INE5646 Programação para Web

- Tópico :

  Processamento no Cliente - Modularidade

(estes slides fazem parte do material didático da disciplina INE5646 Programação para Web)

Sumário

- Desafios da era Web 2.0
- Modularidade com JavaScript
- Modularidade com RequireJS
- RequireJS: exemplo

Desafios da era Web 2.0

- Aplicações web 2.0 possuem a camada 1 complexa. Implicações:
  - Interface com o usuário sofisticada implica em muitas linhas de código JavaScript.
  - Frequentemente utiliza-se muitas bibliotecas JavaScript desenvolvidas por terceiros.
  - **Reusabilidade de código** é inevitável para melhorar a produtividade (como sempre foi na indústria de software) => criação das próprias bibliotecas.

Modularidade com JavaScript

- Um programa JavaScript é formado, em última análise, por um conjunto de variáveis e funções.
- Todas as variáveis e funções, de todos os arquivos “.js” que fazem parte da camada 1, são definidos no mesmo espaço de nomes (namespace).
- Nada impede que dois arquivos “.js” definam uma variável e/ou função com o mesmo nome.

Modularidade com JavaScript

- Em muitas linguagens de programação, o problema de conflito de nomes é resolvido com a noção de **módulo** ou de **pacote**.
- Como este conceito (módulo/pacote) não existe em JavaScript, ele deve ser "implementado" seguindo-se uma metodologia de trabalho para tentar evitar o problema.
- Há duas abordagens para colocar esta metodologia em prática: solução "*ad hoc*" ou solução usando alguma biblioteca para este fim.

Modularidade com JavaScript

- Modularidade via solução ad hoc:
  - Todas as variáveis e funções do módulo são definidos dentro de um único objeto cujo nome é será o nome do módulo.
  - Asssim, quando o script é carregado adiciona-se um único nome ao *namespace* (página html).
  - Problemas desta solução:
    - Não é possível garantir que o nome seja realmente único pois o nome do objeto é definido sem se saber quais outras bibliotecas serão usadas na aplicação.
    - Módulos frequentemente dependem de outros módulos e não há como registrar, a não ser via comentários, estas dependências.

Modularidade com JavaScript

- Modularidade via biblioteca:
  - O conceito de módulo é implementado (“simulado”) em JavaScript.
  - Esta abordagem define exatamente como (usando funções/objetos da biblioteca) o desenvolvedor cria e utiliza módulos.
  - Bibliotecas exemplo:
    - RequireJS (http://requirejs.org/)
    - CommonsJS (http://www.commonjs.org/)

Modularidade com JavaScript

- Modularidade via biblioteca:
  - O conceito de módulo é implementado (“simulado”) em JavaScript.
  - Esta abordagem define exatamente como (usando funções/objetos da biblioteca) o desenvolvedor cria e utiliza módulos.
  - Bibliotecas exemplo:
    - RequireJS (http://requirejs.org/)
    - CommonsJS (http://www.commonjs.org/)

Modularidade com RequireJS

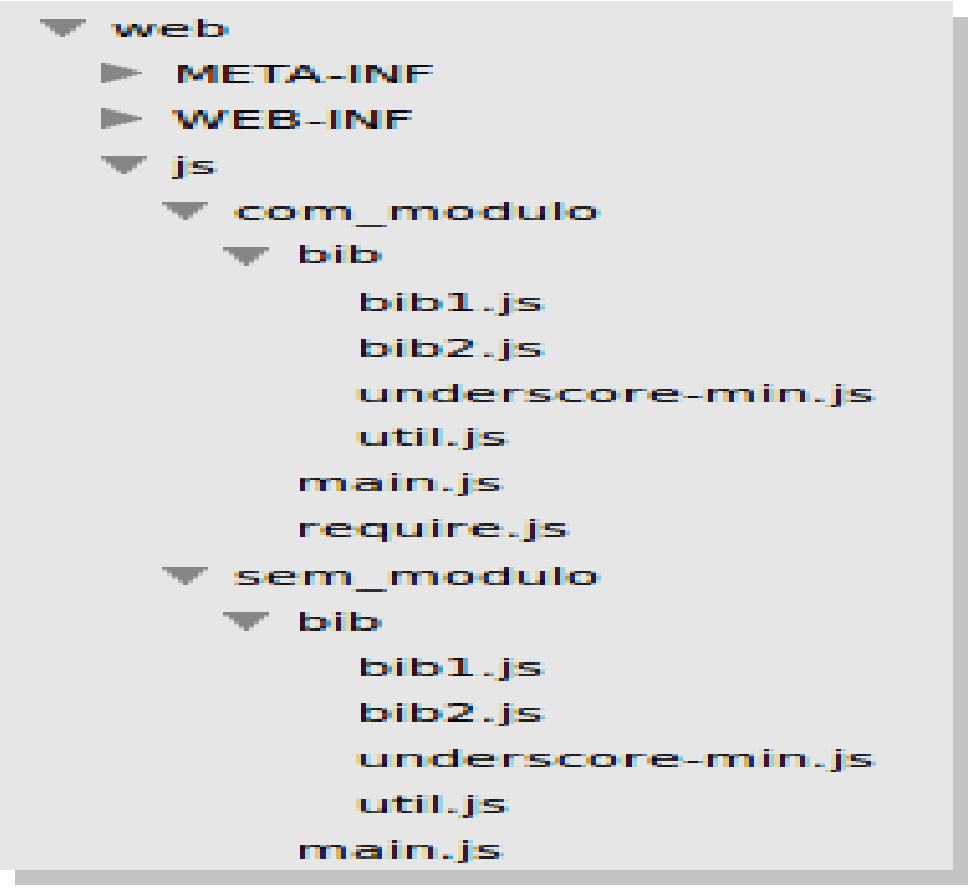
- RequireJS traz as seguintes vantagens ao desenvolvedor de aplicações:
  - Induz o desenvolvimento modular de aplicações.
  - Gerencia automaticamente as dependências entre os módulos.
  - Carregamento assíncrono de arquivos JavaScript (a página não fica “congelada” esperando que os arquivos .js sejam baixados do servidor).
  - Pode ser usada também no lado servidor, embora tenha sido criada para ser usada no browser.

Modularidade com RequireJS

- Para entender como usar e quais os benefícios de RequireJS, é apresentado a seguir uma aplicação formada por duas páginas equivalentes:
  - **semModulo.html**: utiliza a abordagem tradicional, sem módulo, para referenciar e utilizar códigos escritos em JavaScript
  - **comModulo.html**: é uma refatoração da página anterior incorporando o conceito de módulo conforme definido pela biblioteca RequireJS.

Modularidade com RequireJS

- Ambas as páginas utilizam arquivos JavaScript organizados como mostrado na figura abaixo:



RequireJS: exemplo sem módulo

- Arquivo semModulo.html:

```
1  <!DOCTYPE html>
2  <html>
3    <head>
4      <title></title>
5      <meta http-equiv="Content-Type"
6            content="text/html; charset=UTF-8">
7    </head>
8    <body>
9      <div id="dados">
10       <h2>Resultado SEM Modularização</h2>
11     </div>
12     <script src="js/sem_modulo/bib/underscore-min.js"></script>
13     <script src="js/sem_modulo/bib/util.js"></script>
14     <script src="js/sem_modulo/bib/bib1.js"></script>
15     <script src="js/sem_modulo/bib/bib2.js"></script>
16     <script src="js/sem_modulo/main.js"></script>
17   </body>
18 </html>
```

RequireJS: exemplo sem módulo

- Arquivo semModulo.html:
  - Linhas 12 a 16 definem os 5 “módulos” que fazem parte da aplicação.
  - A ordem apresentada é essencial pois alguns arquivos dependem dos outros. Alterar a ordem pode deixar a aplicação sem funcionar ou então apresentar erros de lógica muito difíceis de serem detectados.

```
12      <script src="js/sem_modulo/bib/underscore-min.js"></script>
13      <script src="js/sem_modulo/bib/util.js"></script>
14      <script src="js/sem_modulo/bib/bib1.js"></script>
15      <script src="js/sem_modulo/bib/bib2.js"></script>
16      <script src="js/sem_modulo/main.js"></script>
```

RequireJS: exemplo sem módulo

- Arquivo underscore-min.js:
  - Biblioteca desenvolvida por terceiros (ver site http://underscorejs.org/) que disponibiliza uma série de funções utilitárias.
  - Esta biblioteca é usada em um “módulo” da aplicação exemplo.

```
12      <script src="js/sem_modulo/bib/underscore-min.js"></script>
13      <script src="js/sem_modulo/bib/util.js"></script>
14      <script src="js/sem_modulo/bib/bib1.js"></script>
15      <script src="js/sem_modulo/bib/bib2.js"></script>
16      <script src="js/sem_modulo/main.js"></script>
```

RequireJS: exemplo sem módulo

- Arquivo util.js:
  - Define a função adicioneUm.
  - Sua implementação usa a bilioteca underscore-min.js (linha 5).

```
1  // util.js - sem módulo
2  // utiliza o método _.isNumber definido em underscore-min.js
3
4  function adicioneUm(n) {
5    if (_.isNumber(n))
6      return n + 1;
7    else
8      return 0;
9  }
```

RequireJS: exemplo sem módulo

- Arquivo bib1.js:
  - Define a função **f**.
  - Sua implementação usa a função adicioneUm definida na bilioteca util.js (linha 5).

```
1 // bib1.js - sem módulo
2 // utiliza a função adicioneUm definida em util.js
3
4 function f(v) {
5     return adicioneUm(v) * 2;
6 }
```

RequireJS: exemplo sem módulo

- Arquivo bib2.js:
  - Define a função **f**.

```
1 // bib2.js - sem módulo
2
3 function f(a) {
4     return a * 10;
5 }
```

RequireJS: exemplo sem módulo

- **Conflito**: Arquivos bib1.js e bib2.js definem a mesma função **f**!
  - Este “erro” nunca é detectado.
  - Qual definição de f é a que vale?

```
1 // bib1.js - sem módulo
2 // utiliza a função adicioneUm definida em util.js
3
4 function f(v) {
5     return adicioneUm(v) * 2;
6 }
```

```
1 // bib2.js - sem módulo
2
3 function f(a) {
4     return a * 10;
5 }
```

RequireJS: exemplo sem módulo

- **Conflito**: Arquivos bib1.js e bib2.js definem a mesma função **f** e **a versão que fica valendo depende da ordem de inclusão dos arquivos**:
  - Como bib2.js é incluído (linha 15) depois de bib1.js (linha 14) vale o definido em bib2.
  - **TERROR! A função f está sendo sileciosamente redefinida**.

```
12      <script src="js/sem_modulo/bib/underscore-min.js"></script>
13      <script src="js/sem_modulo/bib/util.js"></script>
14      <script src="js/sem_modulo/bib/bib1.js"></script>
15      <script src="js/sem_modulo/bib/bib2.js"></script>
16      <script src="js/sem_modulo/main.js"></script>
```

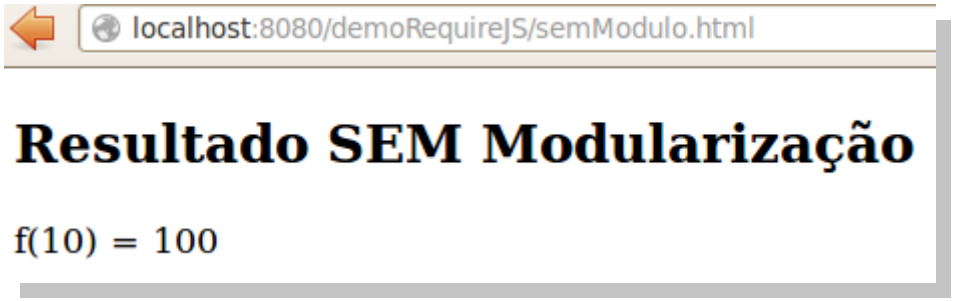
RequireJS: exemplo sem módulo

- Arquivo main.js:
  - Ao invocarmos a função f (linha 12) será usada a definição contida em bib2.js.

```
1  // main.js - sem módulo
2
3  // problema: a função 'f' está definida em duas bibliotecas (bib1.js e bib2.js).
4  // Assim, o valor de 'f(10)' depende da ordem em que os arquivos .js foram
5  // incluídos na página 'semModulo.html'.
6  // Como bib2.js foi incluído depois de bib1.js, é a sua definição que vale,
7  // ou seja, f(10) = 100 e não 22 como definido em bib1.js.
8  // Experimente alterar a ordem das duas bibliotecas no arquivo semModulo.html.
9
10 var d = document.getElementById("dados");
11 d.innerHTML += "f(10) = ";
12 d.innerHTML += f(10);
```

RequireJS: exemplo sem módulo

- Arquivo main.js:
  - O valor de f(10) é, portanto, 100.



RequireJS: exemplo **com** módulo

- O exemplo a seguir é o mesmo apresentado na página semModulo.html, só que agora **refatorado** usando-se a biblioteca RequireJS.
- **Com a refatoração torna-se possível usar as duas versões da função f ao mesmo tempo**!

RequireJS: exemplo com módulo

- Arquivo comModulo.html:
  - Inclusão de um único arquivo (require.js) nas linhas 7 e 8. Os demais arquivos serão incluídos via RequireJS.

```
1  <!DOCTYPE html>
2  <html>
3      <head>
4          <title></title>
5          <meta http-equiv="Content-Type"
6                content="text/html; charset=UTF-8">
7          <script data-main="js/com_modulo/main"
8                  src="js/com_modulo/require.js"></script>
9      </head>
10     <body>
11         <div id="dados">
12             <h2>Resultado COM Modularização</h2>
13         </div>
14     </body>
15 </html>
```

RequireJS: exemplo com módulo

- Arquivo util.js:
  - A **função define** define um módulo.

```
1  // util.js - com módulo
2
3  // Este módulo utiliza o módulo 'underscore' e exporta um objeto
4  // que contém a função 'adicioneUm'.
5
6  define(['underscore'], function(_) {
7
8    return {
9      adicioneUm: function(n) {
10       if (_.isNumber(n))
11         return n + 1;
12       else
13         return 0;
14     }
15   };
16 });
```

RequireJS: exemplo com módulo

- Arquivo util.js:
  - **O primeiro parâmetro da função define** define quais módulos são usados na implementação do módulo que está sendo definido. No caso, utiliza apenas o módulo underscore.

```
1  // util.js - com módulo
2
3  // Este módulo utiliza o módulo 'underscore' e exporta um objeto
4  // que contém a função 'adicioneUm'.
5
6  define(['underscore'], function(_) {
7
8    return {
9      adicioneUm: function(n) {
10       if (_.isNumber(n))
11         return n + 1;
12       else
13         return 0;
14     }
15   };
16 });
```

RequireJS: exemplo com módulo

- Arquivo util.js:
  - **O segundo parâmetro da função define** corresponde à implementação do módulo. O parâmetro "_" da função é o nome do objeto exportado pelo módulo underscore.

```
1  // util.js - com módulo
2
3  // Este módulo utiliza o módulo 'underscore' e exporta um objeto
4  // que contém a função 'adicioneUm'.
5
6  define(['underscore'], function(_) {
7
8    return {
9      adicioneUm: function(n) {
10       if (_.isNumber(n))
11         return n + 1;
12       else
13         return 0;
14     }
15   };
16 });
```

RequireJS: exemplo com módulo

- Arquivo util.js:
  - **O segundo parâmetro da função define** retorna um objeto JavaScript que define a(s) funcionalida(s) do módulo. No exemplo, o módulo util.js exporta apenas a função adicioneUm.

```
1  // util.js - com módulo
2
3  // Este módulo utiliza o módulo 'underscore' e exporta um objeto
4  // que contém a função 'adicioneUm'.
5
6  define(['underscore'], function(_) {
7
8    return {
9      adicioneUm: function(n) {
10       if (_.isNumber(n))
11         return n + 1;
12       else
13         return 0;
14     }
15   };
16 });
```

RequireJS: exemplo com módulo

- Arquivo util.js:
  - Resumindo: o módulo util.js depende do módulo underscore e disponibiliza a função adicioneUm.

```
1  // util.js - com módulo
2
3  // Este módulo utiliza o módulo 'underscore' e exporta um objeto
4  // que contém a função 'adicioneUm'.
5
6  define(['underscore'], function(_) {
7
8    return {
9      adicioneUm: function(n) {
10       if (_.isNumber(n))
11         return n + 1;
12       else
13         return 0;
14     }
15   };
16 });
```

RequireJS: exemplo com módulo

- Arquivo bib1.js:
  - Este módulo **depende do módulo util e exporta a função f**. Observação: a função dobre é interna ao módulo e não pode ser invocada por outro módulo.

```
1  // bib1.js - com módulo
2
3  // este módulo depende do módulo util
4  define(['util'], function(ut) {
5      // função que só existe dentro do módulo
6      function dobre(n) {return 2*n;}
7
8      // define o que o módulo exporta.
9      // no caso, exporta a função f
10     return {
11         f: function(v) {
12             var temp = ut.adicioneUm(v);
13             return dobre(temp);
14         }
15     };
16 });
```

RequireJS: exemplo com módulo

- Arquivo bib2.js:
  - Este módulo **exporta a função f**. A função multiplicarPorDez é interna ao módulo e não pode ser invocada por outro módulo.

```
1  // bib2.js - com módulo
2
3  define(function() {
4    // função visível apenas internamente ao módulo
5    function multiplicarPorDez(n) {
6      return n * 10;
7    }
8
9    // este módulo exporta um objeto contendo apenas a
10   // função f. Poderia conter outras funções ou mesmo
11   // outros objetos.
12   return {
13     f: function(a) {
14       return multiplicarPorDez(a);
15     }
16   };
17 });
```

RequireJS: exemplo com módulo

- Arquivo main.js:

```
1  // main.js - com módulo
2
3  requirejs.config({
4    // define onde estão os arquivos .js
5    baseUrl: 'js/com_modulo/bib',
6    // define nome de módulo diferente do nome do arquivo .js. É útil
7    // quando, por exemplo, o nome do arquivo .js inclui o número da
8    // versão da biblioteca (como em bib-1.2.3.js) pois pode-se atualizar
9    // o arquivo (digamos bib-1.3.js) mantendo o nome do módulo.
10   paths: {
11     // underscore passa a ser o nome do módulo para o
12     // arquivo underscore-min.js
13     underscore: 'underscore-min'
14   },
15   // com o shim podemos modularizar bibliotecas que não seguem
16   // o padrão de módulo usado pelo RequireJS.
17   shim: {
18     underscore: {exports: '_'}
19   }
20 });
```

RequireJS: exemplo com módulo

- Arquivo main.js:

```
22 // o programa Javascript abaixo utiliza as bibliotecas/módulos bib1 e bib2. O
23 // valor 'bib1' corresponde, segundo a configuração definida acima, ao
24 // arquivo 'js/com_modulo/bib/bib1.js'. O mesmo vale para 'bib2'.
25 // Os parâmetros 'b1' e 'b2' representam os objetos exportados pelos módulos.
26 // Observe que, assim, não há mais conflito envolvendo a função 'f'.
27
28 requirejs(['bib1', 'bib2'], function(b1, b2) {
29   var d = document.getElementById("dados");
30   d.innerHTML += "bib1.f(10) = ";
31   d.innerHTML += b1.f(10);
32   d.innerHTML += "<br>";
33   d.innerHTML += "bib2.f(10) = ";
34   d.innerHTML += b2.f(10);
35 });
```

RequireJS: exemplo com módulo

- Graças ao RequireJS, temos como resultado: