2 Gültigkeitsbereich von Variablen

Gültigkeitsbereiche sind für Programmierer so etwas wie Sauerstoff: Sie sind allgegenwärtig, und meistens denken Sie gar nicht darüber nach. Aber wenn Gültigkeitsbereiche überschritten und verunreinigt werden, dann wird es brenzlig.

Die grundlegenden Regeln für Gültigkeitsbereiche in JavaScript sind zum Glück sehr einfach, gut gestaltet und äußerst leistungsstark. Es gibt jedoch Ausnahmen. Um wirkungsvoll mit JavaScript arbeiten zu können, müssen Sie sowohl die Grundprinzipien für die Gültigkeitsbereiche von Variablen als auch die Sonderfälle beherrschen, die zu kleinen, aber gemeinen Problemen führen können.

Verwenden Sie das globale Objekt so wenig wie möglich

Thema 8

Globale Variablen vermeiden

In JavaScript ist es einfach, Variablen im globalen Namespace zu erstellen. Es macht weniger Mühe, globale Variablen anzulegen, da sie keine Form von Deklaration erfordern und automatisch im gesamten Programm zugänglich sind. Dank dieser Annehmlichkeiten sind sie eine große Versuchung für Anfänger. Erfahrene Programmierer aber wissen, dass sie globale Variablen vermeiden sollten, denn deren Definition verunreinigt den gemeinsamen Namespace und eröffnet die Möglichkeit von versehentlichen Namenskonflikten. Außerdem laufen globale Variablen dem Sinn der Modularität zuwider, denn sie verursachen eine unnötige Kopplung zwischen getrennten Bestandteilen eines Programms. Auch wenn es bequem sein mag, »jetzt zu programmieren und sich später um die Gliederung zu kümmern«, achten gute Entwickler ständig auf die Struktur, indem sie im Rahmen des Programmiervorgangs fortlaufend verwandte Wirkungsweisen gruppieren und nicht zusammenhängende Komponenten trennen.

Variablen so lokal wie möglich halten

Da der globale Namespace die einzige echte Möglichkeit ist, mit der getrennte Komponenten eines JavaScript-Programms miteinander interagieren können, ist eine gewisse Nutzung dieses Namespace unvermeidlich. Eine Komponente oder Bibliothek muss einen globalen Namen definieren, damit andere Teile des Programms darauf zugreifen können. Abgesehen davon aber sollten Variablen so lokal wie möglich gehalten werden. Es ist sicherlich *möglich*, ein Programm ausschließlich mit globalen Variablen zu schreiben, aber damit beschwören Sie nur Probleme herauf. Wenn Sie die temporären Variablen von Funktionen grundsätzlich global definieren, müssten Sie selbst bei ganz einfachen Funktionen befürchten, dass dieselben Variablennamen auch von anderem Code verwendet werden:

```
var i, n, sum; // Globale Variablen
function averageScore(players) {
   sum = 0;
   for (i = 0, n = players.length; i < n; i++) {
      sum += score(players[i]);
   }
   return sum / n;
}
```

Diese Definition von `averageScore` funktioniert nicht, wenn die Funktion `score`, auf die sie sich stützt, irgendwelche dieser globalen Variablen für ihre eigenen Zwecke verwendet:

```
var i, n, sum; // Dieselben globalen Variablen wie bei averageScore!
function score(player) {
   sum = 0;
   for (i = 0, n = player.levels.length; i < n; i++) {
      sum += player.levels[i].score;
   }
   return sum;
}
```

Die Lösung besteht darin, solche Variablen lokal zu halten und auf den Teil des Codes zu beschränken, in dem sie gebraucht werden:

```
function averageScore(players) {
   var i, n, sum;
   sum = 0;
   for (i = 0, n = players.length; i < n; i++) {
      sum += score(players[i]);
   }
   return sum / n;
}

function score(player) {
   var i, n, sum;
   sum = 0;
```

```
    for (i = 0, n = player.levels.length; i < n; i++) {
       sum += player.levels[i].score;
    }
    return sum;
}
```

Das globale Objekt so wenig wie möglich verändern

Der globale Namespace von JavaScript wird auch als *globales Objekt* bereitgestellt, das zu Anfang eines Programms als ursprünglicher Wert des Schlüsselworts `this` zugänglich ist. In Webbrowsern ist das globale Objekt auch an die globale Variable `window` gebunden. Wenn Sie globale Variablen hinzufügen oder bearbeiten, wird das globale Objekt automatisch aktualisiert:

```
this.foo; // Nicht definiert
foo = "global foo";
this.foo; // "global foo"
```

Die Änderung des globalen Objekts wiederum aktualisiert automatisch den globalen Namespace:

```
var foo = "global foo";
this.foo = "changed";
foo; // "changed"
```

Das bedeutet, dass Sie zwei Mechanismen zur Auswahl haben, um globale Variablen zu erstellen: Sie können sie mit `var` im globalen Gültigkeitsbereich erstellen oder dem globalen Objekt hinzufügen. Beides funktioniert, aber die Deklaration mit `var` hat den Vorteil, dass sie die Auswirkungen auf den Gültigkeitsbereich des Programms deutlicher macht. Da eine Referenz auf eine nicht gebundene Variable zu einem Laufzeitfehler führt, ist es für die Benutzer Ihres Codes einfacher, wenn Sie den Gültigkeitsbereich deutlich und einfach festlegen, da sie dadurch besser erkennen können, welche globalen Variablen deklariert werden.

Merkmale erkennen

Es ist zwar am besten, die Verwendung des globalen Objekts so weit wie möglich einzuschränken, doch für einen Zweck ist es unverzichtbar. Da das Objekt eine sogenannte *dynamische Reflektion* der globalen Umgebung bietet, können Sie damit eine laufende Umgebung abfragen, um zu ermitteln, welche Merkmale auf der Plattform zur Verfügung stehen. Beispielsweise wurde in ES5 das neue globale Objekt `JSON` zum Lesen und Schreiben des JSON-Datenformats eingeführt. Als Notlösung für die Bereitstellung von Code in Umgebungen, bei denen nicht sicher ist, ob sie bereits das Objekt `JSON` verfügbar machen, können Sie prüfen, ob dieses Objekt vorhanden ist, und eine alternative Implementierung anbieten:

```
if (!this.JSON) {
   this.JSON = {
      parse: ...,
      stringify: ...
   };
}
```

Wenn Sie eine eigene Implementierung von `JSON` bereitstellen, können Sie diese natürlich auch einfach bedingungslos verwenden. Allerdings sind die Implementierungen der Hostumgebung fast immer zu bevorzugen, da sie sorgfältig auf Korrektheit und Standardkonformität geprüft wurden und häufig eine bessere Leistung bieten als Lösungen von Drittanbietern.

Diese Technik der Merkmalserkennung ist vor allem in Webbrowsern wichtig, da der Code von einer breiten Palette verschiedener Browser und Browserversionen ausgeführt werden kann. Dies ist eine relativ einfache Möglichkeit, um Programme unempfindlich gegenüber den Schwankungen im Funktionsumfang der Plattformen zu machen. Diese Technik lässt sich auch auf anderen Gebieten anwenden, beispielsweise für die Veröffentlichung von Bibliotheken, die sowohl in Browser- als auch in JavaScript-Serverumgebungen funktionieren sollen.

Was Sie sich merken sollten

- Vermeiden Sie die Deklaration globaler Variablen.
- Deklarieren Sie Variablen so lokal wie möglich.
- Vermeiden Sie es, Eigenschaften zum globalen Objekt hinzuzufügen.
- Verwenden Sie das globale Objekt, um den Funktionsumfang einer Plattform zu bestimmen.

Thema 9 Vergessen Sie nicht, lokale Variablen zu deklarieren

Wenn es etwas gibt, das störender ist als eine globale Variable, dann ist es eine *ungewollte* globale Variable. Leider kann es aufgrund der Regeln zur Variablenzuweisung in JavaScript nur zu leicht geschehen, dass man versehentlich eine globale Variable erstellt. Ein Programm, das eine Zuweisung zu einer ungebundenen Variable vornimmt, meldet keinen Fehler, sondern erstellt einfach eine neue globale Variable