

リッチな Web アプリケーションをど〜じよ2

Dojo ウィジェットの拡張、再利用、データモデルとの連携

日本アイ・ビー・エム株式会社 ソフトウェア事業
コンサルティング・テクノロジー・エバンジェリスト
米持 幸寿

前提知識

- DynamicHTML
 - HTML
 - Cascading Style Sheet
 - JavaScript、イベント処理
- オブジェクト指向
 - JavaScriptのオブジェクト指向
 - クラス、継承、オーバーライド、インスタンス化
- HTTP
- Ajax
 - XHR
 - JSON

おさらい

WebページへのDojo組み込み

AOL CDNを利用

```
<style type="text/css">
  @import "http://o.aolcdn.com/dojo/1.0.0/dijit/themes/tundra/tundra.css";
  @import "http://o.aolcdn.com/dojo/1.0.0/dojo/resources/dojo.css"
</style>
<script type="text/javascript" src="http://o.aolcdn.com/dojo/1.0.0/dojo/dojo.xd.js"
  djConfig="parseOnLoad: true"></script>
<script type="text/javascript">
  dojo.require("dojo.parser");
</script>
```

サーバー上に配置

```
<style type="text/css">
  @import "/dojoroot/dijit/themes/tundra/tundra.css";
  @import "/dojoroot/dojo/resources/dojo.css"
</style>
<script type="text/javascript" src="/dojoroot/dojo/dojo.js"
  djConfig="parseOnLoad: true"></script>
<script type="text/javascript">
  dojo.require("dojo.parser");
</script>
```

ボタンを表示する最低限のコード

```
<html>
```

```
<head>
```

```
<style>
```

```
  @import "dijit/themes/soria/soria.css";
```

```
</style>
```

```
<script type="text/javascript" src="dojo/dojo.js"
```

```
  djConfig="parseOnLoad: true"></script>
```

```
<script type="text/javascript">
```

```
  dojo.require("dojo.parser");
```

```
  dojo.require("dijit.form.Button");
```

```
</script>
```

```
</head>
```

```
<body class="soria">
```

```
  <div id="button1" dojoType="dijit.form.Button">Dojoボタンです</div>
```

```
</body>
```

```
</html>
```

プレーンなJavaScriptのオブジェクト指向

- 変数も配列もメソッドさえもすべてオブジェクト
- すべてがポインター(参照)をもつ
- ポインターの関連づけでオブジェクトはできている
- クラス、継承、インスタンス化という概念ではない
 - prototypeによる共通化(クラス化)
 - 自動継承に課題

Dojoを拡張しよう

クラスの定義

```
dojo.declare(  
  "myModule.myClass",  
  myModule.myBaseClass,
```

クラス名

スーパークラス

```
{  
  myProp: "",  
  constructor: function(myProp){  
    this.myProp = myProp;  
    ...  
  },  
  myFunc: function(){  
    ...  
  },  
}
```

クラス変数

コンストラクター

メンバー関数

```
);
```

蛇足: Javaで書いたらこうだろう

```
package myModule;
```

```
public class myClass
extends myBaseClass
{
    public myProp= "";

    public myClass(???? myProp){
        this.myProp = myProp;
        ...
    }
    public ??? myFunc() {
        ...
    }
}
```

とりあえず1ファイルでやってみた

```
:  
<script type="text/javascript" src="../../dojo-release-1.1.1/dojo/dojo.js"  
  djConfig="parseOnLoad: true"></script>  
<script type="text/javascript">  
  dojo.require("dojo.parser");  
  dojo.require("dijit.form.Button");
```

```
  dojo.declare("yone.MyButton", null,  
    {  
    }  
  );
```

```
  dojo.addOnLoad(function(){  
    dojo.connect(dojo.byId("button1"), "onclick", function(){
```

```
      var myButton = new yone.MyButton();
```

```
      alert(myButton);
```

```
    });
```

```
  });  
</script>  
</head>  
<body>  
<div dojoType="dijit.form.Button" id="button1">クラス生成</div>  
</body>  
:
```

✓
クラス生成

[JavaScript アプリケーション]



[object Object]

OK

sample01.html

継承してみた

```
:  
<script type="text/javascript" src="../../dojo-release-1.1.1/dojo/dojo.js"  
  djConfig="parseOnLoad: true"></script>  
<script type="text/javascript">  
  dojo.require("dojo.parser");  
  dojo.require("dijit.form.Button");
```

```
  dojo.declare(  
    "yone.MyButton",  
    dijit.form.Button,  
    {  
      ← 継承してみた  
    }  
  );
```

```
</script>  
</head>  
<body>
```

```
<div dojoType="yone.MyButton" id="button1">米持のボタン</div>
```

```
</body>  
:
```

継承してみた

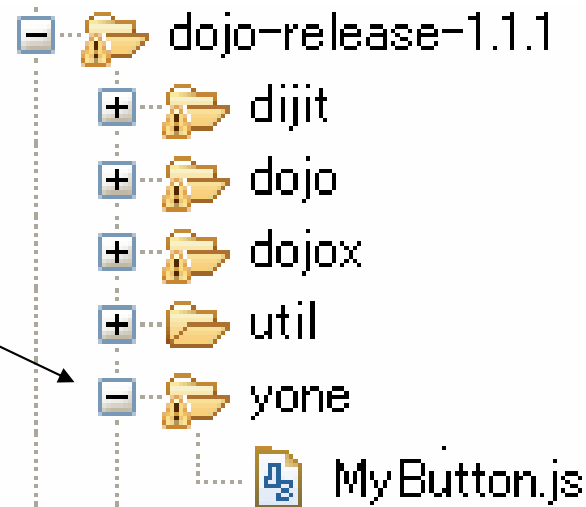
dojoType に
指定してみた

sample02.html

クラスを別のファイルに分けよう

- ファイルの検索方法
 - クラス名を「.」で分解
 - 最後のワードがファイル名
 - それより前はフォルダ名
 - dojoのルート (dojoフォルダの上) から探す

yone.MyButton なら
この位置にあるはず



モジュールの定義

- **dojo.provide(moduleName)**
 - モジュールの登録
 - パッケージ (オブジェクト階層) の生成

myModules/myModule1.js:

```
dojo.provide("myModules.myModule1");
```

```
myModules.myModule1.myFunc = function(){
```

```
    ...
```

```
};
```

モジュールの読み込み

- **dojo.registerModulePath(moduleName, path)**

– モジュールのルートが dojo ディレクトリと同じ場所であれば不要。

dojo ツールキットと
自作アセットを分けておきたい場合

- **dojo.require(moduleName)**

```
<script type="text/javascript">  
  dojo.require("dojo.parser");  
  dojo.require("dijit.form.Button");  
  ...
```

2ファイルにわけてみた(dojoの下に配置)

```
dojo.provide("yone.MyButton");  
dojo.require("dijit.form.Button");  
dojo.declare(  
    "yone.MyButton",  
    dijit.form.Button,  
    {  
    }  
);
```

ツールキットの直下に
同じ名前のフォルダーとjsファイルを置く



分けたファイルを呼び出してみた

```
<script type="text/javascript" src="../../dojo-release-1.1.1/dojo/dojo.js"
  djConfig="parseOnLoad: true"></script>
```

```
<script type="text/javascript">
  dojo.require("dojo.parser");
```

```
dojo.require("yone.MyButton");
```

```
</script>
```

```
</head>
```

```
<body>
```

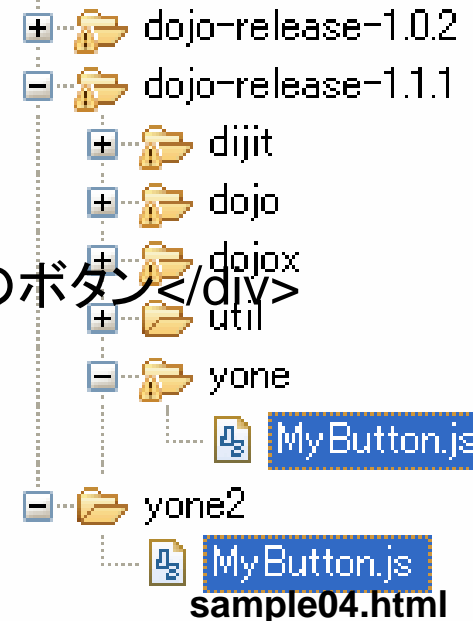
```
<div dojoType="yone.MyButton" id="button1">米持  
  のボタン</div>
```

```
</body>
```

```
</html>
```

dojoの外に配置してみた

```
:  
<script type="text/javascript" src="../../dojo-release-1.1.1/dojo/dojo.js"  
  djConfig="parseOnLoad: true"></script>  
<script type="text/javascript">  
  dojo.require("dojo.parser");  
  dojo.registerModulePath("yone2", "../../yone2");  
  dojo.require("yone2.MyButton");  
</script>  
</head>  
<body>  
<div dojoType="yone2.MyButton" id="button1">米持のボタン</div>  
:
```



djConfig に指定してもよい

```
djConfig=  
"parseOnLoad: true,  
  modulePaths:{yone2: '../..../yone2'}  
"
```

注：djConfig に指定される文字列は、{} で囲まれて、
パースされてJavaScript オブジェクトに変換される

HTMLで定義したレイアウトの再利用

- dijit._Templated
 - テンプレート ウィジェットのベース クラス
 - templatePath (HTML ファイル)、templateString (文字列)、または templateNode でテンプレートを指定
 - dojoAttachPoint でノードをウィジェットのプロパティに格納
 - dojoAttachEvent でノードのイベントをウィジェットのイベントにマップ

```
dojo.declare("myModule.myWidget",  
[dijit._Widget, dijit._Templated], {  
  templateString: "<div>Hello," +  
    "<span dojoAttachPoint='nameNode'>`${myName}</span>" +  
    "</div>",  
  myName: "",  
  ...  
});
```

多重継承

テンプレートウィジェットを作ってみた

```
dojo.provide("yone.HelloWorld");
dojo.require("dijit._Widget");
dojo.require("dijit._Templated");
dojo.declare(
  "yone.HelloWorld",
  [dijit._Widget, dijit._Templated],
  {
    templateString:
      "<table
order='1'><tr><td>Hello</td><td>World</td></tr></table>"
  }
);
```

yone/HelloWorld.js

```
dojo.require("yone.HelloWorld");

<div dojoType="yone.HelloWorld">子テキスト</div>
```

sample06.html

HTMLを分けてみた

```
dojo.provide("yone.HelloWorld2");
dojo.require("dijit._Widget");
dojo.require("dijit._Templated");
dojo.declare(
  "yone.HelloWorld2",
  [dijit._Widget,dijit._Templated],
  {
    templatePath:
      dojo.moduleUrl("yone", "templates/HelloWorld2.html")
  }
);
```

```
<html>
<head>
<title>Insert title here</title>
</head>
<body>
<table border="1">
<tr>
<td>Hello</td>
<td>World</td>
</tr>
</table>
</body>
</html>
```

yone/templates/HelloWorld2.html

yone/HelloWorld2.js

```
<div dojoType="yone.HelloWorld2">子テキスト</div>
```

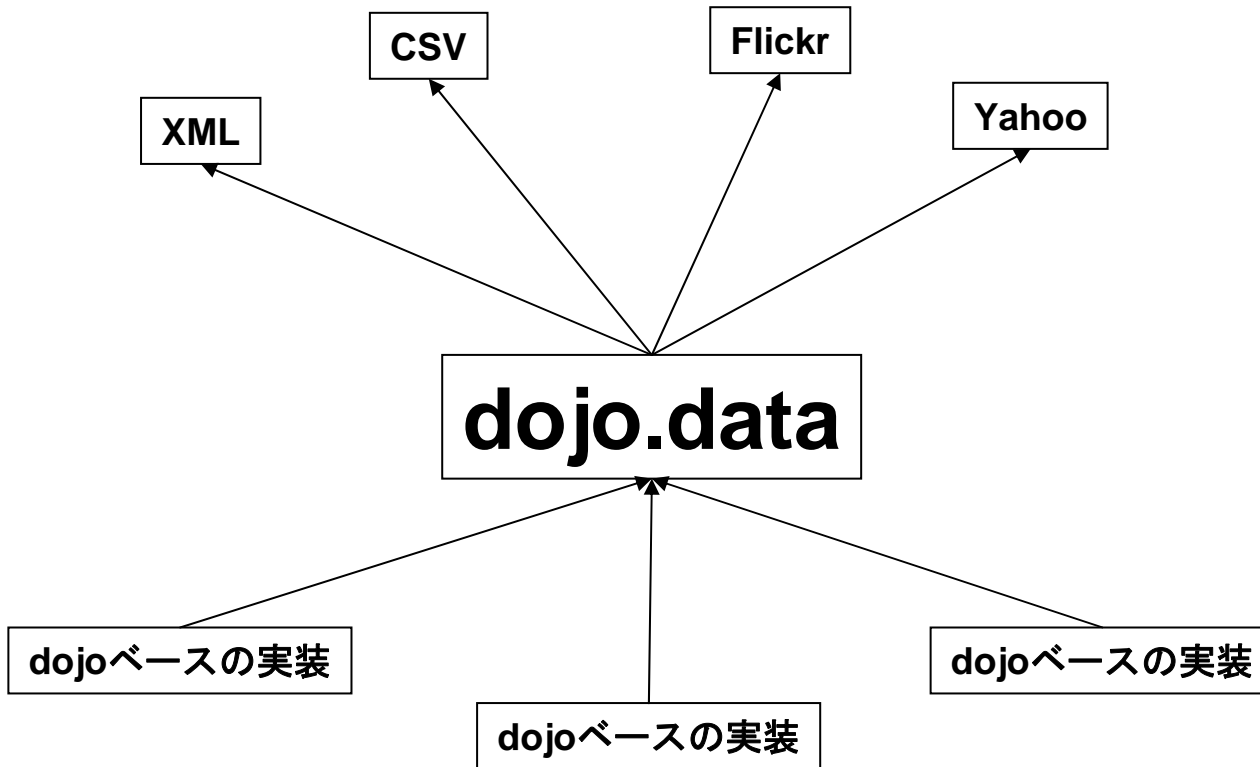
sample07.html

dojo.data

dojo.data とは？

- Dojoツールキットの統一データアクセス層
- item と attribute ですべてのデータを表現する
- 様々なデータフォーマットをdojo.dataで表現する
 - JSON、XML、CSV
- 様々なサービスをdojo.dataで呼び出せるようにする
 - Yahoo、Google、Twitter、Picasa、Flickr

データのアクセスを共通化



用語

用語	データベースで 言えば	説明
datastore	cursor	このオブジェクトを通してdata source からデータを読む
data source	database	データが保存されている場所
item	row	値を持つattributeを複数もったデータ項目
attribute	column	データ項目のフィールドやプロパティ
value		attributeの内容
reference		別のアイテムを指し示す値
identity	primary key	一つのデータストア上でアイテムを識別するための識別子
query	WHERE句	データストアからサブセットを取り出すしくみ
dojo.data API	JDBC / ODBC	dojo.require("dojo.data.api") データAPI
request	SELECT	クエリ、ソート、上下の範囲、コールバック

book of dojo より

dojo.data API

- `dojo.data.api.Read`
 - 読み出し、検索、ソート、フィルターなど
- `dojo.data.api.Write`
 - create / delete / update
 - 必ずしも更新できるとは限らない
- `dojo.data.api.Identity`
 - 識別子による抽出
 - かならずしも識別子が使えらるとは限らない
- `dojo.data.api.Notification`
 - create, delete, and update のような変更が加わったときに呼び出されるイベント

ItemFileReadStore

- dojoコアが提供
- シンプルなデータストア実装
 - JSON
 - Read / Identity

dijit への接続

- `<div>` を使ってストアを生成
 - `jsId` で名前を設定
- `<div dojoType="dijit.XX" store jsId="xxx">`
 - を指定する


こんなJSONを

```
{label: 'name',
items:[
  {name:'Windows', children: [
    {name:'System32', children: [
      {name:'drivers'},
      {name:'config'}
    ]
  },
  {name:'System32', children:[
    {name:'Table.dat'}
  ]
}]
}
```

directories.json

dijit.Tree へ接続してみた

```
<script type="text/javascript">
  dojo.require("dijit.Tree");
  dojo.require("dojo.data.ItemFileReadStore");
</script>
:
<body class='soria'>
  <div dojoType="dojo.data.ItemFileReadStore"
    jsld="dirStore"
    url="directories.json"></div>
  <div dojoType="dijit.Tree" store="dirStore" label="C:¥">
  </div>
</body>
```



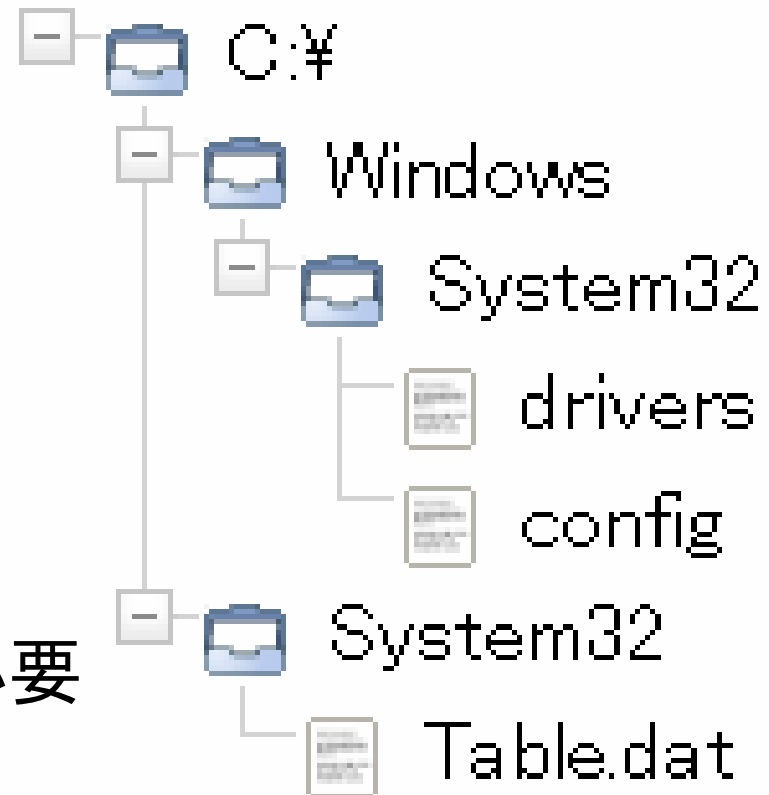
The diagram illustrates the data flow. A light blue box on the right represents a 'JSONファイル' (JSON file). An arrow points from this box to the 'url="directories.json"' attribute of the 'dojo.data.ItemFileReadStore' widget in the HTML code. Another arrow points from the 'jsld="dirStore"' attribute of the same widget to the 'store="dirStore"' attribute of the 'dijit.Tree' widget, showing how the tree widget is connected to the data store.

JSONファイル

sample08.html

こうなった

- 注:「System32」は二個ある
- identifier に指定することはできない
- identifier を使いたい場合は、絶対パスをすべてのオブジェクトに持たせるなどの工夫が必要



identifier を追加

```
{ identifier: 'absolute', ←
  label: 'name',
  items:[
    {name:'Windows', absolute: 'Windows',
      children: [
        {name:'System32', absolute: 'Windows\System32',
          children: [
            {name:'drivers', absolute: 'Windows\System32\drivers'},
            {name:'config', absolute: 'Windows\System32\config'}
          ]
        }
      ]
    },
    {name:'System32', absolute: 'System32',
      children:[
        {name:'Table.dat', absolute: 'System32\Table.dat'}
      ]
    }
  ]
}
```

directories2.json

JavaScript 上で特定アイテムにアクセス

```
var dirStore =
```

```
new dojo.data.ItemFileReadStore(  
    {url: "directories2.json"}  
);
```

```
dojo.connect(dojo.byId("button"), "onclick", function(evt){
```

```
dirStore.fetchItemByIdentity({  
    identity: dojo.byId("key").value,  
    onItem: function(item){  
        alert(item.name);  
    }  
});
```

```
});
```

```
キーワード<div dojoType="dijit.form.TextBox" id="key"></div>  
<div dojoType="dijit.form.Button" id="button">検索</div>
```

キーワード

sample09.html

複数データの取得

```
dojo.connect(dojo.byId("button"), "onclick", function(evt){
  dirStore.fetch({
    queryOptions: {ignoreCase: true, deep: true},
    query: {"*" + dojo.byId("key").value + "*"},
    onComplete: function(items){
      dojo.forEach(items, function(item){
        alert(item.absolute);
      });
    },
    onError: function(){ alert(error); }
  });
});
```

正規表現



※ query を付けなければ全データ

Lazyロード

- `store.loadItem(`
 `{item: value, onItem: *function});`

ページネーション(ページ割)

- `request = store.fetch(`
 `{start: n, count: pagesize, xxx});`

ソート

- `request = store.fetch(`
 `{sort: sortKeys, xxx`

様々なストア

<code>dojo.data.ItemFileReadStore</code>	read-only store for JSON data
<code>dojo.data.ItemFileWriteStore</code>	read/write store for JSON data
<code>dojox.data.CsvStore</code>	read-only store for comma-separated variable (CSV) formatted data
<code>dojox.data.OpmlStore</code>	read-only store for Outline Processor Markup Language (OPML)
<code>dojox.data.HtmlTableStore</code>	read-only store for data kept in HTML-formatted tables
<code>dojox.data.XmlStore</code>	read/write store for basic XML data
<code>dojox.data.FlickrStore</code>	read store for queries on flickr.com, and a good example data store for web services
<code>dojox.data.FlickrRestStore</code>	read store for queries on flickr.com, and a good example data store for web services. More advanced version of FlickrStore
<code>dojox.data.QueryReadStore</code>	like ItemFileReadStore, read-only store for JSON data, but queries servers on each request
<code>dojox.data.AtomReadStore</code>	read store for Atom XML documents.

参考URL

- ソフトウェア・テクノロジーWiki

<http://www.ibm.com/developerworks/wikis/display/swtechnologyj>

- dojotoolkit.org

<http://dojotoolkit.org/>

– book of dojo

- <http://dojotoolkit.org/book/dojo-book-1-0>

- 米持幸寿のブログ

<http://www.ibm.com/developerworks/blogs/page/pandrbbox>