5 Arrays und Dictionaries

Objekte sind die vielseitigsten Datenstrukturen von JavaScript. Je nach Situation kann ein Objekt für Unterschiedliches stehen: für einen festen Eintrag einer Name-Wert-Zuweisung, für eine objektorientierte Datenabstraktion mit geerbten Methoden, für ein Array mit hoher oder geringer Dichte oder für eine Hash-Tabelle. Ein solches Vielzweckinstrument zu beherrschen, erfordert naturgemäß unterschiedliche Idiome für die verschiedenen Verwendungszwecke. Im vorangegangenen Kapitel haben wir uns die Nutzung strukturierter Objekte und die Vererbung angesehen. In diesem Kapitel geht es um die Verwendung von Objekten als *Collections*, also als zusammengefasste Datenstrukturen mit unterschiedlicher Anzahl von Elementen.

Erstellen Sie schlanke Dictionaries mit Object — Thema 43

Dictionaries

Im Grunde genommen ist ein JavaScript-Objekt eine Tabelle, die die Strings von Eigenschaftsnamen zu Werten zuweist. Dadurch sind Objekte angenehm schlank, um *Dictionaries* (»Wörterbücher«) zu implementieren. Dabei handelt es sich um Collections variabler Größe, die Strings zu Werten zuweisen. Java Script bietet sogar ein bequemes Konstrukt zum Aufzählen der Eigenschaftsnamen eines Objekts, nämlich die `for...in`-Schleife:

```
var dict = { alice: 34, bob: 24, chris: 62 };
var people = [];

for (var name in dict) {
   people.push(name + ": " + dict[name]);
}

people; // ["alice: 34", "bob: 24", "chris: 62"]
```

Da jedes Objekt aber auch die Eigenschaften seines Prototypobjekts erbt (siehe Kapitel 4), führt die `for...in`-Schleife neben den »eigenen« Eigenschaften eines Objekts auch seine geerbten Eigenschaften auf. Was aber geschieht, wenn wir eine eigene Dictionary-Klasse erstellen, die ihre Elemente als Eigenschaften des Dictionary-Objekts selbst speichert?

```
function NaiveDict() { }

NaiveDict.prototype.count = function() {
   var i = 0;
   for (var name in this) { // Zählt jede Eigenschaft
      i++;
   }
   return i;
};

NaiveDict.prototype.toString = function() {
   return "[object NaiveDict]";
};

var dict = new NaiveDict();

dict.alice = 34;
dict.bob = 24;
dict.chris = 62;

dict.count(); // 5
```

Das Problem ist, dass wir sowohl die festen Eigenschaften von `NaiveDict` (also `count` und `toString`) als auch die variablen Einträge des Dictionarys (`alice`, `bob`, `chris`) im selben Objekt speichern. Wenn `count` die Eigenschaften eines Dictionarys aufzählt, werden nicht nur die Eigenschaften berücksichtigt, an denen wir interessiert sind, sondern alle (`count`, `toString`, `alice`, `bob` und `chris`). Eine verbesserte Dictionary-Klasse namens `Dict`, die ihre Elemente nicht als Eigenschaften der Instanz speichert, sondern stattdessen die Methoden `dict.get(key)` und `dict.set(key, value)` bereitstellt, finden Sie in Thema 45. In diesem Thema konzentrieren wir uns auf die Technik, Objekteigenschaften als Dictionary-Elemente zu verwenden.

Ein ähnlicher Fehler ist es, den Typ `Array` zur Darstellung von Dictionaries zu verwenden, eine Falle, in die vor allem Programmierer tappen, die mit Sprachen wie Perl und PHP vertraut sind, da Dictionaries dort gewöhnlich als »assoziative Arrays« bezeichnet werden. Da wir zu allen Arten von JavaScript-Objekten Eigenschaften hinzufügen können, scheint diese Vorgehensweise auf trügerische Weise manchmal sogar zu funktionieren:

```
var dict = new Array();

dict.alice = 34;
dict.bob = 24;
dict.chris = 62;

dict.bob; // 24
```

Prototype Pollution

Leider ist dieser Code anfällig für eine *Prototyp-Verunreinigung* (*Prototype Pollution*), bei der die Eigenschaften eines Prototypobjekts dazu führen können, dass beim Aufzählen von Dictionary-Einträgen unerwartete Eigenschaften auftauchen. Beispielsweise kann es vorkommen, dass eine andere Bibliothek in der Anwendung einige Hilfsmethoden zu `Array.prototype` hinzufügt:

```
Array.prototype.first = function() {
   return this[0];
};

Array.prototype.last = function() {
   return this[this.length – 1];
};
```

Wenn wir die Elemente dieses Arrays aufzuzählen versuchen, geschieht Folgendes:

```
var names = [];

for (var name in dict) {
   names.push(name);
}

names; // ["alice", "bob", "chris", "first", "last"]
```

Nur direkte Object-Instanzen verwenden

Das führt uns zu der wichtigsten Regel für die Verwendung von Objekten als schlanke Dictionaries: Nutzen Sie ausschließlich direkte Instanzen von `Object` als Dictionaries – keine Subklassen wie `NaiveDict` und auf keinen Fall Arrays. So können wir beispielsweise `new Array()` in vorstehendem Code durch `new Object()` oder sogar durch ein leeres Objektliteral ersetzen. Das Ergebnis ist deutlich weniger empfindlich gegenüber Prototyp-Verunreinigung:

```
var dict = {};

dict.alice = 34;
dict.bob = 24;
dict.chris = 62;

var names = [];
```

```
for (var name in dict) {
   names.push(name);
}

names; // ["alice", "bob", "chris"]
```

Völlig ausschließen lässt sich eine Verunreinigung aber auch bei dieser neuen Version nicht. Es könnte immer noch jemand daherkommen und Eigenschaften zu `Object.prototype` hinzufügen, wodurch wir wieder vor demselben Problem stehen würden. Da wir aber eine unmittelbare Instanz von `Object` verwenden, ist dieses Risiko einzig und allein auf `Object.prototype` beschränkt.

Warum ist die neue Lösung besser? Wie in Thema 47 erklärt, sollte niemand jemals irgendwelche Eigenschaften zu `Object.prototype` hinzufügen, wohingegen es durchaus sinnvoll sein mag, `Array.prototype` um neue Eigenschaften zu ergänzen. Beispielsweise haben Sie in Thema 42 erfahren, wie Sie zu `Array.prototype` Standardmethoden hinzufügen, die in manchen Umgebungen nicht implementiert sind. Solche Eigenschaften verunreinigen letzten Endes die `for...in`-Schleife. Auch benutzerdefinierte Klassen weisen gewöhnlich Eigenschaften in ihrem Prototyp auf. Wenn Sie bei direkten Instanzen von `Object` bleiben (und sich stets an die Empfehlung aus Thema 47 halten), dann bleiben Ihre `for...in`-Schleifen frei von Verunreinigungen.

Aber Obacht! Wie die Themen 44 und 45 zeigen, ist diese Regel eine notwendige, aber keine hinreichende Bedingung, um ordnungsgemäß funktionierende Dictionaries zu erstellen. So bequem schlanke Dictionaries auch sein mögen, so bergen sie doch eine Reihe von Gefahren. Es ist daher wichtig, sich alle drei Themen anzusehen. Falls Sie sich die Regeln nicht merken wollen, können Sie auch eine Abstraktion wie die Klasse `Dict` aus Thema 45 verwenden.

Was Sie sich merken sollten

- Verwenden Sie zur Konstruktion von schlanken Dictionaries Objektliterale.
- Als Schutz gegen eine Prototyp-Verunreinigung in `for...in`-Schleifen sollten schlanke Dictionaries unmittelbare Abkömmlinge von `Object.prototype` sein.