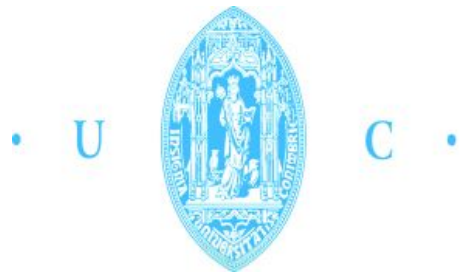

Faculdade de Ciências e Tecnologia

Universidade de Coimbra - DEI

IS Project 1 - XML and XML Manipulation, Java Message Service (JMS) and Message Oriented Middleware (MOM)

Mestrado em Engenharia Informática
autores: João Batanete 2009113460, Ricardo Rei 2014233736

Introdução

Nesta secção iremos fazer uma breve descrição das tecnologias utilizadas ao longo do projeto.

Java Message Service(JMS):

O JMS é uma API de Java criada para troca de mensagens entre dois ou mais processos e/ou máquinas do mesmo sistema distribuído(Message Oriented Middleware). Um MOM*,* na prática, oferece uma solução para o problema consumidor/produtor com determinadas vantagens e desvantagens sobre outros métodos utilizados para este fim(por exemplo, ligações TCP). A maioria destas diferenças prende-se com o facto de um MOM fornecer comunicação assíncrona entre os constituintes do sistema(ou seja, não existe nenhum tipo de ligação constante entre eles, e o emissor da mensagem não recebe um *acknowledge* após a receção da sua mensagem). Isto traz as seguintes vantagens, entre outras:

- Uma vez que os componentes não necessitam de aguardar pela execução uns dos outros para enviar as mensagens, existe um maior desacoplamento temporal entre estes. Por exemplo, o cliente poderá enviar uma mensagem ao servidor e continuar a executar as suas outras tarefas que não dependem da receção de uma resposta, sem recurso a *threads*.
- O cliente que faz o *request* não necessita de esperar pela resposta do cliente para continuar a execução(não bloqueante).
- Necessita de um menor número de máquinas do lado do servidor para garantir semântica *exactly-once*, não necessitando de utilizar o modelo *two-phase commit,* o que indiretamente também ajuda a prevenir falhas, uma vez que é possível o sistema ter menos componentes para satisfazer os *requests* do cliente.
- As mensagens são enviadas de forma *store-and-forward* não sendo necessário o recetor estar *online* aquando do envio, nem o emissor aquando da receção.

No entanto, em situações em que pretendemos ter sincronismo entre os componentes do sistema, o modelo *publish/subscribe* perde muitas das suas vantagens, uma vez que adiciona *overhead* extra à aplicação com a espera pela resposta, e é também em geral um modelo de programação menos familiar aos *developers,* e o JMS em específico ainda possui, a nosso ver, muito espaço para melhorar a nível de facilidade de utilização.

eXtensible Markup Language(XML):

Esta linguagem de markup permite a codificação de documentos num formato acessível quer para um leitor humano, quer para uma máquina(através de *parsing*). Entre outras aplicações, encontra grande uso na troca de informação entre duas componentes de um sistema, sendo facilmente portável entre tecnologias e/ou linguagens de programação diferentes.
Os *schemas* XML trazem a possibilidade de introduzir regras e normas a um documento XML, por forma a tornar mais fácil a sua leitura por parte de um programa, e possibilitar a sua validação sintática antes da leitura dos dados. Embora existam outras tecnologias destinadas a esta validação, para este projeto foi utilizado XSD, que também utiliza XML na sua sintaxe.
Para a conversão dos ficheiros em HTML para melhor leitura dos dados extraídos do website foi utilizada tecnologia XSLT, conforme especificado mais abaixo.
Uma alternativa possível ao uso de XML para o fim dado durante o projeto seria JSON, que em geral possui menor overhead aquando da leitura por parte do computador, mas é também mais complexo de ler para um leitor humano e de validar.

JSoup:

JSoup é uma API de Java que possibilita a extração de dados de uma página HTML. Funciona através de expressões regulares especificadas na própria documentação para aceder a elementos específicos da página, possibilitando assim a extração de texto e/ou informações das mesmas conforme o utilizador pretender, programaticamente.

Componentes do sistema

Nesta secção será descrito o funcionamento e as decisões tomadas ao longo da implementação das várias componentes do projeto.

Nota geral:

Ao longo de todo o projeto, optámos pela utilização de um estilo de programação procedimental(ao contrário de OOP, como é habitual em Java), com recurso sobretudo a métodos estáticos. Seria possível cada componente do sistema ser implementado recorrendo ao modelo de programação *singleton.* No entanto, esta opção apenas iria, da nossa perspetiva, introduzir esforço desnecessário à realização do projeto.
Por outro lado, os componentes do sistema que serão executados indefinidamente (Keeper e Summary Creator) fazem uso de Hook Threads que fecham o contexto JMS aberto quando o programa fecha, através do tratamento do sinal SIGINT. Isto impede que sessões sejam deixadas abertas a cada execução da aplicação, não sendo necessário reiniciar o servidor wildfly entre cada uma.

Crawler:

Esta aplicação tem como função a extração da informação pretendida do website (recorrendo ao JSoup após a obtenção do DOM da página), a sua conversão para o formato XML e a consequente publicação da mesma num tópico do servidor Wildfly.

As expressões JSoup utilizadas para aceder aos elementos necessários foram obtidas sobretudo a partir de um processo tentativa-erro, recorrendo à ferramenta disponível no link try.jsoup.org e a inspeções manuais do código HTML. A informação encontra-se organizada da seguinte forma:

Para obter o nome e sigla de cada país, foram utilizadas as seguintes expressões:

- “[data-odfcode]”(elementos com atributo data-odfcode)
- Ao percorrer a lista de elementos obtidos da expressão anterior, aplicar a expressão “.country”(elementos da classe country). O tamanho da lista é também o número total de países que ganhou medalhas.
- Para cada país, a sigla está presente no texto do primeiro elemento obtido e o nome completo no segundo

Para obter a informação relativa às medalhas propriamente ditas(nome do atleta ou equipa, desporto, categoria e tipo de medalha) foram utilizadas as expressões:

- “.table-expand”(elementos da classe table-expand). Obtemos um elemento por cada lista de medalhas de um país.

- para cada elemento obtido, aplicar a expressão “tr”(elementos do tipo tr). Obtemos os elementos de cada medalha do país atual.
- "td.col-1"(elementos do tipo td, e com classe col-1). Caso este campo possua um tipo de medalha(Bronze,Silver ou Gold) sabemos que todas as medalhas subsequentes(incluindo a atual) irão ter este tipo. Caso contrário continuamos a considerá-las do tipo lido anteriormente.
- ".col-2"(elementos da classe col-2). Obtemos o elemento com o texto relativo ao desporto da medalha.
- ".col-3"(elementos da classe col-3). Obtemos o elemento com o texto relativo à categoria do desporto da medalha.
- ".col-4"(elementos da classe col-4) Obtemos o elemento com o texto relativo ao atleta ou equipa que ganhou a medalha.

Após a extração da informação e a criação dos objetos java, o Crawler irá criar a string com o XML relativo à informação obtida e publicá-la no tópico.

Keeper:

O Keeper tem como função armazenar a informação atualizada das medalhas em memória após a sua leitura a partir do tópico, a validação do XML e responder aos pedidos do *Requester*. Na nossa implementação, estas duas operações são realizadas por duas *threads* separadas, de forma concorrente. Na subscrição feita ao tópico, é utilizada uma subscrição durável, para permitir a leitura de mensagens publicadas com a aplicação em baixo.

Uma das duas threads principais monitoriza o tópico, aguardando mensagens por ler publicadas pelo Crawler. Quando uma nova mensagem é lida com o XML de novas medalhas, a lista de medalhas existente em memória é atualizada.

Após a receção de uma *query* através de uma mensagem da fila, é criada uma *thread* separada para atender o cliente. Esta começa por fazer *parsing* da *query* com um *split* e filtra as medalhas consoante os termos de pesquisa recebidos. Seguidamente junta todos os resultados relevantes numa string e envia-a ao Requester através da fila temporária criada por este e referenciada no *reply to* da mensagem. Caso ainda não tenha sido lido nenhuma mensagem do tópico com o XML das medalhas, limita-se a emitir uma mensagem “NOT AVAILABLE” ao cliente.

Para garantir que não existem problemas de concorrência nos acessos à lista de medalhas(uma vez que temos duas ou mais *threads* a aceder a esta, e uma delas irá

realizar operações de escrita), foi utilizado um *lock* da classe *ReadWriteLock*, sendo o *lock* fechado para operações de escrita quando estamos a atender o pedido de um cliente, e para operações de leitura quando estamos a atualizar a lista de medalhas recebidas.

Requester:

Esta aplicação simula o papel de "cliente" do sistema, recebendo uma "*query*" como *input* do utilizador, e envia-a para a fila de mensagens do Keeper, obtendo uma lista de medalhas. Para tal, cria uma fila temporária aquando do envio da mensagem e modifica o *reply to* da mensagem para esta fila temporária, aguardando depois pela resposta do Keeper na mesma.

Para efeitos de simplificação de testes e execução, foi utilizada a seguinte sintaxe para as *queries*(o campo "country" pode referir-se tanto à sigla como ao nome completo do país):

country;sport;category;athlete name;medal type

Summary Creator:

O HTML Summary Creator trata-se da componente responsável pela criação de um site HTML, utilizando os dados contidos no ficheiro XML gerado pelo Crawler. Esta aplicação começa por se conectar a um tópico JMS onde residem mensagens que contêm os ficheiros XML publicados previamente pelo Crawler, lê as mensagens XML do tópico((subscrição durável), valida o XML obtido recorrendo ao XSD e por fim gera o HTML. De forma a gerar o HTML, O Summary Creator, recorrer a um ficheiro XSL (extensible stylesheet layout) que permite gerar o HTML a partir do XML. A versão do XSLT utilizada foi a 2.0, uma vez que foi necessário utilizar a instrução *xsl:for-each-group* no ficheiro XSL de forma a contar o número de cada tipo de medalha ganho por cada atleta. De forma a suportar a versão 2.0 do XSLT foi também necessário adicionar a jar externa Saxon-HE, assim como definir o Saxon como processador de XSLT no java utilizando a instrução:

System.setProperty("javax.xml.transform.TransformerFactory","net.sf.saxon.TransformerFactoryImpl");

Relativamente ao layout da página pretendeu-se apresentar toda a informação obtida pelo Crawler dividida em 3 categorias principais, as medalhas conquistadas por cada país, as medalhas conquistadas por cada atleta e categorias de desporto praticadas por cada atleta. Desta forma é possível apresentar toda a informação de forma bastante organizada. Uma vez que a informação é consideravelmente extensa, considerou-se importante que a página apresentasse algumas funcionalidades para filtragem dos

resultados obtidos, de forma a que o utilizador possa interagir com os resultados e obter a informação pretendida de forma mais eficiente. Assim, optou-se por utilizar a biblioteca jQuery Dynatable, esta permite transformar os valores das tabelas existentes no código HTML num array de objetos JSON, onde cada objeto JSON corresponde a uma linha na tabela, consoante a acção efectuada pelo utilizador na página, o JSON é modificado e escrito novamente no DOM na forma de elementos da tabela HTML.

A página HTML gerada é apresentada na Figura 1.

Figura 1: Página HTML gerada pelo Summary Creator

Referências

- slides da disciplina
- https://www.rio2016.com/en/medal-count-country
- http://eai-course.blogspot.pt (blog do professor)
- https://en.wikipedia.org/wiki/XML
- https://en.wikipedia.org/wiki/Jsoup
- https://en.wikipedia.org/wiki/Message-oriented_middleware
- https://jsoup.org/cookbook/extracting-data/selector-syntax
- https://www.dynatable.com/