

Softwaretechnik

Lecture 16: Specification with Types

Peter Thiemann

University of Freiburg, Germany

16.07.2012

Contents

Specification with Types

Excursion: Scripting Languages

Ultra-Brief JavaScript Tutorial

Thesis

Excursion to a World Without Types: Scripting Languages

Scripting Languages

- ▶ Lightweight programming languages evolved from command languages
- ▶ Lightweight data structures
hashmap (object), strings
- ▶ Lightweight syntax
familiar, no semicolon, (often not well specified), ...
- ▶ Lightweight typing
dynamic, weak, duck typing
- ▶ Lightweight metaprogramming
- ▶ Lightweight implementation
interpreted, few tools

JavaScript, a Typical Scripting Language

- ▶ Initially developed by Brendan Eich of Netscape Corp.
- ▶ Standardized as ECMAScript (ECMA-262 Edition 5.1)
- ▶ Application areas (scripting targets)
 - ▶ client-side web scripting (dynamic HTML, SVG, XUL, GWT)
 - ▶ server-side scripting (Whitebeam, Cocoon, iPlanet, nodejs)
 - ▶ animation scripting (diablo, dim3, k3d)
 - ▶ cloud scripting (Google Apps Script)

JavaScript, Technically

- ▶ Java-style syntax
- ▶ Object-based imperative language
 - ▶ no classes, but prototype concept
 - ▶ objects are hashtables
- ▶ First-class functions
 - ▶ a functional language
- ▶ Weak, dynamic type system

Slogan: Any type can be converted to any other reasonable type

```
node.onmouseout =  
function (ev) {  
  init();  
  state++;  
  node.className =  
    "highlight-"  
    + state;  
  ev.stopPropagation();  
};
```

Problems with JavaScript

Symptomatic for other scripting languages

- ▶ No module system
 - ▶ No namespace management
 - ▶ No interface descriptions
 - ▶ No static type system
 - ▶ No application specific datatypes
primitive datatypes, strings, hashtables
 - ▶ Type conversions are sometimes surprising
“A scripting language should never throw an exception [the script should just continue]” (Rob Pike, Google)
- ⇒ Conceived for small applications

Specific Problems with JavaScript

- ▶ Most popular applications
 - ▶ client-side scripting
 - ▶ AJAX
- ▶ Dynamic modification of page content via DOM interface
 - ▶ DOM = document object model
 - ▶ W3C standard interface for accessing and modifying XML
 - ▶ Mainly used in web browsers

Specific Problems with JavaScript

- ▶ Most popular applications
 - ▶ client-side scripting
 - ▶ AJAX
 - ▶ Dynamic modification of page content via DOM interface
 - ▶ DOM = document object model
 - ▶ W3C standard interface for accessing and modifying XML
 - ▶ Mainly used in web browsers
 - ▶ Incompatible DOM implementations in Web browsers
- ⇒ programming recipes instead of techniques
- ⇒ platform independent libraries like jQuery
- ▶ Security holes via dynamically loaded code or XSS
- ⇒ sandboxing, analysis

Can You Write Reliable Programs in JavaScript?

- ▶ Struggle with the lack of e.g. a module system
 - ▶ Ad-hoc structuring of large programs
 - ▶ Naming conventions
 - ▶ Working in a team
- ▶ Work around DOM incompatibilities
 - ▶ Use existing JavaScript frameworks (widgets, networking)
 - ▶ Frameworks are also incompatible
- ▶ Wonder about unexpected results

An Ultra-Brief JavaScript Tutorial

Rule 1:

JavaScript is object-based. An object is a hash table that maps named properties to values.

An Ultra-Brief JavaScript Tutorial

Rule 1:

JavaScript is object-based. An object is a hash table that maps named properties to values.

Rule 2:

Every value has a type. For most reasonable combinations, values can be converted from one type to another type.

An Ultra-Brief JavaScript Tutorial

Rule 1:

JavaScript is object-based. An object is a hash table that maps named properties to values.

Rule 2:

Every value has a type. For most reasonable combinations, values can be converted from one type to another type.

Rule 3:

Types include `null`, `boolean`, `number`, `string`, `object`, and `function`.

An Ultra-Brief JavaScript Tutorial

Rule 1:

JavaScript is object-based. An object is a hash table that maps named properties to values.

Rule 2:

Every value has a type. For most reasonable combinations, values can be converted from one type to another type.

Rule 3:

Types include `null`, `boolean`, `number`, `string`, `object`, and `function`.

Rule 4:

'Undefined' is a value (and a type).

Some Quick Questions

Let's define an object `obj`:

```
js> var obj = { x: 1 }
```

What are the values/outputs of

- ▶ `obj.x`
- ▶ `obj.y`
- ▶ `print(obj.y)`
- ▶ `obj.y.z`

Answers

```
js> var obj = {x:1}
```

```
js> obj.x
```

```
1
```

```
js> obj.y
```

```
js> print(obj.y)
```

```
undefined
```

```
js> obj.y.z
```

```
js: "<stdin>", line 12: uncaught JavaScript exception:  
ConversionError: The undefined value has no properties.  
(<stdin>; line 12)
```


Weak, Dynamic Types in JavaScript II

Rule 5:

An object is really a dynamic mapping from strings to values.

```
js> var x = "x"
js> obj[x]
1
js> obj.undefined = "gotcha"
gotcha
js> obj[obj.y]
```

What is the effect/result of the last expression?

Weak, Dynamic Types in JavaScript II

Rule 5:

An object is really a dynamic mapping from strings to values.

```
js> var x = "x"
js> obj[x]
1
js> obj.undefined = "gotcha"
gotcha
js> obj[obj.y]
== obj[undefined]
== obj["undefined"]
== obj.undefined
== "gotcha"
```

Weak, Dynamic Types in JavaScript III

Recall Rule 2:

Every value has a type. For most reasonable combinations, values can be converted from one type to another type.

```
js> var a = 17
```

```
js> a.x = 42
```

```
42
```

```
js> a.x
```

What is the effect/result of the last expression?

Weak, Dynamic Types in JavaScript III

Wrapper objects for numbers

```
js> m = new Number (17); n = new Number (4)
js> m+n
21
```

Weak, Dynamic Types in JavaScript III

Wrapper objects for numbers

```
js> m = new Number (17); n = new Number (4)
js> m+n
21
```

Wrapper objects for booleans

```
js> flag = new Bool(false);
js> result = flag ? true : false;
```

What is the value of `result`?

Weak, Dynamic Types in JavaScript IV

Rule 6:

Functions are first-class, but behave differently when used as methods or as constructors.

```
js> function f () { return this.x }
```

```
js> f()
```

```
x
```

```
js> obj.f = f
```

```
function f() { return this.x; }
```

```
js> obj.f()
```

```
1
```

```
js> new f()
```

```
[object Object]
```

Distinguishing Absence and Undefinedness I

```
js> obju = { u : {}.xx }
```

```
[object Object]
```

```
js> objv = { v : {}.xx }
```

```
[object Object]
```

```
js> print(obju.u)
```

```
undefined
```

```
js> print(objv.u)
```

```
undefined
```

Distinguishing Absence and Undefinedness II

Rule 7:

The `with` construct puts its argument object on top of the current environment stack.

```
js> u = "defined"
defined
js> with (obju) print(u)
undefined
js> with (objv) print(u)
defined
```


Distinguishing Absence and Undefinedness III

Rule 8:

The `for` construct has an `in` operator to range over all defined indexes.

```
js> for (i in obju) print(i)
```

```
u
```

```
js> for (i in objv) print(i)
```

```
v
```

```
js> delete objv.v
```

```
true
```

```
js> for (i in objv) print(i)
```

```
js> delete objv.v
```

```
true
```

Thesis

- ▶ Common errors such as
 - ▶ using non-objects as objects
e.g. using numbers as functions
 - ▶ invoking non-existing methods
 - ▶ accessing non-existing fields
 - ▶ surprising conversions

can all be caught by a

Static Type System

- ▶ and much more.