

JavaScript – this

Ingo Köster

Diplom-Informatiker (FH)

this als globaler Kontext

- › this zeigt standardmäßig auf das globale Objekt window
- › Beispiel aus der Browser Konsole
 - › **this**
 - › Window {top: Window, window: Window, location: Location, external: Object, chrome: Object...}
 - › **window**
 - › Window {top: Window, window: Window, location: Location, external: Object, chrome: Object...}
 - › **this === window**
 - › true

this als globaler Kontext

- › Für alle Funktion die im globalen Kontext aufgerufen werden, ist `this` gleich `window`

```
function foo()  
{  
  return this;  
}
```

- › Aufruf und Ausgabe

```
foo();
```

```
Window {top: Window, window: Window, location: Location, external:  
  Object, chrome: Object...}
```

this in Objekten

- › Innerhalb einer Methode, die über `objekt.funktion()` aufgerufen wird, verweist `this` auf das Objekt

```
class MyClass {  
    x = 42;  
    ausgabe() { return this; }  
}
```

- › Aufruf

```
let m = new MyClass();  
m.ausgabe();           // MyClass {x: 42}
```

this im Strict Mode bei globalen Funktionen

- › Bei Funktionsaufrufen ohne den Strict Mode ist `this` in der Funktion das globale Objekt

- › Beim Versuch Methoden und/oder Eigenschaften für `this` zu definieren, werden globale Variablen und Funktionen erzeugt!!!

```
function foo() {  
  return this; // globales Objekt  
}
```

- › Im Strict Mode ist `this` in Funktionsaufrufen `undefined`

```
function foo() {  
  "use strict";  
  return this; // undefined  
}
```


"use strict" Beispiel

```
// Globale Variable
a = 5; // Bitte nicht nachmachen !!!
function foo()
{
  "use strict";
  let a = 10;
  console.log(a);           // Ausgabe 10
  console.log(window.a);    // Ausgabe 5
  console.log(this.a);      // Fehler! Kein Zugriff
                             // auf undefined.a
}
foo();
```

Bindung von this bei Lambda Ausdrücken

- › Anders als „normale“ Funktionen in JavaScript, haben Lambdas keine eigene Definition einer this-Referenz, sondern binden die this-Referenz an den umschließenden Kontext:

```
function Person() {  
    this.age = 0;  
    setInterval( () => { this.age++; }, 1000);  
}
```



Event-Handler und this

› Bei Event-Handler Funktionen verweist `this` auf das Element, welches den Event auslöst

› Beispiel

```
<p id="hi"> Click me </p>
```

```
...
```

```
document.getElementById("hi").onclick = function (event)
{
    alert(event.target === this); // true
}
```


Event-Handler und this

- › Sollen Methoden von Objekten als Event-Handler dienen ist dieses Verhalten etwas hinderlich

```
class MyClass {  
  x = 5;  
  myFunction() { console.log(this.x); }  
}  
...  
let m = new MyClass();  
document.getElementById('myid').onclick = m.myFunction;
```

- › Beim Klick wird die Ausgabe des Wertes von x (5) erwartet, es wird jedoch undefined ausgegeben

Event-Handler und this

- › Abhilfe schaffen Lambda Ausdrücke oder Funktionen
- › Beispiel Lambda Ausdruck
 - › `let m = new MyClass();`
 - › `document.getElementById('myid').onclick = () => m.myFunction();`
- › In dem Aufruf von `myFunction` verweist `this` jetzt wieder auf das aufrufende Objekt, d.h. auf `m`