Rich UIs and Easy Ajax with Dojo and Zend Framework

Matthew Weier O'Phinney
Software Architect
Zend Framework

zend

The PHP Company

What we'll talk about

- **What the Zend Framework/Dojo integration offers**
- **Demonstration of some Dijits exposed by Zend Framework**
- **Benefits of using the integration**
- **Demonstration of building a Dojo-ized application in Zend Framework**
- **Creating custom Dojo builds for deployment**

Overview of Dojo Integration

- Shipping Dojo with Zend Framework as of 1.6.0
- Dojo view helper for managing Dojo environment
- Dijit-specific view helpers and form decorators and elements
- dojo.data response payloads with Zend_Dojo_Data
- JSON-RPC server implementation

How is Dojo shipped with ZF?

- **Lean-and-mean distribution: it's not**
  - Use the CDN
  - Download Dojo yourself
  - Create your own custom build to use

How is Dojo shipped with ZF?

- **"Kitchen Sink" distribution: full Dojo source build**
  - Contains Dojo source build (basically, full source minus a few artifacts)
  - All tools necessary for
    - building your own custom builds (Rhino)
    - testing (Doh!)

How is Dojo shipped with ZF?

- **Subversion**
  - svn:externals to latest release branch of Dojo
  - Full dojo source (including all artifacts)

dojo() View Helper

- Sets up the dojo environment
- Specify CDN or local install
- Specify dojo.require statements for including arbitrary Dojo modules
- Specify module paths for custom modules
- Specify layer (build) files
- Specify onLoad events
- And more!

Dijit Support

- **Support for (most) dijits (Dojo widgets)**
- **View helpers for rendering dijits**
  - Dijits are generated programmatically by default
  - You can specify Declarative style generation if desired
- **Form decorators for layout and form dijits**
  - Use layout dijit decorators typically with forms, sub forms, and display groups
- **Form elements for form dijits**
  - Map to the dijit view helpers

dojo.data Payloads

- dojo.data is a powerful data abstraction used across a variety of Dojo components
- Zend_Dojo_Data generates dojo.data compatible payloads
- Attach any traversable item (arrays, Iterators, etc.), specify an identifier field, and spit out as JSON

JSON-RPC Support

- JSON-RPC is a Remote Procedure Call protocol using JSON for the message serialization
- JSON Schema specification includes a Service Mapping Description (SMD) for defining available methods
- Zend_Json_Server implements a JSON-RPC server with SMD support
- Primary use case is for heavy client-side applications, where the client-side code is the View in MVC

You've said all this before in two other webinars...

> Can I have something concrete to look at, please?

Demonstration

Features seen

- **TabContainer**
  - Attached to the form as a decorator
- **Content Panes**
  - One per sub form, and a drop-in decorator to the form adding the Grid tab
- **Most form dijits**
  - via Zend_Dojo_Form
- **Remoting**
  - via ContentPane; Grid content pane pulls content dynamically
- **Grids**
  - consuming Zend_Dojo_Data as a dojo.data source

Code examples: form decorators

```
$this->setDecorators(array(
    'FormElements',
    'Grid',
    array('TabContainer', array(
        'id' => 'tabContainer',
        'style' => 'width: 100%; height: 500px;',
        'dijitParams' => array(
            'tabPosition' => 'top'
        ),
    )),
    'DijitForm',
));
```

Code example: sub form decorators

```
$textForm = new Zend_Dojo_Form_SubForm();
$textForm->setAttribs(array(
    'name'    => 'textboxtab',
    'legend' => 'Text Elements',
    'dijitParams' => array(
        'title' => 'Text Elements',
    ),
));
```

Code example: form element

```
$textForm->addElement(
        'TextBox',
        'textbox',
        array(
            'value'      => 'some text',
            'label'      => 'TextBox',
            'trim'       => true,
            'propercase' => true,
        )
    );
```

Code example: view helper

```
$html = $view->contentPane(
    'grid',
    'Grid demo is loading...',
    array(
        'title'       => 'Grid Demo',
        'preload'     => false,
        'href'        => '/dojo/grid/format/html',
        'parseOnLoad' => true,
    ),
    array(
        'class' => 'tab',
    )
);
```

That's nice. So what?

> What do I really gain?

What you gain:

- **Familiar PHP and ZF interface**
  - If you know how to use view helpers, you can use this
  - If you know how to create forms with Zend_Form, you can use this
- **In many cases, no need to learn Dojo immediately**
  - Takes care of things behind the scenes
  - Sprinkle in where it makes sense
- **Pretty interfaces**
- **Create consistent, beautiful interfaces, with little or no extra effort**

Show me

> I won't believe it until I see it.

Demonstration: Pastebin, without Dojo

Demonstration: Pastebin, with Dojo

Cool! How do I do it?

> What's the code behind it?

How we get there: bootstrap

```
Zend_Dojo::enableView($view);
$view->dojo()->setDjConfigOption('usePlainJson', true)
             ->addStylesheetModule('dijit.themes.tundra')
             ->addStylesheet('/js-src/dojox/grid/_grid/tundraGrid.css')
             ->setLocalPath('/js/dojo/dojo.js')
             ->addLayer('/js/paste/main.js') // custom code
             ->addJavascript('paste.main.init();')
             ->disable(); // enable only when necessary
```

- **Set djConfig options**
- **Add dijit themes and custom stylesheets**
- **Specify path to dojo, as well as any custom code**
- **Specify javascript to run at initialization**
- **Disable by default (to allow enabling only when necessary)**

How we get there: layout script

```
<?  $this->borderContainer()->captureStart('layout', array('design' => 'headline')) ?>
<?= $this->render('_headline.phtml') ?>
<?= $this->render('_mainPane.phtml') ?>
<?= $this->render('_footer.phtml') ?>
<?= $this->borderContainer()->captureEnd('layout') ?>
```

- **Create BorderContainer – master layout – and capture content to put in it**
- **Add several panes to it**

How we get there: layout script (cont)

```
<?= $this->contentPane(
        'headline',
        '<h1><a href="/">Pastebin</a></h1>',
        array('region' => 'top')
    ); ?>
```

- **Sample ContentPane – note that content can be provided to it directly.**

How we get there: pastebin form

```
class Paste_Form extends Zend_Dojo_Form
{
    public function init()
    {
        $this->addElement('FilteringSelect', 'type', array(
            'label'        => 'Language:',
            'multiOptions' => $this->_languages,
            'required'     => true,
        ));

        $this->addElement('ValidationTextBox', 'user', array(
            'label'        => 'Your name:',
            'regExp'       => '^[a-z][a-z0-9_-]+$',
            'validators'   => array(
                array('Regex', true, array('/^[a-z][a-z0-9_-]+$/i')),
            ),
        ));
```

- **Extends Zend_Dojo_Form: easiest method for using Dojo with Zend_Form**
- **Adding elements is the same – just new types for use with Dojo**

How we get there: pastebin form (cont)

```
$this->addElement('SimpleTextarea', 'code', array(
    'label'    => 'Code:',
    'required' => true,
    'class'    => 'codeTextarea',
));

$this->addElement('submitButton', 'save', array(
    'required'    => false,
    'ignore'      => true,
    'label'       => 'Save',
));
```

- **Elements have config options and accessors for setting Dijit parameters.**

How we get there: landing view

```
<?  $this->dojo()->enable() ?>
<?  $this->tabContainer()->captureStart('pastebin', array('class' => 'paste-tab')) ?>
<?= $this->render('paste/_about.phtml') ?>
<?= $this->contentPane('new-paste', '', array(
        'title' => 'New Paste',
        'class' => 'tab',
        'href'  => '/paste/new/format/ajax',
        'parseOnLoad' => true)) ?>
<?= $this->render('paste/_new-paste.phtml') ?>
<?= $this->contentPane('active', '', array(
        'title' => 'Active Pastes',
        'class' => 'tab',
        'href'  => '/paste/active/format/ajax',
        'parseOnLoad' => true)) ?>
<?= $this->tabContainer()->captureEnd('pastebin') ?>
```

- **Enable dojo when needed**
- **Mixture of content capturing and URL remoting for content panes**
- **All content captured within TabContainer**

How we get there: grid view

```
<?
Zend_Dojo_View_Helper_Dojo::setUseDeclarative();
$this->dojo()->addStylesheet('/js/dojox/grid/_grid/tundraGrid.css')
             ->requireModule('dojo.data.ItemFileReadStore')
             ->requireModule('dojox.grid.Grid');
?>
```

- **Grid setup – first, we tell Dojo to add some grid-specific CSS, and also what additional Dojo modules we need.**

How we get there: grid view (cont)

```
<span dojoType="dojo.data.ItemFileReadStore" jsId="activeStore"
    url="/paste/active-data/format/ajax"></span>
<table id="activePastes" dojoType="dojox.grid.Grid" store="activeStore"
    clientSort="true" query="{ id: '*' }">
    <script type="dojo/method" event="onSelected" args="inRowIndex">
var row = dijit.byId("activePastes").model.getRow(inRowIndex);
location.href = "/paste/display/id/" + row.id;
    </script>
    <thead>
        <tr>
            <th field="id" width="16em">ID</th>
            <th field="type">Type</th>
            <th field="user">User</th>
            <th field="summary">Summary</th>
            <th field="expires">Expires</th>
        </tr>
    </thead>
</table>
```

- **Grid markup – just HTML. This can be done programmatically, but Declarative style is often more expedient and fluent.**

How we get there: grid data generation

```
public function activeDataAction()
{
    $model = $this->getModel();
    $dojoData = new Zend_Dojo_Data('id', $model->fetchActive(), 'id');
    $this->view->data = $dojoData;
}

// View:
<?= $this->data->toJson() ?>
```

- **Grid data generation: pass our data to Zend_Dojo_Data... and that's it.**

Deployment Considerations

> Or, how to reduce the number of XHR requests your Dojo application generates.

What's going on under the hood?

- **Calling requireModule() generates dojo.require() statements.**
- **Each dojo.require() statement triggers one or more requests to the server to pull in the necessary Javascript.**
- **Design is very modular – you only use what you need.**
- **However, this means...**

Firebug output for the sample app

File View Help

Inspect Clear Profile

Console HTML CSS Script DOM Net YSlow Options

GET http://paste/js/dijit/layout/ContentPane.js 6ms	dojo.js (line 20)
GET http://paste/js/dijit/_Widget.js 37ms	dojo.js (line 20)
GET http://paste/js/dijit/_base.js 30ms	dojo.js (line 20)
GET http://paste/js/dijit/_base/focus.js 40ms	dojo.js (line 20)
GET http://paste/js/dijit/_base/manager.js 76ms	dojo.js (line 20)
GET http://paste/js/dijit/_base/place.js 45ms	dojo.js (line 20)
GET http://paste/js/dijit/_base/popup.js 52ms	dojo.js (line 20)
GET http://paste/js/dijit/_base/window.js 70ms	dojo.js (line 20)
GET http://paste/js/dijit/_base/scroll.js 43ms	dojo.js (line 20)
GET http://paste/js/dijit/_base/sniff.js 37ms	dojo.js (line 20)
GET http://paste/js/dijit/_base/bidi.js 50ms	dojo.js (line 20)
GET http://paste/js/dijit/_base/typematic.js 63ms	dojo.js (line 20)
GET http://paste/js/dijit/_base/wai.js 68ms	dojo.js (line 20)
GET http://paste/js/dijit/layout/_LayoutWidget.js 50ms	dojo.js (line 20)
GET http://paste/js/dijit/_Container.js 62ms	dojo.js (line 20)
GET http://paste/js/dojo/parser.js 44ms	dojo.js (line 20)
GET http://paste/js/dojo/date/stamp.js 69ms	dojo.js (line 20)
GET http://paste/js/dojo/string.js 48ms	dojo.js (line 20)
GET http://paste/js/dojo/i18n.js 50ms	dojo.js (line 20)
GET http://paste/js/dijit/nls/loading.js 50ms	dojo.js (line 20)
GET http://paste/js/dijit/layout/TabContainer.js 68ms	dojo.js (line 20)
GET http://paste/js/dijit/layout/StackContainer.js 53ms	dojo.js (line 20)
GET http://paste/js/dijit/_Templated.js 60ms	dojo.js (line 20)
GET http://paste/js/dijit/form/Button.js 52ms	dojo.js (line 20)
GET http://paste/js/dijit/form/_FormWidget.js 64ms	dojo.js (line 20)

>>>

Ouch!
(That's a lot of requests!)

What can be done?

- The solution is to create a custom build
- Custom builds pull all (specified) functionality into a single file
- All template strings are interred into the code
- Code is minified – whitespace removed, heuristics to condense variable names applied, etc.
- dojo.require() statements to modules compiled into the build become no-ops
- Trim the size of the scripts by many, many times, and reduce requests from many dozens to 1 or 2.

Example profile

```
dependencies = {
    layers: [
        {
            name: "../paste/paste.js",
            resourceName: "paste.layer",
            dependencies: [
                "dijit.layout.ContentPane",
                "dijit.layout.BorderContainer",
                "dijit.layout.TabContainer",
                "dijit.form.FilteringSelect",
                "dijit.form.ValidationTextBox",
                "dijit.form.SimpleTextarea",
                "dijit.form.Button",
                "dijit.form.Form",
                "dojo.data.ItemFileReadStore",
                "dojox.grid.Grid",
                "dojo.parser",
                "paste.main"
            ]
        },
    ],
    prefixes: [
        [ "dijit", "../dijit" ],
        [ "dojox", "../dojox" ],
        [ "paste", "../paste" ],
    ]
}
```

Creating the build

- **From the util/buildscripts/ directory, execute something like the following (assuming the profile script is in util/buildscripts/profiles/):**

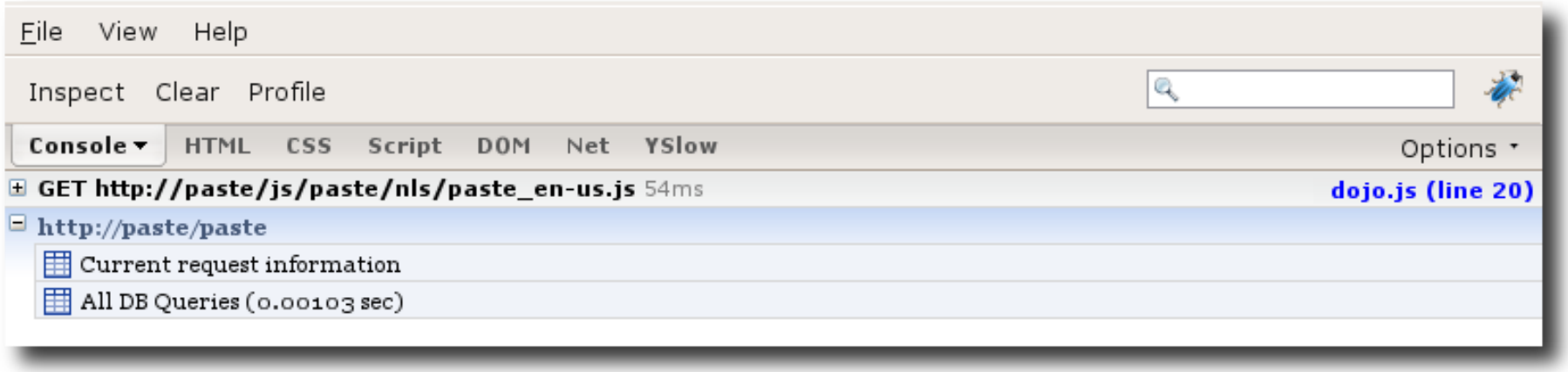
```
% ./build.sh profile="paste" action="release" version="1.1.1-paste" releaseName
="paste" loader="default" optimize="shrinksafe" layerOptimize="shrinksafe" copyT
ests="false"
release:  Using profile: profiles/paste.profile.js
release:  Using version number: 1.1.1-paste for the release.
release:  Deleting: ../../release/paste
release:  Copying: ../../dojo/../dijit to: ../../release/paste/dijit
release:  Copying: ../../dojo/../dojox to: ../../release/paste/dojox
release:  Copying: ../../dojo/../paste to: ../../release/paste/paste
release:  Copying: ../../dojo to: ../../release/paste/dojo
release:  Building dojo.js and layer files
release:  Interning strings for file: ../../release/paste/dojo/dojo.js
release:  Optimizing (shrinksafe) file: ../../release/paste/dojo/dojo.js
```

Other considerations

- **Builds are also called “layers”. After creating it, you need to add the layer to your application:**

```
Zend_Dojo::enableView($view);
$view->dojo()->setDjConfigOption('usePlainJson', true)
             ->addStylesheetModule('dijit.themes.tundra')
             ->addStylesheet('/js-src/dojox/grid/_grid/tundraGrid.css')
             ->setLocalPath('/js/dojo/dojo.js')
             // ->addLayer('/js/paste/main.js') // replacing with layer
             ->addLayer('/js/paste/paste.js')
             ->addJavascript('paste.main.init();')
             ->disable();
```

Firebug output from custom build



Conclusions

- **Develop with a source build – maximum flexibility**
- **Identify what dojo.require statements are necessary**
  - Look at your generated HTML for dojo.require statements
  - Run a screen scraper over your site to identify them
- **Create a profile for your application (or per-application on your site) from the information above**
- **Create a custom build for deployment using the profile(s)**
  - Full distro and SVN of ZF have all tools necessary for creating custom builds

Parting Words

> What was all of that, again?

Summary

- dojo() view helper to setup environment
- Dijit view helpers to create layouts and form elements
- Dijit form decorators and elements to create sophisticated forms
- Zend_Dojo_Data to create dojo.data payloads
- JSON-RPC to create "thick client" apps, where the View of your MVC is your client-side code
- Create custom builds for deployment to production

Where to get more information

- **ZF Zend_Dojo manual: http://framework.zend.com/manual/en/zend.dojo.html**
- **The Book of Dojo: http://dojotoolkit.org/book/dojo-book-1-0**
- **DojoCampus: http://dojocampus.org**

Thanks for listening!

> Now that you have no excuse to build beautiful, dynamic applications, what are you waiting for?