Linguagens e Ambientes de Programação (2018/2019)

Teórica 20 (22/mai/2019)

A Internet e os seus protocolos.
A World Wide Web e o protocolo HTTP.
Web dinâmica.
Scripting do lado do cliente e modelo de eventos.
AJAX.

A Internet

A **Internet** é um sistema mundial de redes de computadores ligadas entre si. Teve origem num projeto militar dos anos 1960, mas a Internet moderna começou em meados dos anos 1980, quando as redes internas de diversas instituições académicas começaram a ser ligadas entre si. Nessa altura também apareceram as primeiras utilizações comerciais com serviços de email fornecidos por algumas companhias de comunicações.

Desenho por camadas

A Internet foi desenhada por forma a ser escalável, robusta, e obedece a um conjunto de padrões oficiais bem definidos. Quem escreve programas sobre a Internet, usa os sockets disponibilizados pelo sistema operativo e por vezes tem um conhecimento relativamente superficial das tecnologias subjacente e isso basta. Mas, para apreciar devidamente a Internet e contextualizar o desenvolvimento de programas para ela, é importante conhecer um pouco mais dessas tecnologias (que só serão verdadeiramente estudadas em pormenor na disciplina do 3º ano "Redes de Computadores").

A Internet está organizada em sete camadas (de acordo com um padrão OSI), cada uma das quais se preocupa com um diferente aspeto das comunicações:

1. **Camada física** - modems, cabos telefónicos, fibra ótica, cabos Ethernet, etc.
2. **Camada de ligação de dados** - Protocolos que permitem aos aparelhos da camada física comunicarem uns com os outros: protocolo Ethernet, protocolos wireless, protocolo PPP, etc.
3. **Camada de rede** - Protocolo IP para transmissão de pacotes de dados entre dois pontos da rede, o que inclui questões de *routing* através de nós intermédios. Esta camada implementa transmissão básica de dados, mas não é responsável pela fiabilidade da transmissão nem pelos aspetos semânticos da transmissão
4. **Camada de transporte** - Protocolos TCP, UDP e mais alguns, que adicionam diferentes graus de fiabilidade à rede e que definem diversos tipos de comunicação. O protocolo UDP suporta o envio de mensagens simples sem ser necessário estabelecer previamente uma conexão formal. O protocolo TCP suporta comunicação altamente fiável e requer o estabelecimento duma conexão formal temporária e dum diálogo entre os dois lados para a transmissão duma mensagem se poder efetuar com fiabilidade.
5. **Camada de sessão** - Permite o estabelecimento de conexões semipermanentes que são automaticamente restauradas em caso de desconexão.
6. **Camada de apresentação** - Trata de transformações básicas dos dados transmitidos e recebidos. Por exemplo: mudança de encoding dos carateres, uso de criptografia, serialização de documentos XML, etc.
7. **Camada da aplicação** - Implementa os protocolos usados pelas aplicações concretas. Exemplos de protocolos aplicacionais: Telnet (terminal virtual), SSH (terminal virtual com criptografia), FTP (acesso a ficheiros), HTTP (web), POP3/IMAP (email), BitTorrent (distribuição de ficheiros grandes), etc.

Endereços IP

O protocolo IP (na camada de rede), identifica os nós da Internet usando um endereço único chamado **endereço IP**. Na variante IPv4 o endereço é um inteiro de 32 bits. Na variante IPv6 o endereço é um inteiro de 128 bits.

Do ponto de vista do computador, um endereço IP é um inteiro. Mas quando se escreve um endereço IP para ser lido por pessoas, usa-se a base 10 e os algarismos são agrupados de forma especial que exibe a estrutura lógica do endereço (estrutura essa que não vamos discutir aqui). Eis dois exemplos de endereços IP:

```
172.16.254.1 (IPv4)
2001:db8:0:1234:0:567:8:1 (IPv6).
```

Não é prático para os humanos memorizar endereços IP e por isso foi criado um serviço de rede chamado "DNS Domain Name System" que associa **nomes de domínio** a endereços IP. Por exemplo, o domínio **ctp.di.fct.unl.pt** está associado ao endereço **193.136.122.73**.

Por convenção, o nome **localhost** representa o computador local.

Portas TCP

O protocolo TCP (na camada de transporte) permite que serviços diferentes partilhem o mesmo computador e a mesma ligação à Internet. Cada nó da rede disponibiliza um conjunto de **portas** identificadas por um inteiro de 16 bits sem sinal.

Cada serviço está associado a uma porta que pode ser escolhida livremente, embora existam muitas associações padronizadas. Por exemplo: HTTP corre na porta 80, HTTPS na porta 443, FTP na porta 21, SSH na porta 22, Telnet na porta 23, BitTorrent na porta 30301.

A World Wide Web

A **World Wide Web** (ou mais simplesmente **WWW** ou **Web**) (na camada de aplicação) resultou da seguinte ideia: Porque não criar na Internet um sistema de visualização de documentos, no qual interligamos os documentos entre si para facilitar a navegação neles?

Em 1989, foi Tim Berners-Lee que teve a ideia de aplicar à Internet a conhecida noção de **hipertexto** e começou a desenvolver o assunto. Criou a primeira versão do protocolo HTTP e da linguagem HTML. Em 1990, também foi ele que escreveu o primeiro protótipo de servidor Web e o primeiro protótipo de browser Web. Eis o resumo da história dos primeiros anos. O primeiro site web foi info.cern.ch e o primeiro endereço de página web foi http://info.cern.ch/hypertext/WWW/TheProject.html.

A Web é um dos vários serviços que correm na camada de aplicação da Internet. Usa os protocolos HTTP/HTTPS e um modelo de cliente/servidor. Os servidores Web estão ligados à Internet e correm um software específico (e.g. Apache ou IIS) que fica à espera de conexões. Os utilizadores ligam-se aos servidores usando um **Web browser**, por exemplo o Firefox, Chrome, Opera, Safari (apenas no Mac/Windows), ou Internet Explorer (apenas no Windows),

URL

Para aceder a um site Web ou a recurso particular dum site Web usando um browser, o utilizador clica num link que tem associado o URL que identifica esse site ou recurso.

URL significa **Uniform Resource Locator**. Tem a seguinte estrutura geral:

```
protocolo://domínio:porta/caminho/recurso?query#fragmento
```

Alguns exemplos de URLs:

http://en.wikipedia.org:80/wiki/Uniform_resource_locator#History

http://www.google.com/search?q=UNL

http://whatismyipaddress.com/ip/193.136.122.73

http://translate.google.com/?langpair=en|pt&text=hello

https://clip.unl.pt/utente/eu/fun%E7%E3o/doc%EAncia/unidade/actividade/inscri%E7%F5es/pautas?tipo_de_per%EDodo_lectivo=s&ano_lectivo=2016&per%ED

O protocolo HTTP

O protocolo HTTP especifica um pequeno conjunto de comandos que o browser envia para o servidor Web a solicitar alguma ação sobre um recurso bem identificado. Eis os comandos mais importantes que o browser pode enviar para o servidor:

- **GET** - Obtém uma cópia do recurso especificado.
- **HEAD** - Pede ao servidor informação sobre o recurso especificado.
- **PUT** - Atualiza um recurso no servidor, no URL especificado.
- **POST** - Atualiza um recurso no servidor, aproximadamente no URL especificado, mas é o servidor que decide qual é o URL final no qual vai ficar guardado o recurso.
- **DELETE** - Elimina o recurso especificado.

O comando mais usado é o comando **GET**. É tipicamente usado para obter o conteúdo dum ficheiro guardado no servidor ou para obter uma página HTML para apresentar no browser.

Para ver o comando **GET** a funcionar, vamos enviá-lo para o servidor usando a aplicação **telnet** (também podiamos usar a aplicação **nc**) e observar a resposta do servidor. Neste exemplo, abrimos uma conexão através da porta 80 do servidor **ctp.di.fct.unl.pt** e depois damos um comando **GET** a pedir o conteúdo da primeira aula teórica de LAP.

```
$ telnet ctp.di.fct.unl.pt 80
Trying 193.136.122.73...
Connected to di73.di.fct.unl.pt.
Escape character is '^]'.

GET /miei/lap/teoricas/01.html

<body text="#00000" bgcolor="#E0F0E0" link="#0000EE" vlink="#551A8B" #FF0000">
<HR><HR><H1>Linguagens e Ambientes de Programação (2018/2019)</H1>
<HR><HR><H2>Teórica 01 (04/Mar/2019)</H2>
<p>Apresentação da disciplina.
<p>Discussão introdutória sobre alguns aspetos importantes da temática das Linguagens de Programação.
<HR>
...
... muitas linhas omitidas ...
...

Connection closed by foreign host.
```

Já agora, testemos também o comando **HEAD** sobre a mesma página:

```
$ telnet ctp.di.fct.unl.pt 80
Trying 193.136.122.73...
Connected to di73.di.fct.unl.pt.
Escape character is '^]'.

HEAD /miei/lap/teoricas/01.html HTTP/1.0

HTTP/1.1 200 OK
Date: Thu, 04 Jun 2019 21:38:10 GMT
Server: Apache/2.2.16 (Debian)
Last-Modified: Sat, 14 Mar 2019 12:54:29 GMT
ETag: "3ba5f-39bf-5113f1e172b40"
Accept-Ranges: bytes
Content-Length: 14783
Vary: Accept-Encoding
Connection: close
Content-Type: text/html

Connection closed by foreign host.
```

Neste exemplo, comunicámos com o servidor usando a aplicação **telnet**. Mas em que circunstâncias é que o browser envia realmente comandos **GET** para o servidor? Eis dois exemplos:

Exemplo 1

O primeiro exemplo é trivial e consiste num simples link que refere um endereço da Google. Ao clicar nele, fazemos o browser enviar um comando **GET** ao servidor da Google. O browser recebe do servidor a página HTML correspondente ao URL e mostra-a.

http://www.google.com/search?q=Internet

Código:

```
<a href="http://www.google.com/search?q=Internet">http://www.google.com/search?q=Internet</a>
```

Exemplo 2

No exemplo 1 interrogámos o servidor da Google com a query fixa "Internet". Suponhamos que agora queremos interrogar o servidor usando uma query variável, indicada pelo utilizador. A solução passa por usar um formulário HTML com um atributo "ACTION" e com o botão "submit", o que para nós é novidade pois não se falou disto na aula em que apresentaram os formulários HTML.

Google query: Internet Search

Código:

```
<FORM NAME="form1" ACTION="http://www.google.com/search">
    Google query:
    <INPUT TYPE="text" NAME="q" VALUE="Internet">
    <INPUT TYPE="submit" VALUE="Search">
</FORM>
```

Neste formulário, quase tudo o que lá se encontra tem a sua razão de ser:

- No atributo "ACTION" especifica-se a parte inicial (prefixo) do URL.
- A caixa de input tem o nome "q", que vai fazer parte do URL gerado.
- Finalmente, existe um botão de tipo "submit". Premir esse botão faz o browser construir um URL completo e enviá-lo para o servidor num comando **GET**.

Scripting do lado do servidor - Web dinâmica

Em 1991, a primeira versão do protocolo HTTP suportava apenas o comando **GET** e a resposta do servidor tinha de ser obrigatoriamente uma página HTML fixa. Não havia a possibilidade de gerar páginas dinamicamente do lado do servidor. Portanto, nessa altura, um site Web tinha associado uma simples coleção de páginas HTML fixas, geralmente organizadas numa árvore de ficheiros e diretorias.

A **Web dinâmica** começou em 1993, quando foram introduzidos os formulários HTML (do lado do cliente) e o Common Gateway Interface (CGI) (do lado do servidor). Do lado do browser (cliente), usando formulários HTML, passou a ser possível efetuar pedidos com argumentos variáveis (como no último exemplo da secção anterior). Do lado do servidor, passou a ser possível correr scripts para gerar dinamicamente páginas HTML.

As linguagens Perl e C começaram por ser as mais usadas na escrita de scripts do lado do servidor. Mas logo foram desenvolvidas linguagens mais especializadas. Uma delas, o PHP, apareceu em 1995 e tornou-se rapidamente muito popular - atualmente continua a ser a linguagem mais usada do lado do servidor. Eis outras linguagens que se usam atualmente do lado do servidor: Java, ASP, Python, Ruby, Scala, TCL, JavaScript, etc.

Agora um detalhe importante dos servidores tradicionais: por cada pedido recebido, cria-se um processo independente para correr o script que vai gerar a página de resposta. Portanto, o servidor pode ficar congestionado se forem recebidos muitos pedidos num curto espaço de tempo. (Em todo o caso, demasiados acessos concorrentes à base de dados costuma ser o fator mais determinante de possíveis congestionamentos.)

Os exemplos 1 e 2, atrás, são representativos das funcionalidades existentes no início de 1996. Eram suportados formulários HTML com o atributo "ACTION", os servidores eram capazes de gerar páginas dinâmicas. Mas as páginas não eram interativas do lado do browser. Neste modelo, por cada pedido feito pelo browser, o servidor envia uma página completa que substitui a anterior (e o utilizador vê a página a ser completamente redesenhada).

Nessa altura ainda não tinha sido inventado o JavaScript, nem introduzido o atributo "ONCLICK" nos formulários HTML...

Scripting do lado do cliente - JavaScript

Em 1996 foram testadas novas funcionalidades que ficaram padronizadas em 1997 sob o nome de **HTML 4**. Por exemplo, foram introduzidas frames e style sheets (CSS). Mas a adição mais revolucionária foi a introdução de scripting do lado do cliente. Com scripts a correr do lado do cliente, passou a ser possível ter páginas Web realmente interativas. A interatividade proporcionada pelos formulários era muito pobre.

A primeira linguagem de scripting usada do lado do cliente foi o JavaScript. Continua a ser a mais usada nesse contexto.

Dois atributos importantes que foram adicionados ao HTML, ligados ao scripting: "ONCLICK", que pode ser usado na maioria dos elementos HTML, mas que é mais usado nos botões, para invocar uma função JavaScript em resposta a um clique de rato; "ONLOAD", que é usado no elemento "BODY" para executar uma função JavaScript logo que o carregamento da página termine.

Com o JavaScript passou a ser possível, por exemplo:

- Validar o formato dos dados dos formulários do lado do cliente e dar feedback imediato em caso de erro.
- Inserir dinamicamente novo texto numa página HTML, sem passar pelo servidor.
- Reagir imediatamente a eventos gerados pelo utilizador através do teclado e rato.

Atualmente existe a tendência para implementar cada vez mais funcionalidades do lado do cliente e menos do lado do servidor. Do ponto de vista do utilizador a experiência de utilização torna-se mais rica e mais rápida (parecendo que se está a usar uma aplicação local). Contudo, há limites naturais para esta tendência. Por exemplo, as bases de dados, que contêm os dados consultados, são centralizadas e residem do lado do servidor.

Modelo de eventos

Como é comum nas interfaces gráficas interativas, os scripts que correm no browser usam um modelo de programação baseado em eventos. Os eventos normalmente são gerados por ações do utilizador, através do rato ou do teclado. Os eventos são tratados de forma assíncrona.

O uso de lógica assíncrona tem consequências dramáticas no estilo de programação! Nos programas clássicos, o programa consegue controlar todo o fluxo de execução de forma estrita. Na programação assíncrona, escreve-se código para reagir a eventos e não há possibilidade de controlar o fluxo de execução geral.

Quando um programa JavaScript corre num browser, há uma thread principal que executa o código do programa e se chama a **thread dos eventos**. A thread dos eventos já está implementada (não pode ser alterada). Executa continuamente um ciclo que retira e processa os eventos guardados numa fila de espera. Por cada evento, a thread invoca a função de tratamento adequada, previamente associada a esse tipo de evento. Os eventos são tratados sequencialmente e enquanto não terminar o tratamento dum evento não se inicia o tratamento do seguinte. Por isso é importante que o tratamento de cada evento seja rápido - caso contrário, a página Web terá bloqueios desagradáveis. (Para além da thread dos eventos, há algumas threads auxiliares usadas na geração de eventos, mas elas trabalham discretamente nos bastidores, e o programador não pensa nelas.)

Eis a lista completa de eventos do DOM para os quais se podem instalar tratadores de eventos numa página HTML. Os tratadores de eventos podem ser instalados nos atributos ONCLICK e ONLOAD, ou usando a função addEventListener em código JavaScript. Para criar um timer e especificar qual a função tratadora de eventos de tempo, usa-se a função setInterval.

AJAX

Sem usar AJAX, um pedido do cliente (browser) ao servidor implica apresentar uma página completa nova. Foi o que vimos nos dois exemplos mais atrás.

AJAX é um mecanismo que permite ao cliente obter informação do servidor de forma completamente flexível. O cliente faz o pedido e depois pode usar a resposta (assíncrona) como entender, por exemplo para atualizar uma porção da página corrente, mostrar a resposta num alerta, ou outra coisa qualquer.

O mecanismo é fácil de usar e envolve uma API chamada XMLHttpRequest, que está disponível para ser usada em JavaScript em todos os browsers modernos.

Os conceitos subjacentes ao AJAX foram inventados pela Microsoft por volta de 1998. A Google começou a usar AJAX de forma massiva nos seus sites em 2004 começando pelo GMail e Google Maps.

Pedido assíncrono

Eis um exemplo, onde se troca o texto "AJAX lava mais branco", pelo conteúdo do ficheiro "ajax.txt", que se encontra no servidor. Experimente clicar:

Eis o código do exemplo. A função `update` faz o seguinte: (1) cria um pedido XMLHttpRequest vazio, (2) indica como será tratada a futura resposta do servidor, (3) preenche o pedido, (4) envia o pedido. Sabemos que a resposta já está disponível quando **`xhttp.readyState == 4`** e sabemos que o operação teve sucesso se adicionalmente **`xhttp.status == 200`**.

```
<SCRIPT TYPE="text/javascript">
function update(elementname, filenameURL) {
    var xhttp = new XMLHttpRequest();       // cria pedido vazio
    xhttp.onreadystatechange = function() { // especifica tratamento da resposta
        if( xhttp.readyState == 4 )
            if( xhttp.status == 200 ) {
                var el = document.getElementById(elementname);
                el.innerHTML = xhttp.responseText;
            }
            else alert("Error");
    };
    xhttp.open("GET", filenameURL, true);    // preenche o pedido
    xhttp.send();                            // envia o pedido para o servidor
}
</SCRIPT>

<H1>AJAX</H1>
<INPUT TYPE="button" VALUE="Click Me" ONCLICK="update('ajax', 'ajax.txt')">
<SPAN id="ajax">AJAX lava mais branco.</SPAN>
```

Atenção, que por questões de segurança, a comunicação AJAX só funciona se o cliente e o servidor estiverem no mesmo domínio. O exemplo anterior funciona porque esta é uma página situada em **ctp.di.fct.unl.pt** a comunicar com um servidor também situado em **ctp.di.fct.unl.pt**.

Pedido síncrono

O pedido XMLHttpRequest anterior foi tratado de forma assíncrona que é o que geralmente se pretende. Mas se, numa situação excecional, não nos importarmos de fazer o pedido de forma síncrona, o que implica bloquear o programa por momentos à espera da resposta, o código da função anterior fica bastante simplificado. A seguinte função permite obter o conteúdo dum ficheiro situado no servidor. Neste código não é possível fazer tratamento de erros, pelo que o programa bloqueia simplesmente se o ficheiro não existir.

```
function getFile(filenameURL) {
    var xhttp = new XMLHttpRequest();       // cria pedido vazio
    xhttp.open("GET", filenameURL, false);  // preenche o pedido
    try {
        xhttp.send();                      // envia o pedido para o servidor
        return xhttp.responseText;
    }
    catch(err) {
        return null;
    }
}
```

Acesso local

Normalmente, cada pedido XMLHttpRequest é feito usando o protocolo HTTP. Mas muitos browser generalizam o mecanismo e também permitem pedidos XMLHttpRequest para obter recursos situados na mesma máquina do browser, ou seja recursos locais com URLs da forma "file://..." ou "localdir/localfile". Esta funcionalidade é útil para testar os programas com ficheiros locais.

O browser Firefox oferece esta funcionalidade diretamente. No caso do Chrome, esta funcionalidade precisa de ser ativada invocando o browser na linha de comando da seguinte forma:

```
chrome --allow-file-access-from-files
```

#80