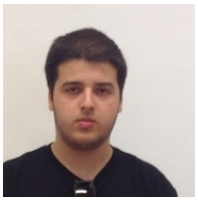
TP2-Gestão de Redes-MIEI

João Bernardo Freitas a74814

29 Janeiro 2020



1 Introdução

Neste trabalho é pedido para desenvolver um programa de monitorização que seja configurável através do browser e monitorize através da ferramenta **SNMP**.
Para a resolução com sucesso deste trabalho desenvolvi uma interface no browser com base em **HTML** e **JavaScript** sendo que para haver comunicação entre a interface e o *backend*, escrito em **Java**, utilizei a API **JAVAX** e **Ajax**,*Asynchronous JavaScript and XML*.
Por fim a API **SNMP4J** foi utilizado para poder utilizar as primitivas oriundas do **SNMP** em linguagem **Java**.

2 Resolução

Tendo em conta que o programa deve monitorizar a utilização de recursos de cada processo activo num certo host decidi utilizar as seguintes **oid's**.

- **hrSWRunName**-*.1.3.6.1.2.1.25.4.2.1.2* —indica o nome dos processos que estão activos.
- **hrSWRunPerfMem**-*.1.3.6.1.2.1.25.5.1.1.2* —indica a memória em Bytes que cada processo tem alocado.
- **hrSWRunPerfCPU**-*.1.3.6.1.2.1.25.5.2.1.1* —indica o nº de centi-segundos de recursos de CPU consumidos por cada processo.

De seguida desenvolvi o backend da aplicação, tendo como base o exemplo fornecido pelos docentes. Esse backend, escrito em **Java**, tem que conseguir obter o endereço, a porta e os segundos fornecidos pelos utilizador e, através desses, obter o resultado das primitivas.
Para tal desenvolvi o método **doPost** que dependendo do espaço de monitorização fornecidos irá fazer um **snmpWalk** a cada:

- 1 segundo: se o espaço de monitorização for menor que 20 segundos
- 5 segundos: se o espaço de monitorização for menor que 200 segundos
- 10 segundos: se o espaço de monitorização for maior ou igual a 200 segundos

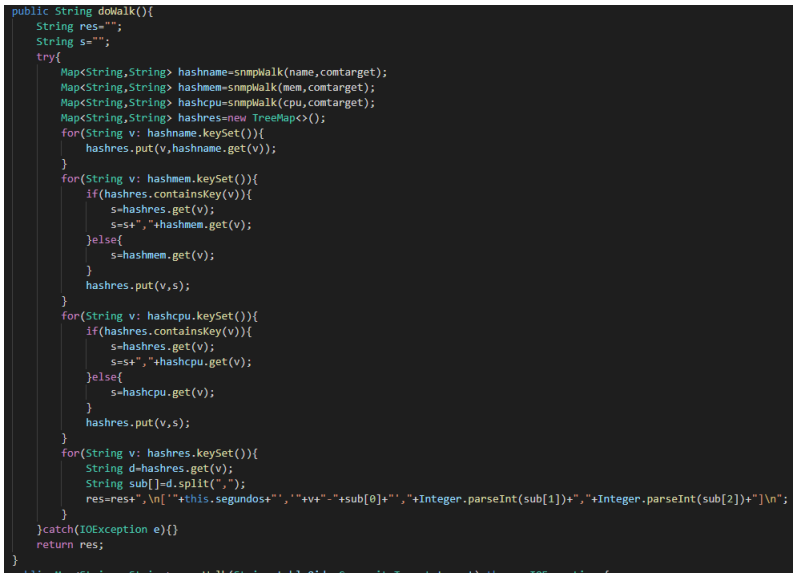
```java
public class BlockingServlet extends HttpServlet {
    /**
     *
     */
    private static final long serialVersionUID = 1L;
    //https://stackoverflow.com/questions/3831680/httpservletrequest-get-json-post-data
    protected void doPost(HttpServletRequest request, HttpServletResponse response) throws IOException {
        String ip=request.getParameter("adressInput");
        String porta=request.getParameter("portaInput");
        String sec=request.getParameter("secInput");
        int segundos=Integer.parseInt(sec);
        int i=0;
        int d=0;
        SNMPHandler snmpHandler=new SNMPHandler(ip, porta);
        String resp="";
        if(segundos<50){
            d=1;
        }else{
            if(segundos<200){
                d=5;
            }else{
                d=10;
            }
        }
        while(i<segundos){
            snmpHandler.setSec(i);
            resp =resp+snmpHandler.doWalk();
            i+=d;
            try{
                Thread.sleep(d*1000);
            }catch(InterruptedException ex){
                Thread.currentThread().interrupt();
            }

        }
        response.setContentType("application/json");
        response.setStatus(HttpServletResponse.SC_OK);
        response.getWriter().println("['Segundo','Processo','hrSWRunPerfMem','hrSWRunPerfCPU']"+resp);
    }
}
```

Figure 1: Código doPost

Para termos comunicação **SNMP->Java** foi desenvolvida a seguinte classe que através da primitiva **snmpwalk** irá obter os valores das *oid's* referenciadas em cima e irá criar uma string contendo esses resultados.

```java
public Map<String, String> snmpWalk(String tableOid, CommunityTarget target) throws IOException {
    Map<String, String> result = new TreeMap<>();
    TransportMapping<? extends Address> transport = new DefaultUdpTransportMapping();
    Snmp snmp = new Snmp(transport);
    transport.listen();

    TreeUtils treeUtils = new TreeUtils(snmp, new DefaultPDUFactory());
    List<TreeEvent> events = treeUtils.getSubtree(target, new OID(tableOid));
    if (events == null || events.size() == 0) {
        System.out.println("Error: Unable to read table...");
        return result;
    }

    for (TreeEvent event : events) {
        if (event == null) {
            continue;
        }
        if (event.isError()) {
            System.out.println("Error: table OID [" + tableOid + "] " + event.getErrorMessage());
            continue;
        }

        VariableBinding[] varBindings = event.getVariableBindings();
        if (varBindings == null || varBindings.length == 0) {
            continue;
        }
        for (VariableBinding varBinding : varBindings) {
            if (varBinding == null) {
                continue;
            }
            String v=varBinding.getOid().toString();
            String[] sub=v.split("\\.");
            String s=sub[sub.length-1];
            result.put("." + s, varBinding.getVariable().toString());
        }

    }
    snmp.close();
```

Figure 2: Código snmpwalk



```
public String doWalk(){
    String res="";
    String s="";
    try{
        Map<String,String> hashname=snmpWalk(name,comtarget);
        Map<String,String> hashmem=snmpWalk(mem,comtarget);
        Map<String,String> hashcpu=snmpWalk(cpu,comtarget);
        Map<String,String> hashres=new TreeMap<>();
        for(String v: hashname.keySet()){
            hashres.put(v,hashname.get(v));
        }
        for(String v: hashmem.keySet()){
            if(hashres.containsKey(v)){
                s=hashres.get(v);
                s=s+","+hashmem.get(v);
            }else{
                s=hashmem.get(v);
            }
            hashres.put(v,s);
        }
        for(String v: hashcpu.keySet()){
            if(hashres.containsKey(v)){
                s=hashres.get(v);
                s=s+","+hashcpu.get(v);
            }else{
                s=hashcpu.get(v);
            }
            hashres.put(v,s);
        }
        for(String v: hashres.keySet()){
            String d=hashres.get(v);
            String sub[]=d.split(",");
            res=res+",\n['"+this.segundos+"','"+v+"-"+sub[0]+"',"+Integer.parseInt(sub[1])+","+Integer.parseInt(sub[2])+"]\n";
        }
    }catch(IOException e){}
    return res;
}
```

Figure 3: Função que trata o resultado do snmpwalk

Exemplo:

- **Operação**- 0
- **hrSWRunName**- .1-init
- **hrSWRunPerfMem**- 2555
- **hrSWRunName**- 24251
- **String resultado** -[0,.1-init,2555,24251]

Este resultado é enviado de volta para o browser que após o tratamento desse *output* cria os gráficos referentes aos recursos de memória e recursos CPU.

```
<script type="text/javascript">
    var adressInput=document.getElementById('adress').value;
    var portaInput=document.getElementById('porta').value;
    var secInput=document.getElementById('ini').value;
    var info={portaInput,adressInput,secInput};
    $('#ajax').click(function(){
        var timeleft=secInput;
        var downloadTimer=setInterval(function(){
            document.getElementById("countdown").innerHTML=timeleft+" seconds remaining";
            timeleft--1;
            if(timeleft<=0){
                clearInterval(downloadTimer);
                document.getElementById("countdown").innerHTML="Finished";
            }
        },1000);
         $.ajax({
            type: "POST",
            data: info,
            dataType: "text",
            url: "get_rest",
            success: function(dataOutput){
                console.log(dataOutput);
                var queryString="?par1="+dataOutput;
                window.location.href="dashboard.html"+queryString;
            }
         });
    });
</script>
<div id="countdown"></div>
</body>
```

Figure 4: Recepção do resultado

Os gráficos foram criados através da *API* **Google Charts**.

```
function drawChart() {
  var data = new google.visualization.DataTable();
  data.addColumn('number','Segundo');
  for(var i=1;i<processofinal.length;i++){
      data.addColumn('number',processofinal[i]);
  }
  for(var i=1;i<segundosfinal.length;i++){
      var com=new Array();
      com[0]=Number(segundosfinal[i].split("'")[1]);

      for(var j=0,k=1;j<processofinal.length-1;j++,k+=processofinal.length-1){
          var val=Number(memoriafinal[j+k]);
          com.push(val);
      }
      data.addRow(com);
  }

  var options = {
      chart:{
          title: 'Memória utilizada por cada processo'
      },
      width:900,height:500
  };

  var chart = new google.charts.Line(document.getElementById('linechart_material'));

  chart.draw(data, google.charts.Line.convertOptions(options));
}
```

Figure 5: Função que desenha o gráfico

3 Resultados

Para utilização da ferramenta é necessário primeiro ligar o servidor.

Figure 6: Servidor a correr

De seguida preenchemos o formulário com a informação do *host*, porta e endereço, e o espaço de tempo em segundos no qual a monitorização deve ser efectuada.

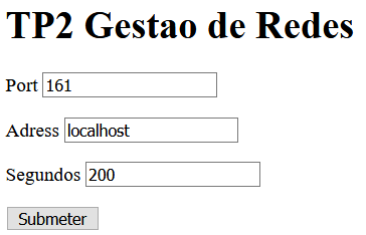


Figure 7: Interface

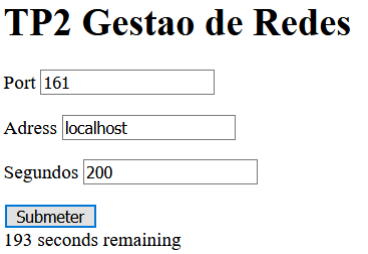


Figure 8: Countdown

Após o espaço de tempo fornecido obtemos os seguintes gráficos.

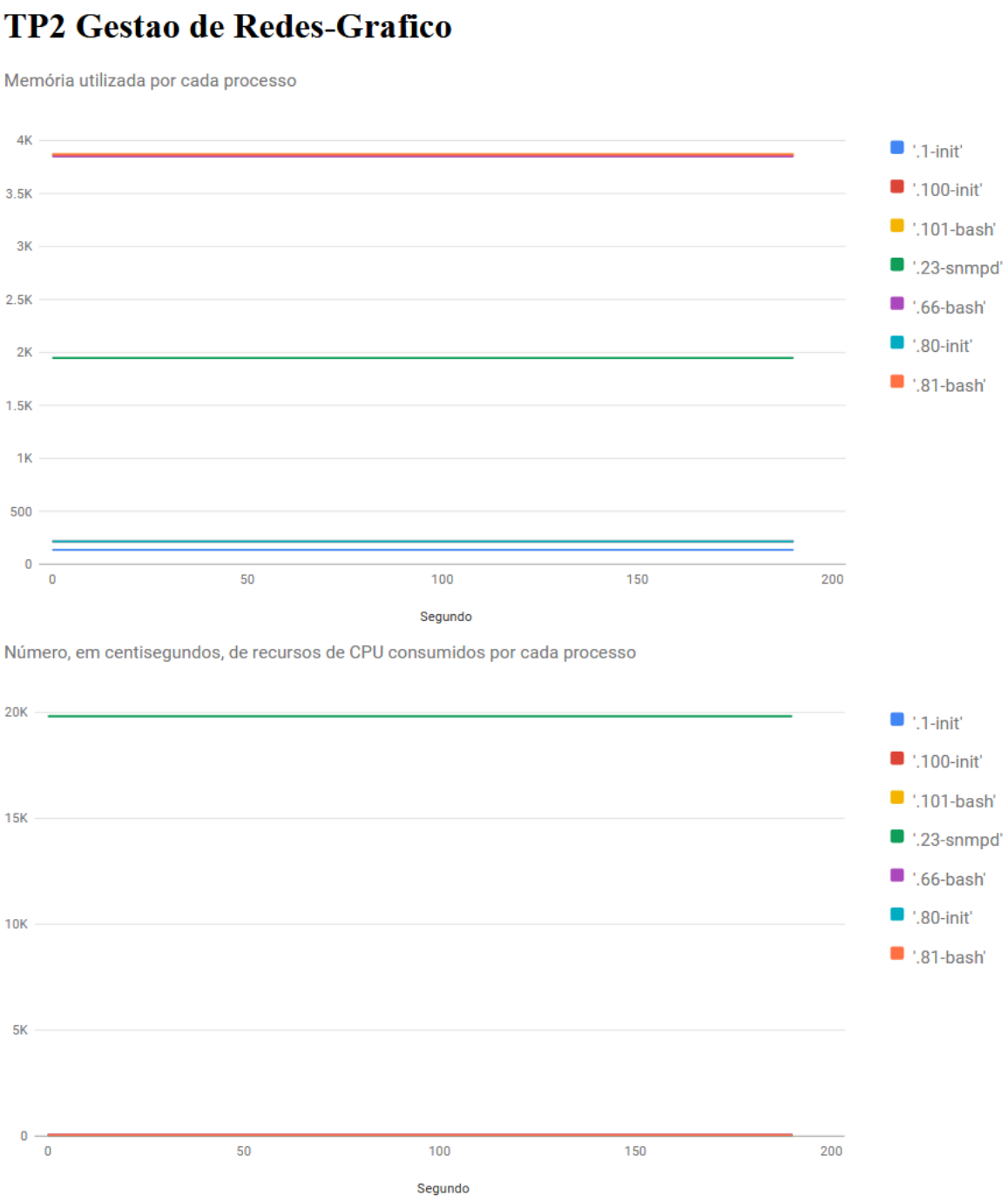


Figure 9: Gráficos obtidos

4 Conclusão

Este trabalho consegui o objectivo a que se propunha, servindo como uma introdução ao **Ajax** bem como ao **JavaScript/HTML**, resultando também num melhor entendimento do **SNMP**.

5 Outras considerações

Neste trabalho presumi que os processos a monitorizar estão sempre a correr no espaço de tempo fornecido, ou seja, os processos que estão a correr no segundo 0 da monitorização serão os mesmos que estão a correr no último segundo da monitorização.