufgaben werden in den Tutorien besprochen und im Moodle veröffentlicht. Sie brauchen *keine* Lösungen zu den Theorieaufgaben abgeben!

1. Erklären Sie die Funktionsweise clientseitiger Programmierung.

Lösung: Der Ablauf clientseitiger Programmierung ist in Abbildung 1 dargestellt. Der Client stellt zunächst eine Anfrage (im Web-Kontext eine HTTP-Anfrage) an den Server. Dieser verarbeitet die Anfrage und erstellt eine Antwort (im Web-Kontext eine HTTP-Response inklusive einer HTML-Beschreibung). In der Antwort ist Quellcode verankert, der vom Client ausgeführt werden soll (im Web-Kontext beispielsweise JavaScript-Quellcode, welcher über das `script`-Tag kenntlich gemacht wird). Der Client empfängt die Antwort, identifiziert die enthaltenen Quellcode-Teile und führt diese lokal bei Bedarf aus; beispielsweise sofort oder erst bei Nutzereingaben.

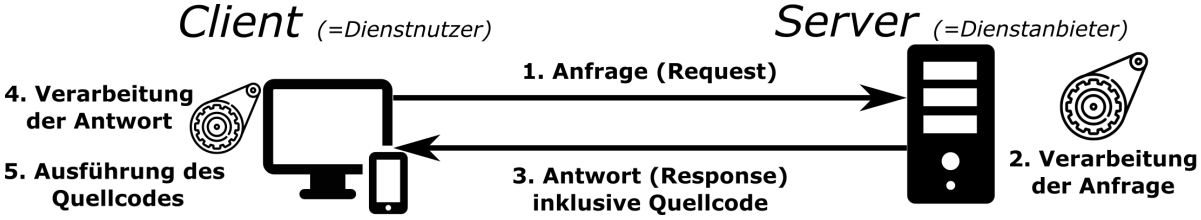

Abbildung 1: Ablauf clientseitiger Programmierung

2. Diskutieren Sie Vor- und Nachteile dieses Ansatzes.

Lösung: Durch die Ausführung des Quellcodes auf den Nutzer-Rechner ergibt sich für den Dienstanbieter der Vorteil, dass seine Server entlastet werden. Dadurch können die Server in derselben Zeit mehr Anfragen beantworten als wenn sie gleichzeitig noch die Programmausführung durchführen müssten. Aufgrund der Ausführung im lokalen Betriebssystem kann das clientseitige Programm auf dessen Schnittstellen zugreifen und erhält somit Zugriff auf individuelle Parameter; beispielsweise die Mauszeigerposition oder die exakte Bildschirmauflösung. Auf diese Weise kann das Programm direkt auf Nutzerverhalten reagieren. Ebenso können fehlerhafte Eingaben direkt auf dem Client erkannt und abgefangen werden. Dies steigert einerseits die Nutzerfreundlichkeit, da keine Wartezeit beim absenden eines fehlerhaften Requests entsteht und andererseits wird das Netz entlastet, was insbesondere bei langsamen oder Bandbreite-beschränkten Verbindungsarten relevant

ist; beispielsweise einem mobilem Datennetz. Ein weiterer Vorteil der clientseitigen Programmierung ist, dass der Client die Programmausführung direkt beeinflussen kann. Beispielsweise bei freiwerdenden Ressourcen mehr Rechenleistung auf die Programmausführung aufbringen oder die Ausführung auf Nutzerwunsch beenden.

Nachteilig für den Nutzer ist, dass seine CPU während der Ausführung belastet wird. Weiterhin ist eine Anbindung an zentrale Datenbanksysteme theoretisch zwar möglich, jedoch nicht praktibal Umsetzbar. Dies liegt daran, dass das Programm für den Zugriff auf die Datenbank ein Login und ein Passwort benötigt die an alle zugreifenden Clients mit ausgeliefert werden müssten. Somit hätte jeder Zugriff auf die Datenbank und könnte nach Belieben Daten einpflegen oder auslesen, was in professionellen Projekten nicht realistisch umsetzbar ist. Ein weiterer Nachteil für den Nutzer ist, dass er von teilweise unvertrauenswürdigen Quellen Programmcode auf seinen Rechner lädt und auf diesem ausführt. Dieser kann potentiell gefährlich sein, können auf diese Weise unbemerkt Viren, Trojaner und sonstige Schadsoftware auf den Nutzer-Rechner geladen und installiert werden. Das Problem verstärkt sich dadurch, dass über nachgeladenen clientseitigen Programmcode von Dritten Quellen der Dienstanbieter nicht unbedingt weiß, welchen Quellcode er an seine Dienstnutzer ausliefert. Der Angriff wird als Cross-Site-Scripting (XSS) bezeichnet.

Für den Serverbetreiber ergibt sich das Problem, dass der Quellcode an den Client ausgeliefert werden muss. Somit können die Zugreifenden problemlos Kopien des Quellcodes erstellen und selbst Plagiate veröffentlichen. Weiterhin hat der Server keinerlei Kenntnis über die Programmausführung beziehungsweise den Zustand der Programmabarbeitung auf dem Client. Dementsprechend kann der Server nicht auf (Zwischen-)ergebnisse reagieren, ohne das der Client diese dem Server über eine neue Anfrage mitteilt. Das wohl schwierigste Problem für den Serverbetreiber ist, dass nicht sichergestellt ist, dass die benötigte Programmausführungs-Umgebung auf dem Client existiert. So kann beispielsweise JavaScript vom Nutzer im Browser deaktiviert sein oder Adobe Flash nicht installiert sein, so dass die Ausführung solcher Programme zu keinem Zeitpunkt erfolgen kann. Insbesondere kritisch ist dies, wenn die Entwicklungsprogramme Versionsbasiert sind und eine Abwärtskompatibilität nicht gewährleistet ist: In diesem Fall kann zwar die Ausführungsumgebung vorhanden sein jedoch bei der Ausführung durch fehlerhafte Versionen ein Fehler auftreten.

3. Nennen Sie typische clientseitige Anwendungen.

Lösung: clientseitige Anwendungen sind alle Anwendungen, die auf dem lokalen Nutzer-Rechner ausgeführt werden und mit Anwendungen auf anderen Netzwerk-Rechnern kommunizieren. Im Webumfeld ist typischer Weise der Webbrowser als Client-Anwendung bekannt. Ebenso sind jedoch andere Anwendungen die über das Web agieren clientseitige Anwendungen; beispielsweise Mail-Clients (z.B. Outlook,

Thunderbird, ...), Spiele-Clients (z.B. Starcraft2, League of Legends, ...) oder verteilte Speicher (z.B. ownCloud, Dropbox, ...).

4. Wie können Sie Nutzer darauf hinweisen ihr deaktiviertes JavaScript zu aktivieren?

Lösung: Über das so genannte `noscript`-Tag können Nutzern bei ausgeschaltetem JavaScript Informationen angezeigt werden. Da dies allein über HTML angegeben werden kann, können Browser prüfen, ob JavaScript aktiviert ist und in diesem Fall alle Kind-Elemente innerhalb von `noscript`-Tags rendern. Andernfalls überspringt der Browser diese Elemente einfach. Somit können Dienstanbieter darauf hinweisen JavaScript zu aktivieren.

5. Weshalb ist JavaScript so erfolgreich?

Lösung: Der Erfolg von JavaScript lässt sich auf die direkte Browserintegration zurückführen. Dadurch ist es für Dienstanbieter einerseits sehr einfach möglich in Ihren Weboberflächen Dynamik einzubauen, welche direkt auf Nutzerverhalten reagiert. Somit wird die Nutzerfreundlichkeit deutlich erhöht und es können Anwendungen erstellt werden, die ansonsten nicht ohne weitere Ausführungsumgebungen möglich wären. Dies ist auch der zweite Punkt für den Erfolg von JavaScript: Dienstanbieter können nahezu sicher sein, dass ihre Programme auf den Nutzergeräten ausgeführt werden können, da die relevanten Browser JavaScript (weitestgehend Abwärtskompatibel) ausführen können.

6. Wie können Sie JavaScript-Quellcode auf Ihrer Webseite einbetten? Wer führt den Code aus?

Lösung: JavaScript wird über das `<script>`-Tag in HTML integriert. Dabei ist es möglich, JavaScript-Code direkt innerhalb des `<script>`-Tags zu schreiben oder über eine externe Datei einzubinden (`<script src="myScript.js"></script>`).

Das `<script>`-Tag darf sich sowohl innerhalb des HTML-Head als auch an beliebiger Stelle im HTML-Body befinden. Besser ist es, Skripte am Ende des Bodys zu plazieren, weil dies die Ladegeschwindigkeit der Seite erhöht.

JavaScript-Quellcode wird von einem JavaScript-Interpreter ausgeführt, der im Browser integriert ist. Dementsprechend erfolgt die Ausführung als Teil des Browserprozesses.

7. JavaScript ist eine nicht-typisierte Skriptsprache. Was bedeutet das? Welche Datentypen bietet JavaScript?

Lösung: JavaScript ist nichttypisiert. Das bedeutet, der Entwickler muss sich während der Programmierung keine Gedanken über die Datentypen seiner Variablen machen muss. Erst anhand der Werte die eine Variable zur Laufzeit annimmt entscheidet die Ausführungsumgebung welcher Datentyp benötigt wird und weißt diesen der Variablen zu. Somit können Variablen während der Laufzeit durchaus unterschiedliche Datentypen annehmen.

JavaScript bietet folgende primitive Datentypen:

- String für Zeichenketten
- Number für Zahlen
- Boolean für Logikwerte (0/1)
- Null für nichts
- Undefined für Variablen ohne Wert

Zustäzlich bietet JavaScript die komplexen Datentypen `Function` und `Object` sowie (assoziative) Arrays.

Der Typ einer Variablen lässt sich mit dem `typeof`-Operator prüfen. Dieser kann verwendet werden um eine String-Beschreibung des Datentyps einer Variablen zu erhalten.

8. Was ist in `result` gespeichert? `var result = Array(16).join("wat"-1) + " Batman";`

Lösung: Das Ergebnis dieser Berechnung ist: NaNNaNNaNNaNNaNNaNNaNNaNNaNNaNNaNNaNNaNNaNNaNNaN Batman. Das Interessante daran ist, dass dieses Ergebnis keinerlei Sinn macht. Eigentlich erwartet man einen Fehler bei der Berechnung "wat"-1, da die Datentypen String und Integer mit dem Minusoperator nicht vereinbar sind. JavaScript führt die Berechnung dennoch durch und erhält als Ergebnis Not a Number (NaN), welche es als Ergebnis in die join-Methode einfügt. Diese konkatiniert diesen Wert 16mal miteinander und durch die Konkatenation mit Batman ergibt sich obige Ausgabe.

9. Was ist das DOM und wofür wird dies verwendet?

Lösung: Das Document Object Model (DOM) ist eine Schnittstellenbeschreibung (Interface) die Definiert, wie Browser Informationen über das Endnutzersystem vorhalten und wie Skriptsprachen auf diese Zugreifen sowie diese Verändern können. Insbesondere wird nach dem Schema der Aufbau gerenderter Webseiten sowie (in Echtzeit) dynamische Anpassungen auf diesen realisiert. Theoretisch unterschieden, praktisch jedoch oft in Kombination mit dem DOM verknüpft existiert zusätzlich das Browser Object Model (BOM). Dieses wird verwendet um den Skriptsprachen Zugriff auf technische Informationen des Browsers zu ermöglichen; beispielsweise Browserinformationen wie Hersteller und Version oder technische Eigenschaften des zugreifenden Rechners, wie beispielsweise Bildschirmauflösung und Pixeldichte. BOM und DOM sind dabei so verbunden, dass das DOM ein Teil des BOM ist: Das BOM-Element `window` ist das Root-Element des Konzepts (siehe Aufgabe 2). Es steht für das aktive Browser-Fenster (nicht den aktiven Tab!) und enthält als Kindelemente die restlichen BOM-Elemente sowie das Root-Element `document` des DOM. Wichtig ist zu wissen, dass es für das BOM keine W3C-Empfehlung gibt, Browser es also nach Belieben umsetzen können, während das DOM in 3 Standards (Level 1, Level 2 und Level 3) standardisiert wurde.

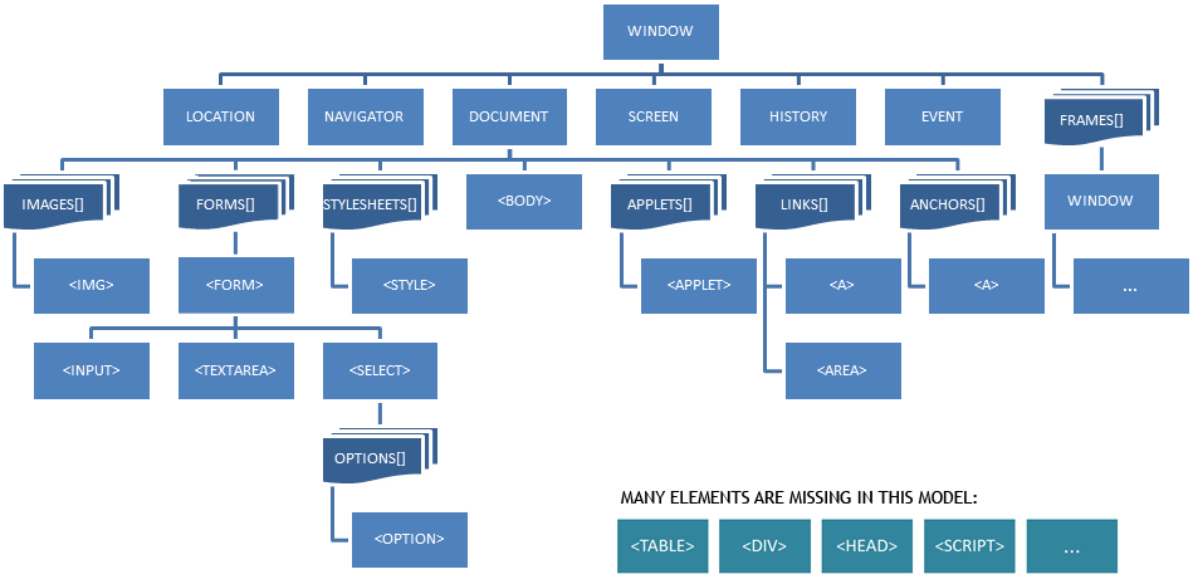

Abbildung 2: Struktur von BOM und DOM

10. Skizzieren Sie das DOM schematisch.

Lösung: Siehe Abbildung 2.

11. Welche Knoten beinhalten Informationen über den Bildschirm des Zugreifenden Geräts, über den Browser und über den aktiven Tab? Geben Sie Beispieleinträge an.

Lösung: Alle gefragten Informationen sind Kindelemente des BOM `window`-Knotens. Sie werden jedoch weitestgehend in eigenen Knoten vorgehalten, die Kindknoten des `screen`-Knotens sind. Der `window`-Knoten beinhaltet die Informationen über das aktive Browserfenster; beispielsweise die Höhe und Breite (window.innerWidth) des Browserfensters (window.innerHeight). Informationen über den Bildschirm des zugreifenden Geräts werden im `screen`-Element vorgehalten; beispielsweise Bildschirmhöhe (window.screen.height) oder Pixeldichte (pixelDepth). Alle Einträge zum verwendeten Browser sind im `navigator`-Knoten gespeichert. Der Name ist hierbei eine Homage an den ersten kommerziell erfolgreichen Browser: Den Netscape Navigator. Beispiele für navigator-Elemente sind der Browserbezeichner (window.navigator.appCodeName) sowie das Ziel-Betriebssystem (window.navigator.platform). Die Informationen zum aktiven Tab sind im `document`-Knoten hinterlegt. Dieser enthält auch die Struktur der gerendert angezeigten HTML-Beschreibung (beginnend beim body) und kann via einer Skriptsprache in Echtzeit dynamisch ausgelesen und verändert werden. Beispiele für Elemente im document sind der body der HTML-Beschreibung (window.document.links), aber auch zusätzliche Informationen auf der Seite eingebettete Bilder (window.document.body) und Links (window.document.links). Aufgrund dessen, dass der `window`-Knoten ohnehin immer der Wurzelknoten ist, kann dieser bei Anfragen in Skriptsprachen

oft weg gelassen werden; z.B. ist ein Zugriff auf document direkt möglich: document.getElementById(„test“);.

12. Erklären Sie, wie Sie in JavaScript über das DOM auf HTML-Elemente zugreifen können.

Lösung: JavaScript kann direkt über das DOM auf HTML-Elemente zugreifen. Dabei kann entweder über den direkten DOM-Pfad beginnend beim document-Knoten auf das Element zugegriffen werden. Dabei werden Kind-Elemente von ihren Eltern-Elementen immer über einen Punkt getrennt und Geschwister-Elemente über die Methode .nextSibling angegeben. Das Problem an der Herangehensweise mit festen Pfaden ist, dass Entwickler den Seitenaufbau exakt kennen müssen, damit die Zugriffe korrekt erfolgen können. Da dies - insbesondere in dynamischen Webseiten - nicht gewährleistet werden kann, existieren zusätzlich die (selbsterklärenden) JavaScript-Methoden .getElementById(„id“), .getElementsByClassName(„classname“), .getElementsByTagName(„tagname“), querySelector(„Query-Abfrage“), sowie .getElementsByName(„name“). Diese liefern entweder direkt das gesuchte Objekt oder Listen in denen gesuchte HTML-Elemente enthalten sind. Sollte das gesuchte Element ein Kindelement der gefundenen Elemente sein, so kann wieder über die Pfadschreibweise von dem gefundenen Element aus fortgefahren werden.

13. Wie können Sie mit JavaScript den Wert eines HTML-Elements auslesen und ändern?

Lösung: Um den Inhalt eines HTML-Elements vom Typ input auszulesen kann das DOM-Attribut .value verwendet werden. Dieses liefert den Inhalt an das aufrufende Skript zurück. Dabei ist der zurückgelieferte Wert abhängig vom Eingabefeld, typischer Weise jedoch eine String-Repräsentation der Eingabe / Auswahl. Die Inhalte von anderen HTML-Elementen können über das DOM-Attribut .innerHTML ausgelesen werden. Dabei muss bedacht werden, der Rückgabewert eine String-Repräsentation ist, die auch HTML-Markup enthalten kann.

Die Werte von HTML-Elementen können ebenfalls über das Attribut .value (für Elemente vom Typ input) sowie .innerHTML (für andere Elemente) gesetzt werden. Einziger Unterschied zum Auslesen ist, dass beim Verändern der Inhalte eine Zuweisung erfolgen muss.

14. Was ist eine Callback-Funktion?

Lösung: Eine Callback-Funktion ist eine Funktion, die einer anderen Funktion als Argument übergeben wird und von dieser erst später ausgeführt wird.

Callback-Funktionen werden in JavaScript sehr häufig eingesetzt. Beispielsweise erfolgt das Event-Handling über Callback-Funktionen (siehe nächste Frage).

15. Wozu dienen JavaScript-Events?

Lösung: JavaScript-Events dienen dazu, Nutzeraktivität im Browser zu erkennen und über JavaScript auf diese zu reagieren. Beispielsweise, wenn ein Benutzer auf einen Button geklickt hat.

Um auf Events zu reagieren, fügt man den entsprechenden HTML-Elementen Listener hinzu. Hierfür gibt es zwei Möglichkeiten:

Über das HTML-Attribut `on`*`event`* kann man einen Listener direkt einem HTML-Element hinzufügen (*`event`* ist ein Platzhalter für ein konkretes Event). z.B.

```
<button id="b" onclick="listener()">Klick mich!</button>
```

Hier wird die Funktion `listener()` sobald der Button angeklickt wurde.

Besser ist es jedoch, Listener über JavaScript zu den HTML-Elementen hinzuzufügen und so ein Vermischen von HTML- und JavaScript-Syntax zu vermeiden. Dies geht über die DOM-Methode `addEventListener`. Hierfür ist es notwendig, per DOM Zugriff auf das konkrete HTML-Element zu erhalten um dann den Listener hinzuzufügen z.B.

```
document.getElementById("b").addEventListener("click", listener);
```

Das erste Argument von `addEventListener` gibt das Event an, auf das der Listener reagieren soll (ohne „on“!), das zweite Argument ist der Listener. Wichtig: die fehlenden Klammern bei `listener` sind beabsichtigt, da an dieser Stelle die Funktion als Callback-Funktion an `addEventListener` übergeben wird. (`listener()` würde die Funktion ausführen)

16. Erklären Sie, wie Sie Nutzereingaben über JavaScript auslesen können.

Lösung: Um Nutzereingaben zu ermöglichen muss die Webseite zunächst HTML-Felder vom Typ input beinhalten. Diese werden wiederum unterteilt in unterschiedliche Elemente: text, Textarea, Selections, Radio Boxen und Checkboxen. JavaScript kann das DOM verwenden um den Pfad zum gewünschten Eingabefeld entlang des Dokumentbaums zu identifizieren. Hierbei sei auf die Möglichkeit von getElementById sowie getElementByTagName verwiesen. Sobald man das Feld identifiziert hat, kann dessen Wert über die Methode .value ausgelesen werden. Dabei ist zu beachten, dass Textfelder, Textareas sowie Selections einen String zurückliefern, Radio Boxen und Checkboxen jedoch den wert checked.

Abgabe bis Montag, 18.05.2020, 14:00 Uhr. Geben Sie unter Checkpoint 03 im Moodle *anklickbare* Links auf die von Ihnen erstellten HTML-Seiten *min-max.html*, *broetchenrechner.html*, *farbenlehre.html* und *tabellenmanipulation.html* auf dem Übungsserver ab.