

Criando um mashup

Produtividade e elegância no desenvolvi

Neste artigo vamos abordar de forma prática os desafios envolvidos no desenvolvimento de um Mashup completo em Java. Vamos explorar tecnologias e conceitos como REST, XML, XPath, JSON, DWR e como consumir web services para integração aos serviços da Amazon e do Yahoo na criação de um serviço simples para recomendação de livros Java.

Mas o que afinal é um Mashup?

De acordo com o Wikipedia um Mashup pode ser definido como uma aplicação web que combina dados de mais de uma origem em uma única ferramenta integrada. O conteúdo de um Mashup é recuperado através da utilização de API's de terceiros, geralmente na forma de web services.

Nos últimos anos o conceito de Mashup vem ganhando bastante força principalmente pelo apoio e incentivo de grandes empresas como Google, Yahoo e Amazon, que liberam o uso de suas API's para praticamente todos os seus serviços.

☞ *Veja a lista completa de API's do Google e Yahoo em <http://code.google.com/> e <http://developer.yahoo.com/>*

Exemplos de Mashups

Existem centenas, ou melhor, milhares de Mashups na internet, vários são criados diariamente, todos buscando seu lugar ao sol. Alguns criados por pura diversão, outros mais elaborados com plano de negócios e investimento pesado de empresas de capital de risco.

Um caso clássico de um Mashup que vem gerando muitos comentários ultimamente é o FriendFeed. Esse serviço é denominado de "agregador social" porque ele consolida as atualizações de sites sociais como blogs, bookmarks, redes sociais, entre outros, permitindo que pessoas com contas em vários sites sociais pela internet como Flickr, Twitter, Digg, Amazon, Last.fm, entre outros, consigam agrupar atualizações sobre suas atividades online em um só lugar.

Resumo DevMan

De que se trata o artigo:

Este artigo fala sobre como criar um Mashup em Java integrando serviços do Yahoo e Amazon para descobrir o numero de referências na internet para um determinado livro.

Para que serve:

Serve para construir aplicações que utilizam recursos e serviços de API's de terceiros, gerando uma nova aplicação que combina funcionalidades das API's utilizadas.

Em que situação o tema é útil:

Sua utilidade não se restringe à criação de Mashups, mas sim aos conceitos e tecnologias adotadas que podem ser aplicadas para inúmeros tipos de projetos que necessitem de algum tipo de integração.

Criando um Mashup em Java:

O Mashup criado nesse artigo utiliza API's da Amazon e do Yahoo para recuperar um livro e o número de suas referências encontradas na web. Três classes foram criadas, uma para comunicação com a Amazon utilizando XML, outra com o Yahoo no formato JSON e uma terceira para a integração dos dois serviços. A camada visual foi desenvolvida em HTML/CSS com jQuery e DWR.



Nota do Devman

Veja o que você vai aprender adicionalmente neste artigo:

- O conceito de DWR;
- A definição de Separation of Concerns.

Outro Mashup recente que anda bastante comentado é o Mozilla Ubiquity, projeto ainda experimental. É uma extensão ao navegador Firefox que permite associar ações

em Java mento para web

Libere sua criatividade e desenvolva aplicações Web 2.0 aliando o poder da linguagem Java com a grande abundância de API's disponíveis na internet

EDUARDO SASSO

a eventos de teclado, por exemplo: atualizar meu status no twitter ou facebook, criar um mapa no Google Maps baseado em um endereço selecionado na janela do navegador e coisas assim; em teoria qualquer aplicação web que disponibilize uma API pode ser integrado ao Ubiquity, permitindo criar os mais diversos tipos serviços.

Exemplos adicionais:

- <http://www.flickrovision.com> – Integração entre Google Maps e Flickr;
- <http://www.wikicrimes.org> – Mashup de Wiki com Google Maps;
- <http://ping.fm/> – Atualiza informações em redes sociais;
- <http://www.4hoursearch.com/> – Site de busca baseado na API do Yahoo Search BOSS.

Porque criar um Mashup?

Em primeiro lugar porque é extremamente divertido e desafiador. Independente da linguagem de programação ou metodologia adotada, o que mais importa na criação de um Mashup bem sucedido é a criatividade, sem contar ainda com o provável reconhecimento da comunidade de programadores e pessoas antenadas nesse meio caso seu serviço seja um sucesso.

Outro motivo é que ele pode servir como um ótimo portfólio em seu currículo, principalmente se você entrou no mercado recentemente e ainda não tem muita experiência profissional.

Como criar um Mashup em Java?

Como citado há pouco, o aspecto mais importante na criação de uma Mashup é a criatividade, sendo necessária uma boa dose de inspiração e o resto é pura codificação. É neste ponto que esperamos contribuir com este artigo.

Para criar um Mashup precisamos entender conceitos como REST, XML e JSON, pois, muito provavelmente, qualquer API

na internet que você venha a utilizar vai utilizar essas tecnologias.

REST

Representational State Transfer ou simplesmente REST, é um estilo de arquitetura ou “pattern” utilizado para construir Web Services que geralmente, mas não necessariamente, transmitem dados através do protocolo HTTP sem nenhuma camada de mensagem adicional como o SOAP. Veja os artigos sobre REST das Edições 54 e 56.

XML

O XML, Extensible Markup Language, é uma especificação para criação de linguagens de markup customizadas. É classificada como “Extensible” pois permite que os usuários definam seus próprios elementos. Seu objetivo principal é ajudar sistemas independentes a trocar dados estruturados principalmente pela internet.

JSON

JavaScript Object Notation, JSON, é um formato leve para troca de dados que pode

ser visto como alternativa ao XML. Baseado em texto e extremamente legível, o formato JSON é utilizado para representar estruturas de dados simples, geralmente para transmitir dados estruturados através da rede em um processo chamado de serialização.

Como forma de assimilação mais fácil para o conteúdo deste artigo, desenvolvemos um simples Mashup que pode ser baixado e modificado livremente (mais detalhes no final do artigo). A aplicação exemplo consiste na integração entre os serviços Amazon AWS e Yahoo Search BOSS, (falaremos mais sobre esses serviços a seguir) onde seu simples propósito é mostrar aleatoriamente um livro Java na lista dos mais vendidos da Amazon e descobrir o número de referências encontradas para esse livro na internet. Veja na **Figura 1** o resultado final esperado para o nosso Mashup de exemplo.

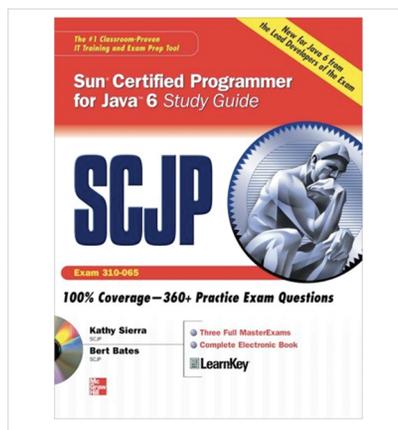
Amazon Web Services

O Amazon AWS é uma coleção de web services oferecidos pela Amazon para criação dos mais diversos tipos de solução, incluindo serviços como:

- *Amazon Elastic Compute Cloud (EC2)*: para criação de servidores privados virtuais;
- *Amazon Simple Storage Service (S3)*: para armazenamento;
- *Amazon Associates Web Services*: para consulta ao catálogo completo de produtos da Amazon.

Yahoo Search BOSS

YahooSearchBOSS ou simplesmente BOSS (Build your Own Search Service) é uma nova iniciativa do Yahoo que coloca todo o seu poder computacional e infraestrutura de *search engine* à disposição dos programadores, sem qualquer tipo de limitação, com o objetivo de alavan-



SCJP Sun Certified Programmer For Java 6 Exam 310-065

Price: \$49.99
Yahoo Links: 145
Editorial Review

Figura 1. Mashup que mostra um livro Java direto da Amazon e o número de links no Yahoo.

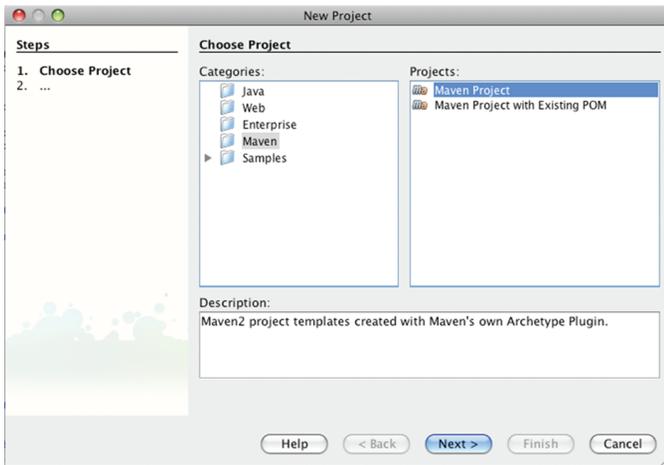


Figura 2. Criando um novo projeto Maven no NetBeans.

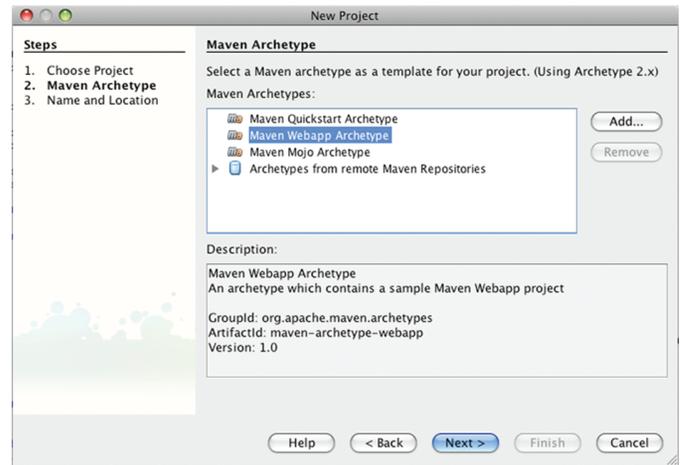


Figura 3. Criando um projeto web a partir do Maven.

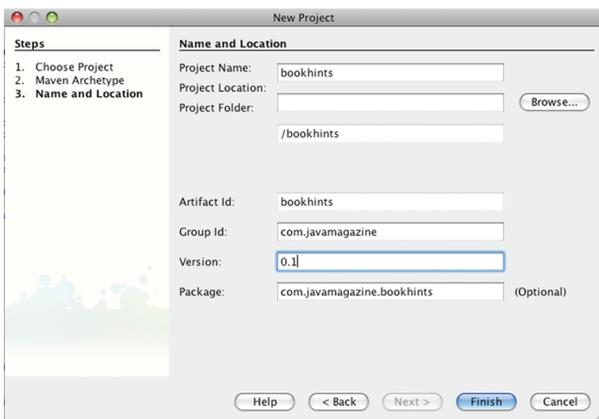


Figura 4. Detalhes do projeto.

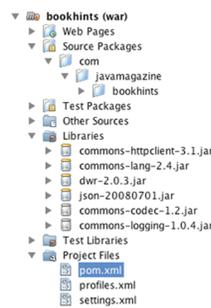


Figura 5. Estrutura completa do projeto no NetBeans.

Preencha o restante das informações do projeto de forma semelhante à **Figura 4**.

Com o ambiente devidamente configurado, modifique o arquivo `pom.xml` criado pelo NetBeans e acrescente as informações de dependência do projeto como visto na **Figura 5** e **Listagem 1**, respectivamente.

Na **Listagem 1** temos o arquivo `pom.xml` completo para o nosso serviço. O `pom.xml` é o arquivo de configuração para projetos baseados no Maven. Como podemos ver nesta listagem, são informados os metadados do projeto como nome, forma de distribuição e principalmente dependências utilizadas.

Com a infra-estrutura completa podemos começar a realmente desenvolver nosso Mashup.

Conversando com a Amazon

Na criação de nosso Mashup existem dois arquivos principais, o `AmazonAws.java` responsável pela comunicação com a API da Amazon para recuperação de livros, e o arquivo `YahooSearchBOSS.java` para a integração com o mecanismo de busca do Yahoo que veremos no próximo tópico.

O objetivo da classe `AmazonAws` pode ser definido em duas etapas, preparação da requisição ao Amazon e recuperação e parse dos resultados.

Preparação da requisição ao Amazon

Para fazer uma requisição ao serviço de web services da Amazon, conhecido simplesmente por AWS, precisamos de uma série de parâmetros, felizmente bem documentados. Como requisito obrigatório para utilização do AWS, é necessário a

car inovações no segmento de buscas na internet.

O Yahoo Search BOSS oferece cinco tipos de buscas, são elas:

- Web Search;
- Image Search;
- News Search;
- Spelling Suggestions.

Requisitos

Por motivos de conveniência e preferência, a aplicação exemplo foi desenvolvida com o NetBeans 6.1, mas qualquer outra IDE pode ser utilizada sem nenhum problema.

Para o controle do projeto utilizamos o Maven, essa ferramenta simplifica muito etapas tediosas como gerenciamento de dependências, compilação e distribuição de aplicações.

Além das ferramentas citadas acima, as seguintes bibliotecas foram utilizadas no projeto:

- **commons-httpclient**: para realização de requisições http;
- **commons-lang**: biblioteca utilitária para fins diversos;
- **json**: para manipulação de dados neste formato;
- **dwr**: para permitir a interação entre JavaScript e Java.

☑ *Se você não tem familiaridade com o Maven, sugerimos a leitura do artigo Gerenciando projetos com Maven publicado na Edição 62.*

Configurando o Ambiente

O primeiro passo é criar um novo projeto no NetBeans através do menu `File>New Project` e para o tipo de projeto vamos selecionar `Maven Project`, como mostra a **Figura 2**.

Na janela seguinte, de acordo com a **Figura 3**, vamos escolher o `Maven Webapp Archetype` como o template padrão para a criação de nosso projeto.

Listagem 1. Arquivo pom.xml com a configuração completa para o projeto incluindo dependências.

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.
w3.org/2001/XMLSchema-instance" xsi:schemaLocation="http://maven.apache.org/
POM/4.0.0 http://maven.apache.org/maven-v4_0_0.xsd">
  <modelVersion>4.0.0</modelVersion>
  <groupId>com.javamagazine.bookhints</groupId>
  <artifactId>bookhints</artifactId>
  <packaging>war</packaging>
  <version>0.1</version>
  <name>bookhints</name>
  <URL>http://maven.apache.org</URL>
  <dependencies>
    <dependency>
      <groupId>junit</groupId>
      <artifactId>junit</artifactId>
      <version>3.8.1</version>
      <scope>test</scope>
    </dependency>
    <dependency>
      <groupId>commons-httpclient</groupId>
      <artifactId>commons-httpclient</artifactId>
      <version>3.1</version>
    </dependency>
    <dependency>
      <groupId>org.json</groupId>
      <artifactId>json</artifactId>
      <version>20080701</version>
    </dependency>
    <dependency>
      <groupId>org.directwebremoting</groupId>
      <artifactId>dwr</artifactId>
      <version>2.0.3</version>
    </dependency>
    <dependency>
      <groupId>commons-lang</groupId>
      <artifactId>commons-lang</artifactId>
      <version>2.4</version>
    </dependency>
  </dependencies>
  <build>
    <finalName>bookhints</finalName>
    <plugins>
      <plugin>
        <artifactId>maven-compiler-plugin</artifactId>
        <version>2.0.2</version>
        <configuration>
          <source>1.5</source>
          <target>1.5</target>
        </configuration>
      </plugin>
    </plugins>
  </build>
  <properties>
    <netbeans.hint.deploy.server>J2EE</netbeans.hint.deploy.server>
  </properties>
</project>
```

Listagem 2. Método que monta uma URL para requisição para a Amazon.

```
private String getRequestURL() {
    StringBuilder URL = new StringBuilder();
    URL.append("http://ecs.amazonaws.com/onca/xml");
    URL.append("?Service=AWSECommerceService");
    URL.append("&AWSAccessKeyId=" + associateId);
    URL.append("&Operation=ItemSearch");
    URL.append("&SearchIndex=Books");
    URL.append("&Sort=relevancerank");
    URL.append("&Condition=All");
    URL.append("&MerchantId=All");
    URL.append("&Availability=Available");
    URL.append("&BrowseNode=3608");
    URL.append("&ResponseGroup=Images,ItemAttributes,EditorialReview");
    URL.append("&Version=2008-08-19");

    return URL.toString();
}
```

criação de um *AssociateId* no próprio site da Amazon (gratuitamente).

Praticamente toda a funcionalidade que vemos no site da Amazon pode ser utilizada de forma programável, o que torna sua API bastante completa e extensiva, permitindo várias combinações de parâmetros para chegar ao resultado desejado.

Em nosso caso específico queremos recuperar livros sobre Java, ordenados pela relevância e que estejam disponíveis para compra. Como resultado desta requisição esperamos informações sobre cada livro, como por exemplo, título, preço, imagem da capa e a crítica do site. Para montar essa requisição o código deve ser semelhante ao da **Listagem 2**.

Como teste vamos simplesmente executar a URL gerada no navegador para termos certeza que os parâmetros estão corretos e também para verificar o XML retornado pela Amazon. Acompanhe na **Listagem 3** a URL completa utilizada na requisição do web service.

☞ *Detalhes sobre os parâmetros utilizados na requisição a Amazon estão fora do escopo deste artigo. Para mais detalhes sobre o Amazon AWS recomendamos consultar a ótima documentação disponível online, veja no final do artigo links relacionados.*

Recuperação e Parse dos resultados

Com a URL da requisição preparada, precisamos além de executar fazer também o parsing do XML resultante, filtrando e adaptando o retorno gerado de acordo com nossas necessidades.

Para realizar o parsing do resultado da requisição ao AWS optamos pela utilização do suporte XML do próprio JDK, o JAXP (Java API for XML Processing).

Antes de seguirmos adiante precisamos entender o conceito de DOM e como criar um parser DOM em Java. Optamos pelo DOM – Documento Object Model, pois ela é uma especificação do W3C (World Wide Web Consortium) que disponibiliza interfaces de programação para documentos XML e HTML, permitindo manipular dinamicamente sua estrutura ou conteúdo.

Para criar um parser DOM utilizando a JAXP devemos obter uma instância da classe **DocumentBuilderFactory** através do método estático **newInstance()**. Em seguida solicita-

Listagem 3. URL completa para requisição ao Amazon.

```
http://ecs.amazonaws.com/onca/xml?Service=
AWSECommerceService&AWSAccessKeyId=04E530AYP52MY2142S02
&Operation=ItemSearch&SearchIndex=Books&Sort=
relevancerank&Condition=All&MerchantId=All&Availability=
Available&BrowseNode=3608&ResponseGroup=
Images,ItemAttributes,EditorialReview&Version=2008-08-19
```

Listagem 4. Trecho de código necessário para fazer a requisição e Parse de um documento XML.

```
DocumentBuilderFactory factory = DocumentBuilderFactory.
newInstance();
DocumentBuilder builder = factory.newDocumentBuilder();
Document doc = builder.parse(getRequestUrl());
```

Listagem 5. Código fonte para o método getBooks() da classe AmazonAWS.

```
public List<Book> getBooks() {
    List<Book> books = new ArrayList();

    try {
        DocumentBuilderFactory factory =
            DocumentBuilderFactory.newInstance();

        DocumentBuilder builder = factory.newDocumentBuilder();
        Document doc = builder.parse(getRequestUrl());

        XPathFactory xPathFactory = XPathFactory.newInstance();
        XPath xpath = xPathFactory.newXPath();

        int items = doc.getElementsByTagName("Item").
            getLength();

        for (int i = 1; i < items + 1; i++) {
            Book book = new Book();
            book.setAsin(xpath.evaluate(getXPathExpr("ASIN", i),
                doc));
            book.setTitle(xpath.evaluate(getXPathExpr(
                "ItemAttributes/Title", i), doc));
            book.setIsbn(xpath.evaluate(getXPathExpr(
                "ItemAttributes/ISBN", i), doc));
            book.setPrice(xpath.evaluate(getXPathExpr(
                "ItemAttributes/ListPrice/FormattedPrice", i),
                doc));
            book.setUrl(xpath.evaluate(getXPathExpr(
                "DetailPageURL", i), doc));
            book.setImage(xpath.evaluate(getXPathExpr(
                "LargeImage/URL", i), doc));

            book.setReview(xpath.evaluate(getXPathExpr(
                "EditorialReviews/EditorialReview/Content", i),
                doc));

            books.add(book);
        }

        return books;
    } catch (Exception ex) {
        ex.printStackTrace();
        return null;
    }
}
```

Listagem 6. Formato da URL para uma solicitação de serviço ao Yahoo Search BOSS.

```
http://boss.yahooapis.com/ysearch/web/v1/{query}?appid=
xyz[&param1=val1&param2=val2&etc
```

Listagem 7. URL para pesquisa no Yahoo BOSS utilizando o ISBN 0596007124 como parâmetro.

```
http://boss.yahooapis.com/ysearch/web/v1/ISBN:0596007124?
appid=uFe0qCjV34GsN0fNriwZvtS6FMUoCVugiga5Gs875VsNpyU66Fw0
pa00vP0yABb9N6n02A--
```

Listagem 8. Retorno JSON para a requisição efetuada na Listagem 7.

```
"ysearchresponse":{
  "responsecode":"200",
  "nextpage":"/ysearch/web/v1/ISBN:0596007124?appid=
123&start=10",
  "totalhits":"462",
  "deephits":"2590",
  "count":"10",
  "start":"0",
  "resultset_web":[
    {
      "abstract":"Head First Design Patterns and other
books by",
      "clickurl":"http://www.alibris.com/search/books/
isbn/0596007124",
      "date":"2008/08/09",
      "dispurl":"www.<b>alibris.com</b>/search/
books/<b>isbn</b>/<wbr><b>0596007124</b>",
      "size":"90465",
      "title":"Head First Design Patterns by Eric
Freeman, Elisabeth",
      "url":"http://www.alibris.com/search/books/
isbn/0596007124"
    }, ...
  ]
}
```

Listagem 9. Código da classe YahooSearchBOSS.

```
package com.javamagazine.bookhints;

import org.apache.commons.httpclient.HttpClient;
import org.apache.commons.httpclient.HttpMethod;
import org.apache.commons.httpclient.methods.GetMethod;
import org.json.JSONObject;

public class YahooSearchBOSS {
    private String appid;

    public YahooSearchBOSS(String appid) {
        this.appid = appid;
    }

    public String getTotalHits(String query) {
        String totalhits = null;
        String url = getRequestUrl(query);

        HttpClient client = new HttpClient();
        HttpMethod method = new GetMethod(url);

        try {
            client.executeMethod(method);
            String result = method.getResponseBodyAsString();

            JSONObject json = new JSONObject(result);
            totalhits = json.getJSONObject("ysearchresponse").
                getString("totalhits");
        } catch (Exception ex) {
            //
        } finally {
            method.releaseConnection();
        }
        return totalhits;
    }

    private String getRequestUrl(String query) {
        StringBuilder url = new StringBuilder();
        url.append("http://boss.yahooapis.com/ysearch/web/
v1/");
        url.append(query);
        url.append("?appid=");
        url.append(appid);

        return url.toString();
    }
}
```

mos ao **DocumentBuilderFactory** para recuperar uma instância da classe **DocumentBuilder**, que contém o método **parse(String url)** aceitando como argumento uma string no formato de URL, fazendo automaticamente a requisição e o parsing do documento. O retorno será uma instância da classe **Document** com a representação do documento XML em memória, como mostra o trecho de código da **Listagem 4**.

O passo seguinte é identificar e extrair as informações desejadas, tornando seu resultado facilmente manipulável. Existem várias maneiras de extrair informações de um arquivo XML, a própria Amazon criou uma biblioteca em Java específica para abstrair essa necessidade do programador, mas como nosso exemplo é bastante simples resolvemos recuperar as informações necessárias utilizando o padrão XPath.

Fazendo referência à Wikipedia, podemos dizer que o XPath é uma linguagem para selecionar “nodos” de um documento XML. A linguagem XPath é baseada na representação do documento XML em forma de árvore e provê a habilidade de navegar por essa árvore usando vários tipos de critérios.

Veja na **Listagem 5** a lógica completa envolvida para extrair informações do XML, prestando atenção ao retorno do método **getBooks()** na forma de **List<Book>**. Conseguimos abstrair e encapsular toda a complexidade envolvida no consumo do web service, retornando uma simples coleção de alto nível composta pelo **POJO Book**.

☞ *Veja no final do artigo links para tutoriais sobre a tecnologia XPath.*

Falando com o BOSS

Em nosso exemplo vamos utilizar a funcionalidade de Web Search do Yahoo Search BOSS para encontrar o número total de referências para um determinado livro utilizando como critério de busca seu número de ISBN.

Diferentemente da Amazon, que retorna XML, o Yahoo retorna seu resultado no formato JSON, um padrão bastante legível, eficiente, simples e enxuto em sua manipulação tanto para extrair quanto para gerar dados.

Listagem 10. Código completo para a classe principal do projeto de Mashup - Amazon.java.

```
package com.javamagazine.bookhints;

import java.util.List;
import org.apache.commons.lang.math.RandomUtils;

public class Amazon {

    public Book getRandomBook() {
        String associateId = "04E530AYP52MY2142S02";
        String associateTag = "context0b-20";

        String yahooAppid = "uFe0qCjV34GsN0fNriwZVtS6FMUoCVugiga5Gs875VsNpyU66Fw0pa00vP0yABb9N6n02A-";

        AmazonAws amazonAws = new AmazonAws(associateId, associateTag);
        List<Book> books = amazonAws.getBooks();
        int id = RandomUtils.nextInt(books.size());
        Book book = books.get(id);
        YahooSearchBOSS ysboss = new YahooSearchBOSS(yahooAppid);
        String totalHits = ysboss.getTotalHits("ISBN:" + book.getIsbn());
        book.setTotalHits(totalHits);

        return book;
    }
}
```

Listagem 11. Código completo para o arquivo dwr.xml.

```
<!DOCTYPE dwr PUBLIC
"-//GetAhead Limited//DTD Direct Web Remoting 2.0//EN"
"http://getahead.org/dwr/dwr20.dtd">
<dwr>
  <allow>
    <convert converter="bean" match="com.context.bookhints.Book" />

    <create creator="new" javascript="Amazon">
      <param name="class" value="com.context.bookhints.Amazon"/>
    </create>
  </allow>
</dwr>
```

Listagem 12. Código para o arquivo web.xml.

```
<!DOCTYPE web-app PUBLIC
"-//Sun Microsystems, Inc.//DTD Web Application 2.3//EN"
"http://java.sun.com/dtd/web-app_2_3.dtd" >
<web-app>
  <display-name>Archetype Created Web Application</display-name>

  <servlet>
    <servlet-name>dwr-invoker</servlet-name>
    <display-name>DWR Servlet</display-name>
    <servlet-class>org.directwebremoting.servlet.DwrServlet</servlet-class>
    <init-param>
      <param-name>debug</param-name>
      <param-value>true</param-value>
    </init-param>
  </servlet>

  <servlet-mapping>
    <servlet-name>dwr-invoker</servlet-name>
    <url-pattern>/dwr/*</url-pattern>
  </servlet-mapping>

</web-app>
```

Antes de entrarmos na parte do código, vamos analisar a estrutura de uma requisição ao BOSS e o seu resultado no formato JSON. Ao contrário da Amazon o BOSS não oferece tantas possibilidades de configuração, o que torna seu uso ainda mais simples

e com uma curva de aprendizado bastante baixa. A essência de uma requisição ao BOSS pode ser vista na **Listagem 6**.

Repare na **Listagem 6** que os únicos parâmetros necessários para uma requisição são:

Listagem 13. Código do arquivo index.html.

```

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
  <head>
    <title>Random Java Books by Amazon</title>
    <meta http-equiv="Content-Type" content="text/html; charset=MacRoman">
    <script type="text/javascript" src="/dwr/interface/Amazon.js"></script>
    <script type="text/javascript" src="/dwr/engine.js"></script>
    <script src="jquery-1.2.6.pack.js" type="text/javascript"
      charset="utf-8"></script>
    <script src="scripts.js" type="text/javascript" charset="utf-8"></script>
    <link href="style.css" rel="stylesheet" type="text/css" media="screen"/>
  </head>
  <body>
    <div id="loader"></div>
    <div id="container">
      <div id="content">
        <div id="book">
          <div id="cover">
            <div id="cover_img">
              <a href="#" id="url" target="_blank">
                
              </a>
            </div>
          </div>
          <div id="details">
            <div id="table">
              <div id="title">
                <a href="#" id="title_url" target="_blank"></a>
              </div>
              <div id="price">
                Price:
              </div>
              <div id="totalhits">
                Yahoo links: <a href="#" id="isbn_search" target="_blank"></a>
              </div>
              <div id="review">
                <a href="#" id="hideshow_review">Editorial Review</a>
              </div>
            </div>
          </div>
          <div id="review">
            <div id="content_review"></div>
          </div>
        </div>
      </div>
    </div>
  </body>
</html>

```

- **query**: o que vai ser pesquisado, em nosso caso um ISBN;
- **appid**: identificação do usuário, basta preencher o cadastro no site e receber por email em instantes.

Vamos fazer o teste de uma requisição para avaliar e interpretar o resultado gerado pelo Yahoo. A requisição pode ser vista na **Listagem 7** e um trecho de seu resultado devidamente formatado pode ser visto na **Listagem 8**.

Repare na **Listagem 8** a linha em negrito onde diz *totalhits*, essa é a informação que desejamos extrair em Java e que veremos a seguir.

Antes de mais nada, vamos criar uma nova classe em nossa aplicação, a **YahooSearch**

BOSS. Em linhas gerais o código necessário para recuperar a informação sobre **TotalHits** é de apenas duas linhas, o restante é apenas código auxiliar, necessário para montar e realizar a requisição ao serviço e tratar exceções. Veja na **Listagem 9** o código completo para a classe **YahooSearchBOSS**.

Observe nas linhas em negrito na **Listagem 9**, esse é o código necessário para extrair a informação sobre o número total de referências encontradas para nossa busca. Simples e ágil, não é à toa que a adoção ao padrão JSON vem crescendo a cada dia, principalmente devido à sua facilidade de uso.

☞ *Não vamos entrar em detalhes sobre a utilização do Apache commons httpclient, basta dizer que é uma biblioteca bastante utilizada*

para fazer requisições HTTP em Java; para mais informações veja as referências no fim do artigo.

Finalmente o Mashup

Com as classes para consumo de web services da Amazon e Yahoo prontas, chegou a hora de fazer sua integração – o tão aguardado Mashup.

Resolvemos chamar essa nova classe simplesmente de **Amazon**, pois o principal serviço utilizado por ela é o da própria Amazon, com o Yahoo BOSS sendo apenas um complemento para enriquecer as informações do livro recuperado. Posteriormente vamos configurar essa classe como um objeto **DWR**, tornando seus métodos acessíveis via JavaScript para a implementação da camada visual de nosso Mashup.

Na **Listagem 10** é possível acompanhar o código completo para a classe **Amazon** com suas principais operações marcadas em negrito.

O código da **Listagem 10** é bastante simples e auto-explicativo, mas vale acrescentar que ele simplesmente instancia a classe **AmazonAWS** e aleatoriamente escolhe um livro, atribuindo a este livro o seu número total de referências na internet através da instanciação da classe **YahooSearchBOSS** e da execução do método **getTotalHits()**.

Parte visual

Com o objetivo de promover uma separação clara entre as camadas de nossa aplicação seguindo o princípio “Separation of Concerns”, optamos por utilizar a tecnologia DWR para comunicação com o código Java através de JavaScript. Dessa maneira conseguimos manter a camada visual enxuta e legível, perfeito para os web designers de plantão.

Para simplificar ainda mais adicionamos o suporte jQuery ao projeto. jQuery é uma

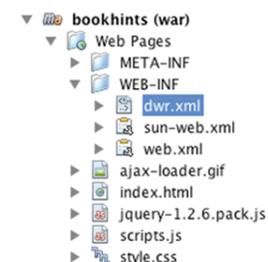


Figura 6. Local do arquivo dwr.xml no projeto.

Listagem 14. `scripts.js`, arquivo javascript que utiliza `dwr` para se comunicar com java no lado servidor e recuperar aleatoriamente um livro para mostrar no navegador.

```

$(document).ready(function(){
    $("#container").toggle();

    Amazon.getRandomBook(function(book){
        $("#review #content_review").toggle();
        $("#loader").toggle();

        $("#url").attr("href", book.url);
        $("#image").attr("src", book.image);

        $("#review #content_review").html(book.review);
        $("#hideshow_review").toggle(
            function(){
                $("#review #content_review").fadeIn("slow");
            },
            function(){
                $("#review #content_review").fadeOut("slow");
            });

        $("#title_url").attr("href", book.url);
        $("#title_url").text(book.title);

        $("#price").append(book.price);

        $("#isbn_search").attr("href", "http://search.yahoo.com/search?p=ISBN:"
            + book.isbn);
        $("#isbn_search").text(book.totalHits);

        $("#container").toggle();
    });

    function center()
    {
        left = (parseInt($("#loader")[0].offsetWidth / 2) - 110) + "px";
        $("#loader").css("marginLeft", left);
    }

    center();
});

```



Nota do Devman

DWR: DWR, ou Direct Web Remoting, é uma biblioteca Open Source Java que auxilia desenvolvedores a escrever web sites que incluem tecnologia Ajax. Ele permite que código Javascript no browser utilize métodos Java que estão rodando no servidor como se essas funções estivessem presentes no próprio navegador.

do DWR ao arquivo `pom.xml` do Maven, restando apenas criar o arquivo `dwr.xml` e alterar o `web.xml` do projeto incluindo uma entrada para seu servlet de controle.

O arquivo `dwr.xml` deve ser criado dentro do diretório **WEB-INF** ao lado do arquivo `web.xml`, como mostra a **Figura 6**.

O código completo do arquivo `dwr.xml` pode ser visto na **Listagem 11**. Detalhes da utilização do DWR estão fora do escopo deste artigo, basta dizer que os comandos da **Listagem 11** simplesmente mapeiam a classe `Amazon` para um objeto JavaScript `Amazon`, que ficará disponível na forma de um arquivo a ser incluído em nosso arquivo HTML.

A configuração do arquivo `web.xml` para suporte ao DWR pode ser vista na **Listagem 12** e é auto-explicativa, simplesmente declarando o servlet do DWR ao projeto.

Vamos agora criar o arquivo HTML com a estrutura visual da aplicação e logo em seguida o código JavaScript responsável por recuperar e apresentar as informações de nosso Mashup.

O arquivo HTML vai se chamar `index.html` e deve ser criado na raiz da pasta `Web Pages` do projeto do NetBeans, como pode ser visto na **Figura 6**.

Na **Listagem 13** é possível ver o código completo do arquivo `index.html`. É importante prestar atenção aos itens marcados em negrito, ali estão declarados os arquivos JavaScript do DWR, jQuery e outros que criamos como `scripts.js` e `style.css` para formatação visual como cor, fontes, etc.

Após criar o arquivo `index.html` precisamos criar o arquivo `scripts.js`, que é responsável pela recuperação e apresentação do Mashup e vai utilizar o jQuery para a manipulação do HTML. Siga os mesmos passos utilizados para a criação do `index`.

biblioteca JavaScript bastante leve e poderosa que simplifica muito a manipulação de código HTML, como veremos a seguir.

☑ *Se você não tem familiaridade com o jQuery, sugerimos a leitura dos artigos `Javascript Fácil`*

com jQuery – partes 1 e 2 – publicados nas Edições 54 e 55.

Antes de continuar precisamos configurar o DWR em nosso projeto. No início do artigo já adicionamos a dependência

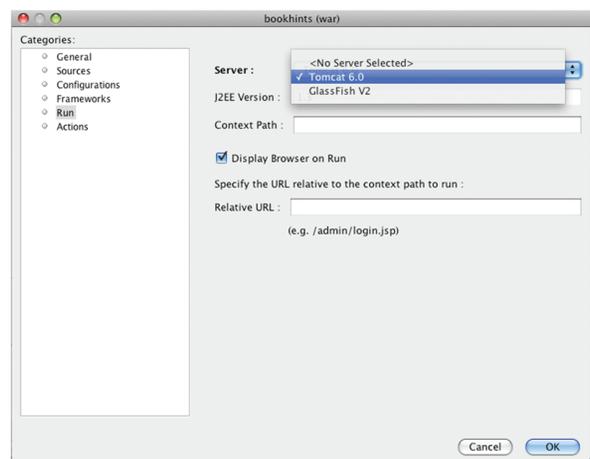


Figura 7. Configurando o servidor de aplicações no NetBeans.



Nota do Devman

Separation of Concerns: É o processo de separar um programa de computador em partes distintas que sobreponham suas funcionalidades o mínimo possível, aumentando a habilidade de gerenciar e manter sistemas e promovendo ordem e elegância ao design da aplicação.

html. O código completo do arquivo pode ser visto na **Listagem 14**.

Mais uma vez repare no item em negrito da **Listagem 14**, principalmente na declaração **function(book)**. Essa linha de código vai utilizar o poder do DWR para instanciar a classe **Amazon** e executar o método **getRandomBook()**.

Como resultado do método **getRandomBook()**, recebemos um objeto **Book** que é convertido pelo DWR em uma função de callback JavaScript. Esta função pode ser identificada pela instrução **function(book)**, onde o objeto **Book** é recebido como argumento, permitindo fácil manipulação e apresentação de seus atributos na tela.

Executando

Para testar nossa aplicação precisamos configurar o servidor de aplicação no

NetBeans. Para fazer isso clique com o botão direito em cima do projeto e escolha a opção *Properties*, em seguida selecione o item *Run*, o resultado deve ser semelhante à **Figura 7**. No item *Server* selecione o servidor de aplicação desejado, em nosso caso utilizamos o Tomcat 6.0. Também é possível executar com o servidor Glassfish V2 que acompanha o NetBeans.

Para rodar a aplicação clique novamente com o botão direito em cima do projeto escolhendo o item *Properties*, e logo em seguida a opção *Run*. Após a compilação e distribuição do projeto no servidor de aplicações, o NetBeans se encarrega de carregar a aplicação diretamente em seu navegador.

Se tudo ocorrer como esperado, você deve ter um resultado semelhante ao da **Figura 1**.

Conclusão

Procuramos mostrar nesse artigo que o Java pode ser um sério concorrente a outras tecnologias mais tradicionais na hora de desenvolver um Mashup ao estilo Web 2.0. Seu poder, aliado à sua grande flexibilidade e ao extenso número de bibliotecas de apoio, tornam o processo de criação de serviços para internet bastante produtivo e elegante.



Eduardo Sasso

eduardo@context.com.br, *www.context.com.br*

é desenvolvedor e entusiasta Java com certificação SCJP, atualmente envolvido no desenvolvimento de aplicações Web 2.0 e Mashups utilizando Wicket, Maven, Hibernate, Spring, DWR e MySQL nos serviços *www.tradd.us*, *www.openjobs.com.br* e *www.abduzeedo.com*.



Dê seu feedback sobre esta edição!

A Java Magazine tem que ser feita ao seu gosto. Para isso, precisamos saber o que você, leitor, acha da revista!

Dê seu voto sobre este artigo, através do link:

www.devmedia.com.br/javamagazine/feedback



<http://bookhints.context.com.br>

Versão online do projeto.

<http://context.com.br/jm-mashup.zip>

Código fonte do projeto utilizado como exemplo no artigo.

[http://en.wikipedia.org/wiki/Mashup_\(web_application_hybrid\)](http://en.wikipedia.org/wiki/Mashup_(web_application_hybrid))

Mashup no Wikipedia.

<http://www.programmableweb.com/>

Site que acompanha Mashups e APIs disponíveis para programação.

<http://mashupawards.com/>

Mashup Awards.

<http://www.ibm.com/developerworks/xml/library/x-mashups.html>

Mashups: The new breed of Web app.

<http://www.programmableweb.com/howto>

How To Make Your Own Web Mashup.

http://java.sun.com/developer/technicalArticles/J2EE/mashup_1/

Mashup Styles, Part 1: Server-Side Mashups.

<http://www.amazon.com/AWS-home-page-Money/b?ie=UTF8&node=3435361>

Amazon Web Services @ Amazon.com.

<http://developer.yahoo.com/search/boss/>

Yahoo! Search BOSS.

http://en.wikipedia.org/wiki/Representational_State_Transfer

Representational State Transfer – REST.

<http://en.wikipedia.org/wiki/XML>

Extensible Markup Language – XML.

http://en.wikipedia.org/wiki/Java_API_for_XML_Processing

Java API for XML Processing – JAXP.

<http://en.wikipedia.org/wiki/JSON>

JavaScript Object Notation – JSON.

<http://en.wikipedia.org/wiki/XPath>

XML Path Language – Xpath.

<http://www.w3schools.com/XPath/default.asp>

XPath Tutorial.

<http://hc.apache.org/httpclient-3.x/>

Apache Commons HttpClient.

<http://directwebremoting.org/>

DWR – Easy Ajax for Java.

<http://jquery.com/>

The Write Less, Do More, JavaScript Library.

