JSON, ADsafe, and Misty

Douglas Crockford

Yahoo!

JSON was the first safe subset

It starts with JavaScript array literals and object literals, and removes all behavior, yielding a convenient data format.

http://www.JSON.org/

ADsafe

A system for safe web advertising.

http://www.ADsafe.org/

Static validation only,
no code rewriting.

No impact on performance.

JSLint is an ADsafe validator.

ADsafe

- ADsafe is a JavaScript subset that adds capability discipline by deleting features that cause capability leakage.
- No global variables or functions may be defined.
- No global variables or functions can be accessed except the **`ADSAFE`** object.
- Use of the `[]` subscript operator is limited.
- These words cannot be used: **`apply arguments call callee caller constructor eval prototype unwatch valueOf watch`**
- Words starting with _ cannot be used.

Impact on the programming model

- Use of `[ ]` for subscripting is extremely limited. **`ADSAFE.get(`**name**`)`** and **`ADSAFE.set(`**name`,` value**`)`** must be used instead. This can be annoying.
- **`this`** cannot be used because it can be made to bind to the global object.
- JavaScript is still quite useable without **`this`**.

Constructor Recipe

1. Make an object.
   - Object literal
   - `new`
   - `Object.create`
   - call another constructor

Constructor Recipe

1. Make an object.
   - Object literal, `new`, `Object.create`, call another constructor
2. Define some variables and functions.
   - These become private members and private methods of the new object.

Constructor Recipe

1. Make an object.
   - Object literal, `new`, `Object.create`, call another constructor
2. Define some variables and functions.
   - These become private members.
3. Augment the object with privileged methods.

Constructor Recipe

1. Make an object.
   - Object literal, `new`, `Object.create`, call another constructor
2. Define some variables and functions.
   - These become private members.
3. Augment the object with privileged methods.
4. Return the object.

Step One

```
function myConstructor(x) {
    var that = otherMaker(x);
}
```

Step Two

```
function myConstructor(x) {
    var that = otherMaker(x);
    var secret = f(x);
}
```

Step Three

```
function myConstructor(x) {
    var that = otherMaker(x);
    var secret = f(x);
    that.priv = function () {
        ... secret x ...
    };
}
```

- The methods should use neither **`this`** nor **`that`**.

Step Four

```
function myConstructor(x) {
    var that = otherMaker(x);
    var secret = f(x);
    that.priv = function () {
        ... secret x ...
    };
    return that;
}
```

Objects made with this pattern do not need hardening.

Object tampering does not cause confusion.

ADsafe does not allow access to `Date` or `random`

This is to allow human evaluation of ad content with confidence that behavior will not change in the future. This is for ad quality and contractual compliance, not for security.

ADsafe DOM Interface

- Light weight.
- JQuery-like.
- Scope of queries is strictly limited to the contents of a the widget's `<div>`.
- Guest code cannot get direct access to any DOM node.

Widget Template

```
<div id="ADSAFEID_">
    HTML content goes here.
<script>
"use strict";
ADSAFE.id("ADSAFEID_");
</script>
<script src="approvedlibrary.js"></script>
<script>
"use strict";
ADSAFE.go("ADSAFEID_", function (dom, lib) {
    Application initialization goes here.
});
</script>
</div>
```

Library Template

```
"use strict";
ADSAFE.lib("libraryname", function () {
        Create that library object
    return that;
});
```

- The widget accesses the library object with **`lib.libraryname`**.

ADsafe validation is not destructive, so it can be performed at any and every point in the ad delivery pipeline.

It can even be done after consumer delivery to test compliance.

Multiple points of validation provide greater confidence that bad content will be blocked.

Dangers

- There may still be undiscovered weaknesses in ECMAScript and its many implementations.
- Those implementations are changing, introducing new weaknesses.
- The wrappers must be flawless.
- We are still subject to XSS attacks.

Misty

An experimental object capability language.

Goal: Correct every problem in JavaScript

Reasonable people will disagree
on what the problems actually are.

Misty Objectives

- Make the language easier for beginners.
- Make the language unastonishing and low cruft.
- Make the language an object capability language.

http://www.crockford.com/misty/

Syntax

- `:=` for assignment `=` for identity
- `+` for addition `&` for concatenation
- No semicolons. No blocks.

```
for i to length koda do
    if koda[i].id is null then
        raise 'misshapen'
    fi
od
```

No Global Object

- Each compilation unit is a function body, which gets the capability to return an object that exposes an interface that can be used by other compilation units.
- Compilation units share a vat, so communication is very fast. They can directly invoke methods. They can share object references.

Misty Object Hardening

- The **`fix`** operator produces an immutable reference. The original object is still mutable, but it cannot be changed with the fixed reference.

```
define frozen := fix my_object
```

- **`frozen`** and **`my_object`** are references to the same object, but the **`frozen`** reference is attenuated.

```
my_object.works := true

frozen.works := false # raise 'fix'
```

Fixed References

- A fixed reference cannot be used to modify an object.
- All references obtained with a fixed reference will be fixed.
- This avoids the ICE-9 problem.
- Function values cannot be obtained with a fixed reference. The functions can only be invoked.
- This prevents confusion.

Methods

- A method can obtain a reference to the object of interest with the `$` operator.
- A function that uses `$` can only be called as a method.
- The `$` operator can modify the object even if the object was fixed.

```
$.status := true     # succeeds
struct := $.struct   # struct is fixed
return $             # returns fixed
```

`$` could be viewed as a rights amplification, but it is only available to functions that are added to the object before it is fixed.

Simplicity

- Very simple operation. Just `fix` references before handing objects to strange code.
- Your own code is not inconvenienced by fixing.
- This level of simplicity is required for successful adoption.