Javascript Sprachfeatures
Objekte

https://iuk.one/1066-1108

Clemens H. **Cap**
ORCID: 0000-0003-3958-6136

Department of Computer Science
University of **Rostock**
Rostock,Germany
clemens.cap@uni-rostock.de

Version 1

1. Objekte

Arten von Objekten

1. Objekte
2. Proxies
3. Symbole
4. Objektorientierung
5. Identität, Gleichheit, Cloning

Objekte

Javascript kennt **drei Arten** von Objekten.

"Alte" Objekte

Sind Key-Value Maps.

"Neue" Objekte

Haben Eigenschaften mit setter & getter Methoden.
Können als aufzählbar, schreibbar, veränderbar deklariert werden.
Siehe `Object.defineProperty`

"Proxy" Objekte

Virtualisieren Objektzugriffe.
Jede Anwendung von Methoden auf Objekte kann auf Stellvertreter umgebogen werden.
Bsp: set, get, delete, apply

Neue Objekte

Aufzählbare Eigenschaften (*enumerable*):

- Interne Meta-Eigenschaft einer Objekt-Eigenschaft.
- Etliche Konstrukte nutzen nur aufzählbare Eigenschaften: `for...in`, `Object.keys()`
- Etabliert: `Object.defineProperty(obj, prop, descriptor)`
- Introspektion: `propertyIsEnumerable()`

Eigene versus geerbte Eigenschaften

- Eigene: Gehören zum Objekt selber
- Geerbte: Liegen auf der `prototype` Chain.
- Unterscheiden:

  `Object.hasOwn()`

Erweitern von Objekten

Typischerweise gilt:

- Objekte sind key-value maps bzw. dictionaries.
- Objekte haben keine festen Schemata.
- Objekte können erweitert werden. Wert einfach setzen.
- Objekte können reduziert werden. `delete obj.zuLoeschen`

Probleme: Gelegentlich zu hohe Flexibilität (vgl. Typsystem)

- Nicht immer soll Anwendung beliebig patchbar sein.
- Nur wenig Konsistenzgarantien möglich.

Einschränkungen:

- **Erweiterungsverbot:** prevent extension
- **Versiegeln:** seal
- **Einfrieren:** freeze
- **Änderungsverbot:** immutable

Erweiterungsverbot

Erstellen: `Object.preventExtensions (object);`
Überprüfen: `Object.isExtensible (object);`

1. Unwiderrufliche Veränderung eines Objekts.
2. Eigenschaften können nicht mehr hinzugefügt werden.
3. Eigenschaften können aber weiterhin gelöscht werden.
4. Prototyp des Objekts kann nicht mehr gewechselt werden.
5. Prototyp-Objekt selber kann noch erweitert werden.
6. Ggf. ein deep prevent Extensions in verbundene Objekte & Prototyp erforderlich

Versiegeln von Objekten

Erstellen: `Object.seal (object);`
Überprüfen: `Object.isSealed (object);`

1. Verbietet Erweiterungen (siehe dort) und *zusätzlich:*
2. Eigenschaften können nicht mehr gelöscht werden.
3. Aufzählbarkeit kann nicht mehr verändert werden.
4. Eigenschaft kann nicht mehr konfiguriert werden, dh:
   Accessoren können nicht mehr veränder werden.
   Schreibbarkeit kann nicht mehr verändert werden.
5. Schreibbare Eigenschaften können weiterhin verändert werden.

Einfrieren von Objekten

Erstellen: `Object.freeze (object);`
Überprüfen: `Object.isFrozen (object);`

1. Verbietet Erweiterungen (siehe dort).
2. Versiegelt das Objekt (seieh dort) und *zusätzlich:*
3. Eigenschaften können nicht mehr verändert werden.
4. Stärkste Form des Objekt-Schutzes.

Immutability

Begriff taucht in JS in zwei Kontexten auf.

Als JS Sprachkonzept:

1. Eigenschaft, die nicht mehr verändert werden kann.
2. Etabliert durch Einfrieren oder durch Property Descriptoren.
3. Siehe `Object.defineProperty()`

Als Design-Konzept:

Immutability als Design-Konzept

Idee: Objekte niemals modifiziert sondern stets neu konstruiert.

Bewertung:

- Keine Buchhaltung darüber, was sich geändert hat.
- Keine Inkonsistenzen von Teilen der Anwendung.
- Nur unidirektionaler Datenfluß durch store und view Komponenten.
- Aber: Kann höheren Kopieraufwand bewirken.
- Aber: Kann durch Implementierung wieder abgefangen werden.

Anwendungen:

- Redux: Anwendung hat nur einen (immutable) Zustandsspeicher.
- Flux: Anwendung kann mehrere immutable Zustandsspeicher haben.
- React: Hat sich aus der Redux Idee heraus entwickelt.
- immutable-js: Unterstützt komplexere Datenstrukturen.

2. Proxies

Stellvertreter für interne Methoden.

Javascript Meta-Programming Hooks.

1. Objekte
2. Proxies
3. Symbole
4. Objektorientierung
5. Identität, Gleichheit, Cloning

Interne Methoden

Jedes Objekt hat interne Methoden als Schnittstelle in die Implementierung.

1. `GetPrototypeOf`
2. `SetPrototypeOf`
3. `IsExtensible`
4. `PreventExtensions`
5. `GetOwnPropery`
6. `DefineOwnProperty`
7. `HasProperty`
8. `Get`
9. `Set`
10. `Delete`
11. `OwnPropertyKeys`
12. `Call`
13. `Construct`

Jede der internen Methoden kann via Prox ersetzt werden.

Forwarding Proxy

```
1 const target = {};
2 const p = new Proxy(target, {});
3
4 p.a = 42;              // Delegation an target Objekt
5
6 console.log(target.a); // 42  Nachweis der Delegation
```

Src. 1: Forwarding Proxy

Hiding / Unhiding Proxy

```
1  class Secret {
2    #secret;
3    constructor(secret) {this.#secret = secret;}
4    get secret() {return this.#secret.replace(/\d+/, "[GEHEIM]");} // hiding get
5  }
6
7  const aSecret = new Secret("123456");
8  console.log(aSecret.secret); // [GEHEIM]
9
10 const proxy = new Proxy(aSecret, {});
11 console.log(proxy.secret);  // TypeError: Cannot read private member #secret
12
13 const proxy = new Proxy(aSecret, {
14   get(target, prop, receiver) {return target[prop];} // unhiding via target
15 });
16 console.log(proxy.secret);  // 123456
```

Src. 2: Hiding / Unhiding Proxy

Validating Proxy

```
1  const validator = {
2    set(obj, prop, value) {
3      if (prop === "age") {
4        if (!Number.isInteger(value)) {throw new TypeError("Age is not an integer");}
5        if (value > 200)              {throw new RangeError("Age improbable");}
6      }
7    obj[prop] = value;     // The default behavior to store the value
8      return true;       // Indicate success
9    }
10 };
11
12 const person = new Proxy({}, validator);
13
14 person.age = 100;
15 console.log(person.age); // 100
16 person.age = "young";     // Throws an exception
17 person.age = 300;         // Throws an exception
```

Src. 3: Validating Proxy

Anwendungen von Proxies

Anforderungen:

- Template Engine greift auf Kontext-Objekte zu.
- Jeder Zugriff soll protokolliert werden.

Idee 1: On the fly Datenflussanalyse des Templates

Idee 2: Faulting in aus cache / DB / server.

Proxy Objekte stellen dazu erforderliche Konzepte bereit.

3. Symbole

Neuer primitiver Datentyp für die Meta-Programmierung

1. Objekte
2. Proxies
3. Symbole
4. Objektorientierung
5. Identität, Gleichheit, Cloning

Was sind Symbole?

- Neuer primitiver Datentyp.
- Typische Nutzung: Als `key` in Objekten.
- Nutzung für Meta-Programmierung

Zwei wichtige Anwendungsfälle.

Anwendung von Symbolen (1): Meta-Daten Keys

Frage: Wie bekomme ich einen neuen `key` für Meta-Daten, der garantiert nie mit einem vom User bereits verwendeten Namen kollidiert?

```
1 const PERSISTENT = Symbol ("persistent");
2 obj[PERSISTENT] = true;
```

Src. 4: Symbol-Nutzung: Meta-Daten Keys

Beachte: Neuer Aufruf von `Symbol ("persistent")` liefert ein neues, anderes, wiederum kollisionsfreies Symbol.

Anwendung von Symbolen (2): Synchronisation von Keys

Frage: Wie synchronisiere ich 2 Programmteile auf dassselbe Symbol?

Lösung 1: In globaler Variablen ablegen (unschön).

Lösung 2: Symbol über Symbol Registry suchen.

```
1 Symbol.for ("persistent");
```

Src. 5: Symbol-Nutzung: Key Synchronisation

4. Objektorientierung

Klassen mit und ohne Klassen

1. Objekte
2. Proxies
3. Symbole
4. Objektorientierung
5. Identität, Gleichheit, Cloning

Syntaktische Ergänzung durch Klassen

```
1  class Shape {
2      constructor (id, x, y) {this.id = id; this.move(x, y);}
3      move (x, y) {this.x = x; this.y = y;}
4  }
5
6  Bisher
7  var Shape = function (id, x, y) {this.id = id; this.move(x, y);};
8  Shape.prototype.move = function (x, y) {this.x = x; this.y = y;};
```

Src. 6: Syntaktische Ergänzung durch Klassen

Syntaktische Ergänzung durch Vererbung

```
1  class Rectangle extends Shape {
2   constructor (id, x, y, width, height) {
3     super(id, x, y);
4     this.width = width;
5     this.height = height;
6    }
7  }
8
9  class Circle extends Shape {
10   constructor (id, x, y, radius) {
11     super(id, x, y);
12     this.radius = radius;
13   }
14 }
```

Src. 7: Syntaktische Ergänzung durch Vererbung

Vererbung ohne class und extends

```
1  var Rectangle = function (id, x, y, width, height) {
2      Shape.call(this, id, x, y);
3      this.width  = width;
4      this.height = height;
5  };
6  Rectangle.prototype = Object.create(Shape.prototype);
7  Rectangle.prototype.constructor = Rectangle;
8  var Circle = function (id, x, y, radius) {
9      Shape.call(this, id, x, y);
10     this.radius = radius;
11 };
12 Circle.prototype = Object.create(Shape.prototype);
13 Circle.prototype.constructor = Circle;
```

Src. 8: Vererbung ohne class und extends

5. Identität, Gleichheit, Cloning

Was ist die Identität eines Objekts?

1. Objekte
2. Proxies
3. Symbole
4. Objektorientierung
5. Identität, Gleichheit, Cloning

Identität und Gleichheit

Identität: Für jedes Objekt über die Adresse gegeben.

Gleichheit: Primitive Werte sind gleich, Struktur ist isomorph.

Tragfähige Definition nur in formalen Modellen möglich.

Drei Arten von Vergleichen

Lose Gleichheit: `a == b`

- Führt eine Typ-Konvertierung vor dem Vergleich durch. `0 == "0"`
- Beachtet die IEEE754 Norm: `NaN != NaN` und `-0 == +0`
- Beachtet diverse weitere Regeln: `null == undefined` und `false == 0`

Strikte Gleichheit: `a === b`

- Führt keinerlei Typ-Konvertierungen durch.
- Beachtet die IEEE754 Norm: `NaN != NaN` und `-0 == +0`

Werte-Gleichheit: `Objekt.is (a, b)`

- Führt keinerlei Typ-Konvertierungen durch.
- Beachtet die IEEE754 Sonderregelungen nicht.

Serialisierung

Serialisierung:	Umwandlung eines *strukturierten* Datentyps in einen *linearen* Datentyp.
String-Serialisierung:	Umwandlung eines Objekts in einen String.
Deserialisierung:	Umkehrung der Operation.

Anwendungen der Serialisierung:

- **Ausgabe von Objekten** für Debugging, Tracing.
- **Transfer von Objekten** quer über Kontextgrenzen hinweg.
  Bsp: Call Stacks, Prozeßgrenzen, Protection Grenzen, Speicherverwaltungen, remote Maschinen Worker, Service Worker, andere Browser Tabs & Fenster andere Anwendungen
- **Persistieren von Objekten** auf Festspeicher.

Hindernisse der Serialisierung

Problem: Nicht jedes Objekt ist serialisierbar.

Teilweise behebbare Schwierigkeiten:

1. **Funktionen:** Code via `Function.prototype.toString()`
2. **Zyklische Objekte:** Eigenes Libraries nötig.
3. **Closures:** Komme in Javascript nicht an den Scope Pointer. Manuelles Programmierung von Accessoren.

Nicht behebbare Schwierigkeiten:

1. **Native Funktionen:** Sind in C(++) geschrieben.
2. **Native Objekte:** Referenzen zu JS Stubs von C(++) Objekten sind nicht verhaltensgleich.

Zyklische Objekte(1): Replacer / Reviver

```
1 obj.itself = obj;                    // Objekt zyklisch machen
2 function replacer(key, value) {      // Replacer definieren
3   return (key === 'itself' ? null : value);
4 }
5 const stringified = JSON.stringify(obj, replacer);
6 console.log(stringified);
```

Src. 9: Nutzung des replacer Mechanismus

Siehe auch:

- https://dillionmegida.com/p/second-argument-in-json-stringify/
- https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/JSON/stringify
- https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/JSON/parse

Umkehrung: Passend zum replacer beim Serialisieren gibt es einen reviver beim Deserialisieren.

Zyklischer Objekte (2): Spezielle Libraries

Bibliothek zum (De)serialisieren zyklischer Objekte:

- Nutzt Marker, damit Algorithmus trotz Schleifen terminiert.
- https://github.com/slashman/circular
- https://blog.slashie.net/2014/08/08/circularjs-1-1-tutorial/

Einfaches Kopieren

```
Object.assign (target, source);
```

- Kopiert alle eigenen aufzählbaren Eigenschaften von `source` auf `target`.
- Nutzt an der Quelle `getter` und im Ziel `setter`.
- Setzt Eigenschaften, konfiguriert diese aber nicht.
- Nicht-aufzählbare Eigenschaften und Prototyp-Eigenschaften werden nicht kopiert.
- Kopiert bei Objekten Referenzen, macht kein deep cloning.
- Kann mehrere Quellobjekte haben:
  `Object.assign (target, source1, source2, source3);` Weiter rechts stehende Quellen werden später kopiert.

Shallow Copy

Eigenschaften eines Shallow Copy

- Hat dieselben primitiven Werte wie das Original.
- Hat dieselben Objekt-Referenzen wie das Original.
- Änderung eines Wertes in einem referenzierten Objekt ändert den Wert in der Kopie und im Original.
- Interne Kopier-Operationen machen typischerweise shallow copy: `Array.prototype.concat`, `Array.prototype.slice`, `Array.from`, `Object.assign`

Durchführen eines Shallow Copy:

1. Object.assign
2. Object Spread Operator
3. Object Rest Operator

```
const clone = Object.assign( {}, org);
```

Object Spread Operator

```
1  // Anwendung
2  const clone = { ...obj };
3
4  // Beispiel:
5  const obj = { name:"Gero", job:"Prof"};
6  const profClone = { ...obj };
7  console.log (profClone);  // { name:"Gero", job:"Prof"}
8  profClone === obj         // false
9
10 // Beispiel mit Update:
11 const profClone2 = { ...obj, Ort:"HRO"};
12 console.log (profClone2); ; // { name:"Gero", job:"Prof", Ort:"HRO"}
```

Src. 10: Object Spread Operator erlaubt Updating

Object Rest Operator

```
1  const { ...clone } = obj;
2
3  // Beispiel:
4  const obj = { name:"Gero", job:"Prof"};
5  const { ...profClone } = obj;
6  console.log (profClone);  // { name:"Gero", job:"Prof"}
7  profClone === obj         // false
8
9  // Beispiel mit Skip:
10 const { name, ...profClone2 } = obj;
11 console.log (profClone2); ; // {job:"Prof"}
```

Src. 11: Objetc Rest Operator erlaubt Skipping.

Object Spread und Object Rest Operator kombiniert

```
1  // Beispiel:
2  const obj = { name:"Gero", job:"Prof"};
3
4  const { name, ...profClone } = { ...obj, Ort:"HRO"};
5  console.log (profClone);   // {job:"Prof", Ort:"HRO"}
```

Src. 12: Kombination: Spread mit Updating und Rest mit Skipping

Deep Copy

Eigenschaften eines Deep Copy:

- Hat dieselben primitiven Werte wie das Original.
- Hat eigenständige, neue Objekt-Referenzen wie das Original und dort aber wieder dieselben primitiven Werte.
- Bei Veränderungen keine Seiteneffekte in das Original.

Durchführen eines Deep Copy:

1. Serialisieren in eine String, String wieder deserialisieren. (Cave: Zyklische Objekte)
2. Structured Cloning Algorithmus.

Structured Cloning Algorithmus

```
const twin = structuredClone (obj);
```

- Von Javascript intern benutzte Funktion des Deep Cloning.
- Funktioniert auch für zirkuläre Objekte.
- Funktioniert für sehr viele Datentypen.
- https://developer.mozilla.org/en-US/docs/Web/API/Web_Workers_API/Structured_clone_algorithm#supported_types

Shallow und Deep Freeze

Jede Funktion, die Objekte traversiert, kennt die Unterscheidung shallow / deep.

Also beispielsweise auch: Freeze.

Diverse weitere Aspekte können eine Rolle spielen:

https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Object/freeze

Anhang

Übersicht

Übersicht

Programmquellenverzeichnis Prog

Rechtliche Hinweise §

Zitierweise dieses Dokuments →

Verzeichnis aller Folien

Programmquellenverzeichnis (1/2)

1 Forwarding Proxy 13
2 Hiding / Unhiding Proxy 14
3 Validating Proxy 15
4 Symbol-Nutzung: Meta-Daten Keys 19
5 Symbol-Nutzung: Key Synchronisation 20
6 Syntaktische Ergänzung durch Klassen 22
7 Syntaktische Ergänzung durch Vererbung 23
8 Vererbung ohne class und extends 24
9 Nutzung des replacer Mechanismus 30
10 Object Spread Operator erlaubt Updating 34

11 Objetc Rest Operator erlaubt Skipping 35
12 Kombination: Spread mit Updating und Rest mit Skipping 36

Rechtliche Hinweise (1)

Die hier angebotenen Inhalte unterliegen deutschem Urheberrecht. Inhalte Dritter werden unter Nennung der Rechtsgrundlage ihrer Nutzung und der geltenden Lizenzbestimmungen hier angeführt. Auf das Literaturverzeichnis wird verwiesen. Das Zitatrecht in dem für wissenschaftliche Werke üblichen Ausmaß wird beansprucht. Wenn Sie eine Urheberrechtsverletzung erkennen, so bitten wir um Hinweis an den auf der Titelseite genannten Autor und werden entsprechende Inhalte sofort entfernen oder fehlende Rechtsnennungen nachholen. Bei Produkt- und Firmennamen können Markenrechte Dritter bestehen. Verweise und Verlinkungen wurden zum Zeitpunkt des Setzens der Verweise überprüft; sie dienen der Information des Lesers. Der Autor macht sich die Inhalte, auch in der Form, wie sie zum Zeitpunkt des Setzens des Verweises vorlagen, nicht zu eigen und kann diese nicht laufend auf Veränderungen überprüfen.

Alle sonstigen, hier nicht angeführten Inhalte unterliegen dem Copyright des Autors, Prof. Dr. Clemens Cap, ©2020. Wenn Sie diese Inhalte nützlich finden, können Sie darauf verlinken oder sie zitieren. Jede weitere Verbreitung, Speicherung, Vervielfältigung oder sonstige Verwertung außerhalb der Grenzen des Urheberrechts bedarf der schriftlichen Zustimmung des Rechteinhabers. Dieses dient der Sicherung der Aktualität der Inhalte und soll dem Autor auch die Einhaltung urheberrechtlicher Einschränkungen wie beispielsweise Par 60a UrhG ermöglichen.

Die Bereitstellung der Inhalte erfolgt hier zur persönlichen Information des Lesers. Eine Haftung für mittelbare oder unmittelbare Schäden wird im maximal rechtlich zulässigen Ausmaß ausgeschlossen, mit Ausnahme von Vorsatz und grober Fahrlässigkeit. Eine Garantie für den Fortbestand dieses Informationsangebots wird nicht gegeben.

Die Anfertigung einer persönlichen Sicherungskopie für die private, nicht gewerbliche und nicht öffentliche Nutzung ist zulässig, sofern sie nicht von einer offensichtlich rechtswidrig hergestellten oder zugänglich gemachten Vorlage stammt.

Use of Logos and Trademark Symbols: The logos and trademark symbols used here are the property of their respective owners. The YouTube logo is used according to brand request 2-9753000030769 granted on November 30, 2020. The GitHub logo is property of GitHub Inc. and is used in accordance to the GitHub logo usage conditions https://github.com/logos to link to a GitHub account. The Tweedback logo is property of Tweedback GmbH and here is used in accordance to a cooperation contract.

Rechtliche Hinweise (2)

Disclaimer: Die sich immer wieder ändernde Rechtslage für digitale Urheberrechte erzeugt ein nicht unerhebliches Risiko bei der Einbindung von Materialien, deren Status nicht oder nur mit unverhältnismäßig hohem Aufwand abzuklären ist. Ebenso kann den Rechteinhabern nicht auf sinnvolle oder einfache Weise ein Honorar zukommen, obwohl deren Leistungen genutzt werden.

Daher binde ich gelegentlich Inhalte nur als Link und nicht durch Framing ein. Lt EuGH Urteil 13.02.2014, C-466/12 (Pressemitteilung, Blog-Beitrag, Urteilstext). ist das unbedenklich, da die benutzten Links ohne Umgehung technischer Sperren auf im Internet frei verfügbare Inhalte verweisen.

Wenn Sie diese Rechtslage stört, dann setzen Sie sich für eine Modernisierung des völlig veralteten Vergütungs- und Anreizsystems für urheberrechtliche Leistungen ein. Bis dahin klicken Sie bitte auf die angegebenen Links und denken Sie darüber nach, warum wir keine für das digitale Zeitalter sinnvoll angepaßte Vergütungs- und Anreizsysteme digital erbrachter Leistungen haben.

Zu Risiken und Nebenwirkungen fragen Sie Ihren Rechtsanwalt oder Gesetzgeber.

Weitere Hinweise finden Sie im Netz hier und hier oder hier.

Zitierweise dieses Dokuments

Wenn Sie Inhalte aus diesem Werk nutzen oder darauf verweisen wollen, zitieren Sie es bitte wie folgt:

Clemens H. Cap: Javascript Sprachfeatures
Objekte. Electronic document. https://iuk.one/1066-1108 14. 6. 2023.

Bibtex Information: https://iuk.one/1066-1108.bib

```
@misc{doc:1066-1108,
   author        = {Clemens H. Cap},
   title         = {Javascript Sprachfeatures
Objekte},
   year          = {2023},
   month         = {6},
   howpublished  = {Electronic document},
   url           = {https://iuk.one/1066-1108}
}
```

Typographic Information:
Typeset on ?today?
This is pdfTeX, Version 3.14159265-2.6-1.40.21 (TeX Live 2020) kpathsea version 6.3.2
This is pgf in version 3.1.5b
This is preamble-slides.tex myFormat©C.H.Cap

Verzeichnis aller Folien

Titelseite 1

1. Objekte

Objekte 3
Neue Objekte 4
Erweitern von Objekten 5
Erweiterungsverbot 6
Versiegeln von Objekten 7
Einfrieren von Objekten 8
Immutability 9
Immutability als Design-Konzept 10

2. Proxies

Interne Methoden 12
Forwarding Proxy 13
Hiding / Unhiding Proxy 14
Validating Proxy 15
Anwendungen von Proxies 16

3. Symbole

Was sind Symbole? 18
Anwendung von Symbolen (1): Meta-Daten Keys 19
Anwendung von Symbolen (2): Synchronisation von Keys . . . 20

4. Objektorientierung

Syntaktische Ergänzung durch Klassen 22
Syntaktische Ergänzung durch Vererbung 23
Vererbung ohne class und extends 24

5. Identität, Gleichheit, Cloning

Identität und Gleichheit 26
Drei Arten von Vergleichen 27
Serialisierung 28
Hindernisse der Serialisierung 29
Zyklische Objekte(1): Replacer / Reviver 30
Zyklischer Objekte (2): Spezielle Libraries 31
Einfaches Kopieren 32
Shallow Copy 33
Object Spread Operator 34
Object Rest Operator 35
Object Spread und Object Rest Operator kombiniert 36
Deep Copy 37
Structured Cloning Algorithmus 38
Shallow und Deep Freeze 39

Legende:
Fortsetzungsseite
Seite ohne Überschrift
Bildseite