

UM ESTUDO COMPARATIVO SOBRE FORMATOS DE DADOS

Relatório final do projeto acadêmico

Aluno

Nome: Rafael Sales Pavarina

RA: 726583

Turma: B

Orientador

Prof. Dr. Delano Medeiros Beder

Resumo:

Este estudo visa desenvolver uma aplicação web baseada em Java que trabalhe com a requisição e transferência de dados no formato JSON e XML, e estudar seu comportamento e eficiência em relação ao formato de dado utilizado e comparar os resultados obtidos para avaliar, dentre outros, os seguintes pontos: Qual o formato mais rápido, qual o formato mais leve, qual formato é mais simples para se manipular, quais alterações são necessárias na aplicação para que seja possível trabalhar com o tipo de dado escolhido. Além de perguntas como: As respostas para as perguntas anteriores continuam as mesmas se mudarmos a quantidade de informação que estamos tratando? Qual delas consegue suportar um maior número de requisições? Em qual o tempo de resposta foi menor? A seguir se encontra uma lista com palavras-chave e outras tecnologias que foram utilizadas no desenvolvimento deste projeto:

Palavras-chave:

HTML, CSS, Java, JSON, XML, HTTP, Servlets Java, JSP, Apache JMeter, NetBeans, Python

INTRODUÇÃO:

O desenvolvimento de software para a web é uma área da computação que evoluiu muito desde o seu surgimento. No decorrer dos anos tornou-se possível desenvolver aplicações cada vez mais poderosas, que utilizam diversos recursos importantes, dentre eles um recurso fundamental: A transferência de informação entre diferentes sistemas. Trata-se do envio de informações e dados entre aplicações, sistemas em escala global, tudo coordenado por protocolos de comunicação e dados que são formatados para o seu envio e recebimento em outra aplicação ou máquina.

Outras utilizações de transferência de informação em diferentes formatos de dados, muito importante hoje em dia são os Web Services, serviços que podem ser acessados ou consumidos através da rede de computadores por outros softwares ou dispositivos capazes de receber informações, processar dados e retornar informações. Podem ser definidos também como um conjunto elaborado de métodos utilizados para tornar possível a comunicação entre o computador e o sistema.

Este projeto acadêmico tem como finalidade desenvolver uma Aplicação Web baseada em Java, que seja capaz de ler arquivos externos tanto no formato XML quanto JSON, simulando um cenário real onde estes arquivos tenham sido gerados previamente por outra parte do sistema, e comparar o comportamento desta aplicação diante de alguns testes feitos sobre suas diferentes partes componentes. Nas seções a seguir serão introduzidos os conceitos dos formatos de dados, das tecnologias utilizadas e dos métodos escolhidos para a avaliação.

TRABALHOS CORRELATOS:

Após feita a revisão sistemática foram levantados 72 artigos, onde foram aceitos apenas 8, o que corresponde a 12% do total. As palavras chaves utilizadas eram bastante amplas e por isto as pesquisas retornavam demasiados artigos.

Era feita uma filtragem buscando por artigos que realizavam comparações e avaliações de resultados. No caso eram avaliados artigos que comparavam os formatos de dados, ainda que não todos ao mesmo tempo, apenas para ter conhecimento sobre resultados anteriores obtidos por outros pesquisadores.

Dentre os artigos selecionados para a leitura, alguns deles se destacaram, todos foram úteis até certo ponto, por introduzirem muito bem os conceitos que serão utilizados nas próximas seções deste artigo.

Os artigos “Comparision between JSON and XML in applications on AJAX”, “Analysis of the Efficiency of Data Transmission Format Based on AJAX Applications” e “Using AJAX to Optimize Web Services Performance” tiveram uma prioridade maior por abordarem diretamente os formatos de dados desejados e realizavam comparações ao decorrer do artigo.

FUNDAMENTAÇÃO TEÓRICA E TÉCNICA:

Aqui serão introduzidos alguns conceitos, tecnologias, métodos de avaliação e comparação. Primeiramente serão abordados os conceitos fundamentais de cada tipo de dado.

Tipos de dados comparados:

XML: O primeiro formato de dado tratado é o formato de dado XML, ou eXtensible Markup Language (Linguagem de Marcação Extensível). Muito similar ao conhecido HTML (HyperText Markup Language). O XML é uma linguagem multi-plataforma, que utiliza de uma estrutura de árvore com diversos subníveis, para a organização do conteúdo. Todo o conteúdo é organizado e exibido dentro de Tags.

Uma tag é uma marcação no formato “<nome_da_tag>conteúdo_da_tag</nome_da_tag>”, onde as tags são os pedaços entre “<>”.

A seguir, imagine que tenhamos o seguinte cenário: Uma turma de alunos, com apenas 2 alunos (para não ficar muito extenso), e cada aluno possui nome e RA. O código gerado, referente a estes dados, em XML é o seguinte:

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <classe>
3      <aluno>
4          <nome>Rafael</nome>
5          <ra>726583</ra>
6      </aluno>
7      <aluno>
8          <nome>Amigo</nome>
9          <ra>123456</ra>
10     </aluno>
11 </classe>
```

Figura 1: Exemplo de XML para o exemplo dado.

JSON: O segundo formato de dado é o formato de dado JSON, ou JavaScript Object Notation (Notação de Objeto JavaScript). É um formato mais leve e enxuto para troca de dados, mais fácil de gerar e interpretar, o que acaba por tornar o AJAX mais apresentável facilmente. Também é multi-plataforma e pode ser utilizado em conjunto com outras linguagens de programação.

No JSON existem alguns tipos diferentes de dados possíveis: Números (reais e inteiros), denotados pelo próprio número; Datas; Valores lógicos; Strings, denotadas por uma cadeia de caracteres entre aspas; Vetores, denotados por um conjunto de elementos entre colchetes, separados por vírgulas; E outros Objetos JSON (JSONObjects), que compõem estruturas mais complexas, aninhadas uma dentro da outra.

A estrutura básica do JSON segue o formato “Chave:valor”, onde uma estrutura complexa pode ser definida como um conjunto de pares, cada atributo é identificado pela chave e ao utilizar a chave seu valor é conseguido indiretamente.

Agora será apresentado o código equivalente ao da Figura 1, utilizando JSON:

```
1  {
2    "classe": {
3      "aluno": [
4        {
5          "nome": "Rafael",
6          "ra": 726583
7        },
8        {
9          "nome": "Amigo",
10         "ra": 123456
11       }
12     ]
13   }
14 }
15
```

Figura 2: Exemplo de JSON para o exemplo dado.

Observe que a presença das chaves e dos colchetes é indispensável para delimitar as estruturas presentes em um JSON, pois neste formato não há a presença das tags.

O protocolo HTTP:

O protocolo HTTP, ou HyperText Transfer Protocol é o protocolo que dita a comunicação que ocorre na internet. Quando o navegador quer se comunicar com o servidor (comunicação client-server) a comunicação depende de algumas regras. Estas regras são chamadas de protocolos. O HTTP é o protocolo responsável por ditar como é feita esta comunicação.

O HTTP possui seus métodos para definir diferentes mensagens que serão transmitidas entre o servidor e o computador do usuário (cliente), dentre eles, 2 são os mais conhecidos, são eles o GET e o POST.

O método GET é utilizado quando desejamos obter um conteúdo ou uma resposta, como o resultado de uma busca, por exemplo. Podemos notar a requisição do tipo GET quando olhamos em uma URL de resposta um conteúdo exibido após o caminho da página atual:

```
?nome_do_parametro=seu_valor&nome_do_outro_param=valor
```

Figura 3: Exemplo de parâmetros e valores

Já o método POST funciona de maneira diferente, os parâmetros não exibidos explicitamente como é o caso do GET. No POST os parâmetros são enviados diretamente no corpo do HTTP, sendo acessado apenas na outra ponta da conexão.

Existe ainda o HTTPS, que funciona de maneira semelhante ao HTTP, mas com uma camada de segurança SSL/TLS, que criptografa a informação de ponta a ponta, que no HTTP transitam como texto puro, o que é perigoso.

Java Servlets:

A segunda tecnologia utilizada são os Java Servlets. Como visto, toda a comunicação web é feita através de requisições e respostas via o protocolo HTTP. O papel dos Servlets é abstrair parte desta comunicação por meio de métodos específicos para o tratamento da comunicação HTTP.

Com estas funções o trabalho de gerar a requisição HTTP (HttpServletRequest), trabalhar a resposta HTTP (HttpServletResponse), estabelecer um canal de comunicação, dentre outras funções, é facilitado e implementado por trás dos métodos utilizados.

Em um arquivo do tipo Java Servlet é possível encontrar os métodos **doGet** e **doPost**, que são chamados quando uma Servlet é acionada, mas estes métodos são específicos para o tipo de método utilizado, doGet para GET e doPost para POST. Existe ainda o método **processRequest**, que é chamado independentemente do método quando os métodos específicos não estão presentes. Estes métodos são encarregados de interpretar a HTTP Request e gerar a HTTP response, que corresponde a uma página web, normalmente gerada de maneira dinâmica, por isto os parâmetros enviados na requisição.

```
protected void processRequest(HttpServletRequest request, HttpServletResponse response)
    throws ServletException, IOException {
    // CORPO DA RESPOSTA
}
```

Figura 4: Método processRequest sem corpo

A segunda característica mais marcante das Servlets é a maneira como ela gera a resposta (response). Uma Servlet é um arquivo Java puro, isto é, a parte HTML não é exibida diretamente, mas sim indiretamente, por meio de funções do Java. E o corpo da resposta acaba sendo gerado todo por meio de funções em Java ao invés de escritas diretamente:

```
try (PrintWriter out = response.getWriter()) {
    out.println("<!DOCTYPE html>");
    out.println("<html>");
    out.println("<head>");
    out.println("<title>Servlet</title>");
    out.println("</head>");
    out.println("<body>");
    out.println("<h1>Hello World! </h1>");
    out.println("</body>");
    out.println("</html>");
}
```

Figura 5: Exemplo de HTTP Response

Tudo é controlado por um objeto da classe PrintWriter que invoca métodos de impressão e o HTML é gerado indiretamente por eles.

JSP: A terceira tecnologia é o JSP, ou Java Server Pages. O JSP é mais recente que as Servlets e surgiu para facilitar ainda mais a criação das páginas geradas dinamicamente.

É muito semelhante à linguagem de programação PHP (PHP Hypertext Processor). Com o uso de JSP, se torna possível utilizar Java como principal estrutura do Back-end da aplicação e realizar a comunicação com HTTP. JSP também abstrai ainda mais as funcionalidades dos Servlets, é o equivalente a literalmente digitar trechos de código Java no meio do código HTML e vice-versa.

```
<%  
// AQUI É INSERIDO CÓDIGO JAVA  
%>  
  
<%@page contentType="text/html" pageEncoding="UTF-8"%>  
<!DOCTYPE html>  
<html>  
  <head>  
    <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">  
    <title>JSP Page</title>  
  </head>  
  <body>  
    <%  
    // AQUI TAMBÉM É INSERIDO CÓDIGO JAVA  
    %>  
    <h1>Hello World!</h1>  
  </body>  
</html>
```

Figura 6: Exemplo de um arquivo JSP

No JSP o código Java fica mais separado do HTML do que com as Servlets. Os trechos destacados em azul correspondem à parte do código onde ficam as funções em Java, delimitados pelas tags “<%” e “%>”. Estes trechos Java são chamados Scriptlets.

Curiosidade: Todo arquivo JSP ao ser compilado é convertido para uma Servlet

Apache JMeter

É uma ferramenta para testes em aplicações Web. Com ela é possível realizar os mais diversos tipos de testes e simulações, como por exemplo simular diversos usuários fazendo requisições ao servidor ao mesmo tempo, onde cada usuário realiza diversas requisições uma após a outra. Também é possível testar o limite de requisições em um curto período que a página suporta até o servidor cair.

O uso desta ferramenta é indispensável para a obtenção de algumas das respostas para este projeto.

DESCRIÇÃO DO PROJETO E DOS PRINCIPAIS RESULTADOS OBTIDOS:

Fase de desenvolvimento:

O primeiro passo para a realização deste projeto foi o desenvolvimento de uma estrutura de páginas para esta aplicação. Ao final foi planejada a seguinte estrutura: Tela inicial, Tela de testes com XML para 100 usuários, Tela de testes com XML para 10000 usuários, Tela de testes para JSON com 100 usuários e Tela de testes para JSON com 10000 usuários. Estas 5 telas compõem a estrutura do projeto.

Para cada tela foi necessário desenvolver a parte visual (Front-end) utilizando HTML, CSS e Bootstrap. Também foi necessário configurar a parte lógica e de processamento dos dados (Back-end) utilizando Java. Para este projeto foi escolhido o JSP ao invés das Servlets, pela facilidade de interpretação pelo usuário que inspecionar o código.

O segundo passo foi o desenvolvimento dos arquivos necessários para teste. Este projeto trabalha com arquivos XML e JSON externos, e para desenvolvê-los foi criado um Projeto em Python, capaz de desenvolver toda a estrutura. O projeto Python conta com 4 Scripts, 2 para a criação dos XML, onde o primeiro apenas concatena os trechos do arquivo para economizar espaço (em bytes) e o segundo cria o XML formatado, com as quebras de linhas e os espaços, para facilitar a interpretação do usuário leitor. E foram desenvolvidos os mesmos 2 arquivos para o desenvolvimento dos arquivos JSON, um deles sem formatar e o outro formatando o JSON. A seguir os códigos em Python:

```
arquivo = open("dados10k.xml", "w")
numero_alunos = 10000      # parâmetro que indica a quantidade de alunos
alunos = numero_alunos+1   # parâmetro utilizado

arquivo.write("<?xml version='1.0'?'>\n")
arquivo.write("<lista>\n")

for i in range(alunos):
    ra = 726000 + i
    aluno = " <aluno>\n"
        " <nome>Nome Aluno {}</nome>\n"
        " <email>email_aluno{}@gmail.com</email>\n"
        " <ra>{}</ra>\n"
        " </aluno>\n".format(i, i, ra)
    arquivo.write(aluno)

arquivo.write("</lista>")
arquivo.close()
```

Figura 7: Código Python para criação do arquivo de testes XML

```
arquivo = open("dados10k.json", "w")
numero_alunos = 10000      # parâmetro que indica a quantidade de alunos
alunos = numero_alunos+1   # parâmetro utilizado

arquivo.write("{\n")
arquivo.write("  \"lista\": {\n")
arquivo.write("    \"aluno\": [\n")

for i in range(alunos):
    index = i
    ra = "{}".format(726000 + i)
    aluno = "      {\n"
        "        \"nome\": \"Nome Aluno \" + str(index) + "\",\n"
        "        \"email\": \"email_aluno{0}@gmail.com\",\n"
        "        \"ra\": {1}\n".format(i, ra)
    arquivo.write(aluno)
    if i != alunos-1:
        arquivo.write("      },\n")
    else:
        arquivo.write("      }\n")

arquivo.write("    ]\n")
arquivo.write("  }\n")
arquivo.write("}")
arquivo.close()
```

Figura 8: Código Python para criação do arquivo de testes JSON

Com estes scripts foi possível criar todos os arquivos XML e JSON necessários para os testes apenas mudando a variável responsável pelo número de usuários desejados. Para facilitar o entendimento do arquivo a seguir, o parâmetro foi utilizado com valor 2 (para criar apenas 2 registros):

```
{
  "lista": {
    "aluno": [
      {
        "nome": "Nome Aluno 0",
        "email": "email_aluno0@gmail.com",
        "ra": 726000
      },
      {
        "nome": "Nome Aluno 1",
        "email": "email_aluno1@gmail.com",
        "ra": 726001
      },
      {
        "nome": "Nome Aluno 2",
        "email": "email_aluno2@gmail.com",
        "ra": 726002
      }
    ]
  }
}
```

Figura 9: Arquivo JSON resultante (formatado)

```
<?xml version="1.0"?>
<lista>
  <aluno>
    <nome>Nome Aluno 0</nome>
    <email>email_aluno0@gmail.com</email>
    <ra>726000</ra>
  </aluno>
  <aluno>
    <nome>Nome Aluno 1</nome>
    <email>email_aluno1@gmail.com</email>
    <ra>726001</ra>
  </aluno>
  <aluno>
    <nome>Nome Aluno 2</nome>
    <email>email_aluno2@gmail.com</email>
    <ra>726002</ra>
  </aluno>
</lista>
```

Figura 10: Arquivo XML resultante (formatado)

Curiosidade: Os arquivos gerados pelo script Python e utilizados neste projeto possuem 50mil linhas cada um!

Com a estrutura das páginas prontas e estilizadas, e os arquivos externos prontos, o próximo passo foi desenvolver a parte funcional da aplicação. Cada arquivo JSP possui uma parte destinada à interpretação de arquivos XML e JSON externos, respectivamente.

Para interpretar o XML externo e transformar seu conteúdo em algo visível, foi gerado o seguinte código Java:

```

<%
String url = "xml\\dados10000sf.xml";
//abre o arquivo, lê o XML e normaliza ele
File file = new File(url);
DocumentBuilder dBuilder = DocumentBuilderFactory.newInstance().newDocumentBuilder();
Document doc = dBuilder.parse(file);
doc.getDocumentElement().normalize();

//acessa a lista de alunos
NodeList nList = doc.getElementsByTagName("aluno");
%>

```

Figura 11: Lendo e interpretando o arquivo XML externo

```

<%for (int temp = 0; temp < nList.getLength(); temp++) {
    Node nNode = nList.item(temp);
    if (nNode.getNodeType() == Node.ELEMENT_NODE) {
        Element eElement = (Element) nNode;%>
        <div class="card">
            
            <div class="card_container">
                <h4><b><%= eElement.getElementsByTagName("nome").item(0).getTextContent() %></b></h4>
                <p><%= eElement.getElementsByTagName("email").item(0).getTextContent() %></p>
                <p>RA: <%= eElement.getElementsByTagName("ra").item(0).getTextContent() %> </p>
            </div>
        </div>
    }
} %>

```

Figura 12: Criando a interface a partir do XML lido

Já para o código JSON foi possível utilizar uma biblioteca externa chamada “**json.simple**” que abstrai um pouco a parte de interpretação do JSON visando manter ou até mesmo aumentar a eficiência na leitura e interpretação, isto faz com que a eficiência seja aumentada.

Para interpretar o JSON externo e transformar seu conteúdo em algo visível, foi gerado o seguinte código Java:

```

<%
String url = "json\\dados10000sf.json";
FileReader reader = new FileReader(url);

JSONParser jsonParser = new JSONParser();
JSONObject jsonObject = (JSONObject) jsonParser.parse(reader);

// obtendo a lista
JSONObject lista = (JSONObject) jsonObject.get("lista");

// obtendo os alunos
JSONArray alunos = (JSONArray) lista.get("aluno");

// varrer o vetor item por item
Iterator i = alunos.iterator();
%>

```

Figura 13: Lendo e interpretando o arquivo JSON externo


```

<% while (i.hasNext()) {
    JSONObject aluno = (JSONObject) i.next();%>
    <div class="card">
        
        <div class="card_container">
            <h4><b><%= aluno.get("nome") %></b></h4>
            <p><%= aluno.get("email") %></p>
            <p>RA: <%= aluno.get("ra") %> </p>
        </div>
    </div>
<% } %>

```

Figura 14: Criando a interface a partir do JSON lido

Isto conclui a primeira etapa, que consiste em desenvolver a aplicação capaz de comparar os tipos de dados. Agora inicia-se a segunda etapa, responsável por comparar de fato os tipos de dados.

Fase de testes:

É importante ressaltar que a aplicação foi desenvolvida para simular um cenário mais próximo da realidade, para que fossem feitos testes sobre ele.

O passo a seguir envolve o desenvolvimento de um plano de testes utilizando o JMeter. Para isto foram criados 4 grupos de usuários, cada um contendo 100 usuários realizando 100 requisições cada, totalizando 10000 requisições totais.

Após realizados os testes, foram obtidos os seguintes valores:

Rótulo	# Amostras	Média	Mediana	90% Line	95% Line	99% Line	Mín.	Máx.	Vazão
XML10000	10000	1499	1488	1625	1682	1945	354	2513	66,1/sec
JSON100	10000	25	22	35	41	50	1	366	3154,6/sec
XML100	10000	53	38	87	124	315	1	583	1665,8/sec
JSON10000	10000	1277	1259	1396	1455	1585	245	1704	77,5/sec
TOTAL	40000	714	369	1518	1566	1701	1	2513	68,3/sec

Figura 15: Resultados dos testes do JMeter

- Na coluna “Rótulo” se encontra o nome de cada página acessada
- Na coluna “# Amostras” se encontra o número de testes executados (100 usuários fazendo 100 requisições cada).
- Em “Mín.” está o menor tempo de acesso (em ms)
- Em “Máx.” está o maior tempo de acesso (em ms)
- Em “Média” está o tempo médio de acesso (em ms)

CONCLUSÃO:

Nesta seção serão respondidas as perguntas que movimentaram este projeto. Iniciaremos comparando pequenas quantidades de informação.

1 - Qual o formato mais leve?

A seguir se encontra uma imagem contendo os arquivos e seus respectivos tamanhos, sobre eles serão feitas algumas observações:

	dados10000.json	1.600 KB	Arquivo JSON
	dados10000.xml	1.268 KB	Documento XML
	dados10000sf.xml	955 KB	Documento XML
	dados10000sf.json	741 KB	Arquivo JSON
	dados100.json	16 KB	Arquivo JSON
	dados100.xml	13 KB	Documento XML
	dados100sf.xml	10 KB	Documento XML
	dados100sf.json	8 KB	Arquivo JSON

Figura 16: Arquivos e seus tamanhos

Primeiro vamos observar o seguinte fato, os arquivos utilizados no projeto foram os arquivos com “sf” no nome, onde sf significa “sem formatação”

Quando comparamos os arquivos “dados100sf.json” e “dados100sf.xml” podemos ver que a diferença entre eles é de apenas 2KB (mas já são 20% a mais). Porém quando comparamos os arquivos formatados (com espaços em branco e quebras de linha) podemos notar que o JSON passa a ser maior que o XML, dentre outros motivos, este é um dos motivos para se utilizar arquivos não formatados, pois a eficiência não está ligada à estética e aparência.

Logo, considerando os arquivos corretos para se comparar (arquivos não formatados), **o formato mais leve é o JSON.**

2 - Qual formato é mais simples para se manipular?

Normalmente a manipulação dos dois tipos costuma ser de mesma complexidade, entretanto a manipulação de arquivos XML é menos trivial, pois ele trata elementos internos como vetores e isto faz com que os comandos de acesso sejam menos intuitivos, como foi visto na figura 12:

```
<b><%= eElement.getElementsByTagName("nome").item(0).getTextContent() %>
%= eElement.getElementsByTagName("email").item(0).getTextContent() %></b>
A: <%= eElement.getElementsByTagName("ra").item(0).getTextContent() %>
```

Figura 17: recorte da figura 12 contendo apenas os métodos de acesso interno

Precisamos buscar pelo item(0) em um elemento que não necessariamente é um vetor, isso não costuma ser trivial. Além de precisar de outros dois métodos para se obter o conteúdo: “getElementsByTagName()” e “getTextContent()”, enquanto no JSON se utiliza apenas o método “get()” passando como parâmetro a chave desejada, e o método retorna o conteúdo, seja ele uma String, um inteiro, um jsonArray ou um jsonObject, como visto na figura 14:

```
><b><%= aluno.get("nome") %>
<%= aluno.get("email") %></b>
RA: <%= aluno.get("ra") %>
```

Figura 18: recorte da figura 14 contendo apenas os métodos de acesso interno

Fica nítida a diferença. Portanto **o formato mais simples de se manipular é o formato JSON.**

3 - Qual o formato mais rápido?

Para responder a esta pergunta utilizaremos a figura com os resultados dos testes de performance.

Rótulo	# Amostras	Mín.	Máx.	Média
XML10000	10000	354	2513	1499
JSON100	10000	1	366	25
XML100	10000	1	583	53
JSON10000	10000	245	1704	1277
TOTAL	40000	1	2513	714

Figura 19: resultados dos testes de performance, com as colunas reorganizadas

Comparando os rótulos XML100 e JSON100, podemos concluir que **o formato com tempo de resposta mais rápido é o JSON**, com 25ms de média, comparado a 53 do XML.

Iremos agora responder a seguinte pergunta: As respostas para as perguntas anteriores continuam as mesmas se mudarmos a quantidade de informação que estamos tratando? Afinal, desejamos estudar os casos não somente onde ocorra um baixo tráfego de informação, queremos analisar situações onde os dados são aumentados em grande quantidade.

Para esta etapa utilizaremos os arquivos com 10000 no nome, correspondendo a 10000 registros de alunos, ou seja, 100 vezes maior que os arquivos anteriores. Começaremos comparando o tamanho do arquivo:

1 – Qual formato é mais leve?

Para responder a esta pergunta utilizaremos um recorte da figura 16, contendo apenas os arquivos com 10000 registros.





	dados10000.json	1.600 KB	Arquivo JSON
	dados10000.xml	1.268 KB	Documento XML
	dados10000sf.xml	955 KB	Documento XML
	dados10000sf.json	741 KB	Arquivo JSON

Figura 20: Recorte da figura 16, com os maiores arquivos

Podemos observar que o tamanho dos arquivos gerados aumenta em grande quantidade, mas as observações feitas para os arquivos de menor tamanho ainda são válidas para este contexto. O JSON não formatado ultrapassa o XML em tamanho, devido à sua quantidade maior de espaços em branco na formatação, entretanto, considerando os arquivos utilizados na execução da aplicação, **o formato JSON continua sendo menor e mais leve**

2 – Qual o formato mais simples de se manipular?

Considerando que a implementação da parte de interpretação é a mesma para todos, a complexidade não aumenta quando a quantidade de dados aumenta. Com isto, **JSON continua sendo mais simples de se trabalhar**

3 – Qual o formato mais rápido?

Para responder a esta pergunta utilizaremos a Figura 19 com os resultados dos testes, mostrada no item anterior. Nesta comparação **o JSON continua sendo o formato com tempo de resposta menor**, com 1277ms de média, comparado com 1499 do XML

Outras observações:

Por fim, algumas observações adicionais sobre os formatos de dados:

- Embora a performance do JSON seja superior à do XML, não podemos desconsiderar o fato de que o XML é mais facilmente interpretado pelo usuário leitor, enquanto o JSON não é de fácil interpretação;

- É mais fácil fazer alterações em um XML do que em um JSON, visto que este é mais fácil de se corromper a estrutura dos dados;

Curiosidade: Embora o uso dos dois formatos seja encontrado em mesma quantidade em aplicações reais, o desenvolvimento de aplicações móveis (mais especificamente Android) utiliza ambos em seus projetos, porém com finalidades diferentes: Para tráfego de informações utiliza-se JSON, visto que ele consome menos rede e é mais leve para ser enviado, considerando uma rede limitada. Porém, na especificação dos componentes de suas telas é utilizado puramente o XML, onde o mesmo é convertido em componentes Java na hora da compilação.

Referências bibliográficas

CHEN, Z. X. – “Using AJAX to Optimize Web Services Performance”.

KIM, Haeran; KIM, Haejin; JANG M.; HAN, S.; CEONG, H. – “A Comparison of Features of AJAX’s Data Formats for the Medicinal Plants Application”.

AIHKISALO, T.; PAASO, T. – “A Performance Comparison of Web Service Object Marshalling and Unmarshalling Solutions”.

WANG, P.; WU, X.; YANG, H. – “Analysis of the Efficiency of Data Transmission Format Based on AJAX Applications”.

LIN, B.; CHEN, Y.; CHEN, X.; YU, Y – “Comparision Between JSON and XML in Applications on AJAX”.

GIMÉNEZ, J.; CAPPO, C. – “HTTP-WS-AD: An Anomaly Detector oriented to web applications and web services”.

Caelum: Apostila Java para Desenvolvimento Web, Curso FJ-21.