Real time recompilation of running JavaScript

Peter van der Zee
JSConf.us 2013

Whois

Peter van der Zee
JavaScript developer
Netherlands

JS1K
ZeonJS
ZeParser
HeatFiler

qfox.nl
@kuvos

surfly.com
"Remote desktop in the browser"

Demonstration

Nothing to see here...

recap

Features

- Lazy evaluation
- Access to variables in a closure

Basically

Step by step

```
todo: func decl rewrite in the end

start("JSConf");
function start(world){
  setInterval(function(){
    console.log("hello "+world);
  }, 100);
}
```

Step 1: Rewrite func decls

```
function foo(){}
var foo = function(){};
```

Step 1: Rewrite func decls

```
function foo(){}
var foo = function(){};

var start = function(world){
  setInterval(function(){
    console.log("hello "+world);
  }, 100);
};
start("world");
```

Rewrite back to front

```
1:
var foo = function(){
  var bar = function(){
    var baz = function(){
      log("foo");
    };
  };
};
```

```
2:
var foo = function(){
  var bar = function(){
    var baz = $get(1,$compiler);
  };
};
```

```
3:
var foo = function(){
  var bar = $get(2, $compiler);
};
```

```
4:
var foo = $get(3, $compiler);
```

Step 2.1: Replace func expression

```
var start = function(world){
  setInterval(function(){
    console.log("hello "+world);
  }, 100);
};
start("world");
```

Step 2.1: Replace func expression

```
var start = function(world){
  setInterval($get(0, $compiler), 100);
};
start("world");
```

2.2 Rewrite func expression

```
var start = $get(1, $compiler);
start("world");
```

3: Run

Globals

- $data
- $get
- $getBody
- $compiler *

$data structure

```
$data = [{
    version: <number>,
    args: [<strings>],
    code: <string>
  },
  ...
  $get: ...
  $getBody: ...
];
```

$get

```
function $get(fid, compile){
  var func = null;
  var version = -1;
  return function wrapper(){
    if (!func || version !== $data[fid].version) {
      func = compile($getBody(fid));
      version = $data[fid].version;
    }
    func.apply(this, Array.prototype.slice.call(arguments, 0));
  };
};
```

$get

```
function $get(fid, compile){
  var func = null;
  var version = -1;
  return function wrapper(){
    if (!func || version !== $data[fid].version) {
      func = compile($getBody(fid));
      version = $data[fid].version;
    }
    func.apply(this, Array.prototype.slice.call(arguments, 0));
  };
};
```

$get

```
function $get(fid, compile){
  var func = null;
  var version = -1;
  return function wrapper(){
    if (!func || version !== $data[fid].version) {
      func = compile($getBody(fid));
      version = $data[fid].version;
    }
    func.apply(this, Array.prototype.slice.call(arguments, 0));
  };
};
```

$get

```
function $get(fid, compile){
  var func = null;
  var version = -1;
  return function wrapper(){
    if (!func || version !== $data[fid].version) {
      func = compile($getBody(fid));
      version = $data[fid].version;
    }
    func.apply(this, Array.prototype.slice.call(arguments, 0));
  };
};
```

$get

```
function $get(fid, compile){
  var func = null;
  var version = -1;
  return function wrapper(){
    if (!func || version !== $data[fid].version) {
      func = compile($getBody(fid));
      version = $data[fid].version;
    }
    func.apply(this, Array.prototype.slice.call(arguments, 0));
  };
};
```

$get

```
function $get(fid, compile){
  var func = null;
  var version = -1;
  return function wrapper(){
    if (!func || version !== $data[fid].version) {
      func = compile($getBody(fid));
      version = $data[fid].version;
    }
    func.apply(this, Array.prototype.slice.call(arguments, 0));
  };
};
```

$get

```
function $get(fid, compile){
  var func = null;
  var version = -1;
  return function wrapper(){
    if (!func || version !== $data[fid].version) {
      func = compile($getBody(fid));
      version = $data[fid].version;
    }
    func.apply(this, Array.prototype.slice.call(arguments, 0));
  };
};
```

$getBody

```
func = $compiler($getBody(fid));

var $compiler = function(){
  return eval(arguments[0]);
};
```

$getBody

```
func = $compiler($getBody(fid));

var $compiler = function(){
  return eval(arguments[0]);
};

func = eval($getBody(fid));
```

$getBody

```
var $getBody = function(fid){
  return (
    '(function('+
      ($data[fid].args||'')+
    '){'+
      'var $compiler = function(){'+
        'return eval(arguments[0]); '+
      '};'+
      $data[fid].code+
    '});'
  );
};
```

$getBody

```
var $getBody = function(fid){
  return (
    '(function('+
      ($data[fid].args||'')+
    '){'+
      'var $compiler = function(){'+
        'return eval(arguments[0]); '+
      '};'+
      $data[fid].code+
    '});'
  );
};
```

$getBody

```
var $getBody = function(fid){
  return (
    '(function('+
      ($data[fid].args||'')+
    '){'+
      'var $compiler = function(){'+
        'return eval(arguments[0]); '+
      '};'+
      $data[fid].code+
    '});'
  );
};
```

$getBody

```
var $getBody = function(fid){
  return (
    '(function('+
     ($data[fid].args||'')+
    '){'+
     'var $compiler = function(){'+
       'return eval(arguments[0]); '+
     '};'+
     $data[fid].code+
    '});'
  );
};
```

$getBody

```
var $getBody = function(fid){
  return (
    '(function('+
      ($data[fid].args||'')+
    '){'+
      'var $compiler = function(){'+
        'return eval(arguments[0]); '+
      '};'+
      $data[fid].code+
    '});'
  );
};
```

$getBody examples

```
// Blue is fid=0
var start = function(world){
  setInterval(function(){
    console.log("hello "+world);
  }, 100);
};
start("world");
```

$getBody(0)

```
(function(){
  var $compiler = function(){
    return eval(arguments[0]);
  };
  console.log("hello "+world);
});
```

getBody example #2

```
// blue is fid=1
var start = function(world){
  setInterval($get(0, $compiler), 100);
};
start("world");
```

$getBody(1)

```
(function(world){
  var $compiler = function(){
    return eval(arguments[0]);
  };
  setInterval($get(0, $compiler), 100);
});
```

Before

```
start("JSConf");
function start(world){
  setInterval(function(){
    console.log("hello "+world);
  }, 100);
}
```

After (runtime)

```
2| var start = function(world){
 |   var $compiler = function(){
 |     return eval(arguments[0]);
 |   };
3|   setInterval(function(){
 |     var $compiler = function(){
 |       return eval(arguments[0]);
 |     };
4|     console.log("hello "+world);
5|   }, 100);
 | };
1| start("JSConf");
```

Indirect direct eval

```
function f(){
  var foo = 5;
  return function(){ log(foo); };
}
var g = f();
g(); // logs 5
```

Indirect *direct* eval

```
var $body = '(function(){ log(foo); })';
function f(){
  var foo = 5;
  return eval($body);
}
var g = f();
g(); // logs 5
```

Indirect direct eval

```
var $body = '(function(){ log(foo); })';
function f(){
  var foo = 5;
  return $get(1, eval);
}
function $get(id, evaller){
  return evaller($body);
}
var g = f();
g(); // error: foo is undefined
```

Indirect ***direct*** eval

```
var $body = '(function(){ log(foo); })';
function f(){
  var foo = 5;
  return $get(1);
}
function $get(id){
  return eval($body);
}
var g = f();
g(); // error: foo is undefined
```

Indirect direct eval

- Direct eval has access to scope
- Indirect access only access to global
- Functions can access parent scopes

Indirect direct eval

- Direct eval has access to scope
- Indirect access only access to global
- Functions can access parent scopes

=> Direct eval wrapped in function

Indirect direct eval

```
var $body = '(function(){ log(foo); })';
function f(){
  var foo = 5;
  return $get(
    1,
    function(s){ return eval(s); })
}
function $get(id, evaller){
  return evaller($body);
}
var g = f();
g(); // logs 5
```

Named function expressions

```
var foo = function(){
  console.log("hello "+world);
};
```

Named function expressions

```
var foo = function(){
  var $compiler = function(){
    return eval(arguments[0]);
  };
  console.log("hello "+world);
};
```

Named function expressions

```
var foo = function name(){
  console.log("hello "+world);
};
```

Named function expressions

```
var foo = (function(){
  var name = function(){
    var $compiler = function(){
      return eval(arguments[0]);
    };
    console.log("hello "+world);
  };
  return name;
})();
```

Named function expressions

```
setTimeout(function repeat(){
  ...
  setTimeout(repeat, n);
}, n);
```

Open issues

- Inserting new functions
- Variable clashes
- Hard to explain

Code

http://github.com/qfox/recompiler

Other magic

- Parameter tracking
- Source-to-output tracking
- Output-to-source tracking
- Modify source with UI

Screencast: http://vimeo.com/53017149

Parameter tracking

- For DSL

- Wrap symbols
`` `new Rect(..) -> sym(new Rect(..))` ``

- Wrap params
`` `-> new Rect(param(...),..)` ``

Source-to-output tracking

Symbols

- have a unique source id
- have (unique) source range
- can generate multiple instances

Output-to-source tracking

- Special mouse events
- Find source range for element

Modify source with UI

1. Lookup source range for object
2. Carefully (!) modify existing expression
3. Recompile result