springone 2GX CHICAGO 2011

Modern JavaScript

Keith Donald – SpringSource

https://github.com/kdonald/modernjs

springsource A division of vmware

Agenda

- A look at Modern JavaScript Application Engineering
  - Core Language
  - Frameworks
- Q & A

- Goal of this talk
  - Help you with developing and structuring large JavaScript applications

Introduction

- Traditionally, JavaScript (JS) has gotten a bad wrap
  - a toy for script-kiddies
  - a cut & paste ghetto

- *"Hell is other people's JavaScript"*
  - Some guy on Twitter

Introduction (2)

- Yet, JS is not a toy, it is
  - An expressive, powerful language
  - Everywhere
  - Performant
- *JS is worth your time to master*

Core Language and Patterns

Modern JS

Key JavaScript Concepts

- Functions are first-class
- Everything is an object
- Closures
- Prototypes

Functions are first class

```
var sayHello = function(name) {
  return "hello " + name;
}

function logResult(fx, arg) {
  console.log(fx.call(null, arg));
}

logResult(sayHello, "keith");
["roy", "craig"].forEach(function(name) {
  logResult(sayHello, name);
});
```

Everything is an object

```
var obj = {};
console.log(obj instanceof Object);
console.log([] instanceof Object);
console.log((function() {}) instanceof Object);

obj.property = "foo";
obj.method = function() {
  console.log(this);
  return "bar";
}
console.log(obj.method());
```

Closures

```
var sayHelloMaker = function(name) {  return function() {
    return  "hello " + name;
  }
}
var helloKeith = sayHelloMaker(“keith”), helloRoy = sayHelloMaker(“roy”);
console.log(helloKeith());
console.log(helloRoy());

var obj = (function() {
  var priv = “value”;
  return {
    pub: function() {
      console.log(“Invoked a public function that can access private data ” + priv);
    }
  };
})();
console.log(obj.pub());
```

Prototypes

```
var obj = {}, obj2 = Object.create(Object.prototype); // equivalent

var user = (function() {
  function encode(password) { ... } // private
  return {
    authenticate: function(password) {
      return encode(password) === this.encodedPassword;
    },
    toString: function() { return this.username } // overrides Object.toString();
  };
})();
var keith = Object.create(user, {
  username: { value: “kdonald”, enumerable: true } ,
  encodedPassword: { value: “abfdfca234598b721” }
});
console.log(keith);
console.log(keith.authenticate(“melbourne”));
```

Notes about this operator

```
var fx = function() {
  console.log(this);
}
fx(); // this === window

var obj = {};
obj.property = fx;
obj.property(); // this === obj

jQuery(“#myDiv”).click(fx); // on click, this === myDiv

// key Function primitives - ‘obj’ becomes ‘this’ when ‘fx’ is invoked
fx.call(obj, arg1, arg2, ...);
fx.apply(obj, [args]);
```

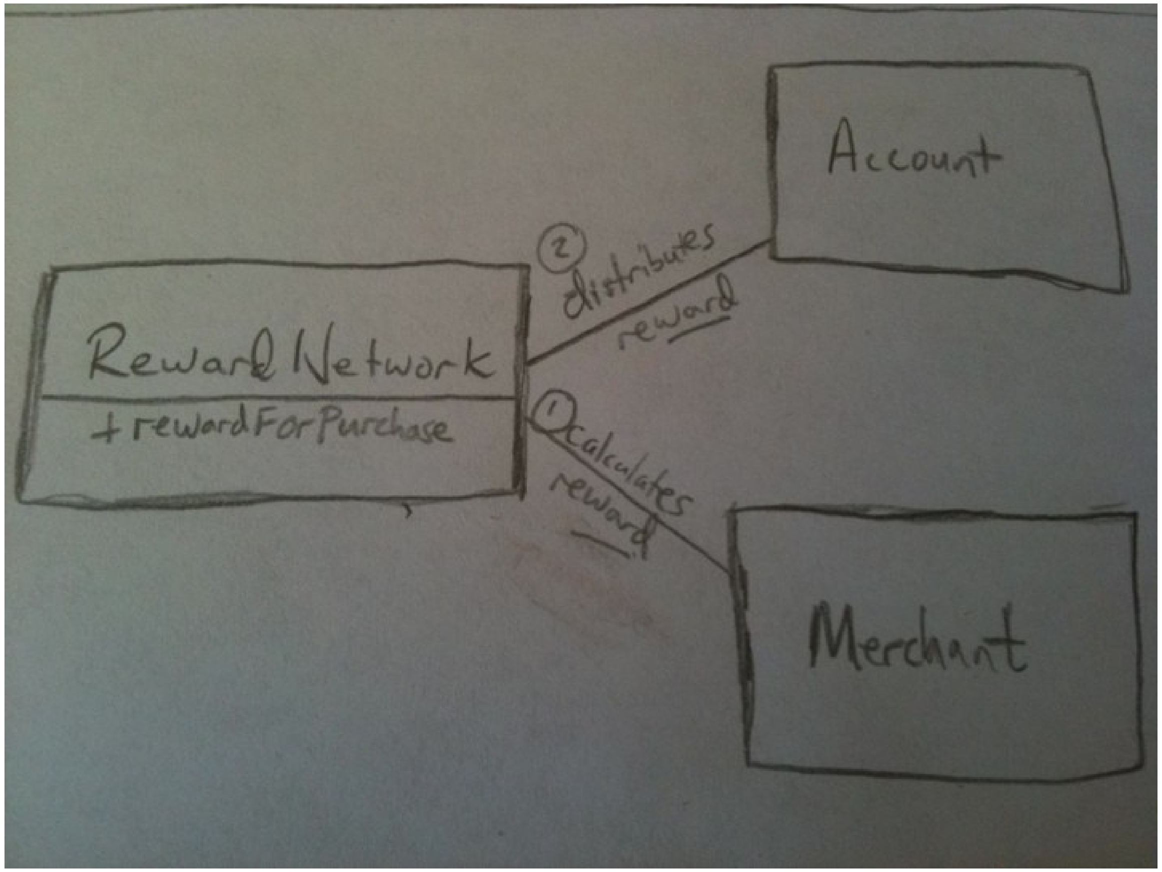
Notes about new operator

```
var User = function(name) {
  this.name = name;
}
User.prototype.awesome = true;

var keith = new User(“keith”);
var roy = new User(“roy”);

// new operator psudocode
function psudoNew(initializer, args) {
    var obj = Object.create(initializer.prototype);
    initializer.apply(obj, args);
    return obj;
}
var craig = psudoNew(User, ['craig']);
```

RewardNetwork: Reference App



RewardNetwork.java

```java
public class RewardNetwork {

  public RewardConfirmation rewardForPurchase(Purchase p) {
    Account account = accounts.withCreditCard(p.getCreditCard());
    Merchant merchant = merchants.withId(p.getMerchantId());
    Money reward = merchant.calculateReward(purchase, account);
    account.distribute(reward);
    accounts.saveChanges(account);
    return rewards.log(reward, account, purchase);
  }

}
```

Usage Example

```
public class Main {

  public static void main(String[] args) {
    RewardNetwork rewardNetwork = new RewardNetwork();
    rewardNetwork.setAccounts(accounts);
    rewardNetwork.setMerchants(merchants);
    Purchase p = new Purchase(new Money(“30.00”), “bizzarros”,
      new CreditCard(“1234123412341234”, “2/2013”));
    RewardConfirmation confirmation = rewardNetwork.rewardForPurchase(p);
  }
}
```

How to implement in JavaScript?

- With *idiomatic* JS
- Not JS that tries to emulate Java

rewardnetwork.js

```
define(["accounts", "merchants", "rewards"], function(accounts, merchants, rewards) {
  return {
    rewardForPurchase: function(purchase) {
      var account =  accounts.withCreditCard(purchase.creditCard);
      var merchant = merchants.withId(purchase.merchantId);
      var reward = merchant.calculateReward(purchase, account);
      account.distribute(reward);
      accounts.saveChanges(account);
      return rewards.log(reward, account, purchase);
    }
  };
});
```

main.js

```
require(["rewardnetwork"], function(rewardnetwork) {
  var confirmation = rewardnetwork.rewardForPurchase({
    amount: 30.00,
    merchantId: “bizzarros”,
    creditCard: { number: “1234123412341234”, expiration: “02/2013” }
  });
});
```

Demo

RewardNetwork JS

Summary

- Learn the key concepts
  - Function are first-class
  - Everything is an object
  - Closures
  - Prototypes

- Favor idiomatic JS

- Use modules to define units of JS code that
  - Have dependencies
  - Encapsulate private data and behavior
  - Export public behavior

Frameworks

Modern JS

General Categories

- DOM and Ajax
  - jQuery, Zepto
- Client Side Templating
  - Handlebars, Mustache, Eco
- MVC
  - Backbone, AgilityJS, SammyJS
- Module (or Script) Loaders
  - RequireJS
- Utilities
  - Underscore
- Full-Stack
  - Dojo, SproutCore

Demo: Frameworks

Modern JS

People and Resources

- Brendan Eich
- Douglas Crockford
- Alex Russell
- John Resig
- Jeremy Ashkenas
- Ryan Dahl
- Yehuda Katz
- Rebecca Murphy
- Brian Leroux

- JavaScript Weekly, Hacker News

Q&A