Laboratórios de Informática 2014/2015

DETI - Universidade de Aveiro

Docentes
João Paulo Barraca <jpbarraca@ua.pt>
Diogo Gomes <dgomes@ua.pt>
João Manuel Rodrigues <jmr@ua.pt>
Mário Antunes <mario.antunes@ua.pt>

TEMA 17

Aplicações e Serviços Web

Objetivos:
- Servidores Web
- Serviços Web

17.1 Introdução

A World Wide Web (WWW) ou *Web* como é hoje popularmente conhecida, teve a sua génese em 1990 no Conseil Européen pour la Recherche Nucléaire (CERN) pelas mãos de Tim Berners-Lee. No seu inicio a *Web* pretendia ser um sistema de hiper-texto que permitisse aos cientistas rapidamente seguir as referencias num documento, evitando o processo tedioso de apontar referencias e pesquisar as mesmas. A *Web* pretendia na altura ser um repositório de informação estruturado em torno de um grafo (daí a *Web*) em que o utilizador pudesse seguir qualquer percurso entre os diversos documentos interligados pelas suas referencias.

A *Web* assenta em 3 tecnologias, já tratadas nesta disciplina:

· Um sistema global de identificadores únicos/uniformes (URL, URI)

· Uma linguagem de representação de informação (HTML)

· Um protocolo de comunicação cliente/servidor (HTTP)

Com base nestas tecnologias, podemos não só transferir ficheiros entre um servidor um cliente equipado com um *Web browser*, como também podemos construir documentos de forma dinâmica a partir de dados enviados. Esta secção irá te guiar no desenvolvimento de uma aplicação *Web* com base na linguagem de programação *Python* e no formato de documentos JSON.

17.2 Servidores *Web*

17.2.1 Common Gateway Interface

Um servidor *Web* é uma aplicação de software que permite a comunicação entre dois equipamentos através do protocolo HTTP. A função inicial de um servidor *Web* era a de fornecer documentos armazenados em disco em formato HyperText Markup Language (HTML)[1] a um cliente remoto equipado com um *Web* browser. Esta simples tarefa desde cedo demonstrou-se limitativa, uma vez que vários casos existiam em que era necessário condicionar os dados nos documentos HTML a vários fatores tais como identidade do utilizador, a sua localização a sua língua nativa, etc.

É desta forma que surge o conceito de Common Gateway Interface (CGI). A CGI permite ao servidor interagir com um programa externo capaz de produzir dinamicamente conteúdos de qualquer formato. O *standard* CGI define um conjunto de parâmetros que são passados do servidor *Web* para a aplicação externa (denominada de forma abusiva de CGI), assim como o formato que essa mesma aplicação deve de obedecer por forma ao servidor *Web* re-interpretar o seu *output* antes de enviar ao *Web* browser do cliente.

É possível criar programas CGI em qualquer linguagem, inclusive de *scripting* como o *Bash*.

Exercício 17.1

No servidor **xcoa.av.it.pt**, no diretório **public_html**, crie um novo diretório com o nome *cgi-bin*. Dentro desse diretório crie um novo ficheiro com extensão *.sh* e com o seguinte conteúdo:

```
#!/bin/bash

echo Content-type: text/plain
echo ""

echo "Hello World"
```

Altere as permissões do ficheiro para o mesmo ser executável (ex.: **chmod +x test.sh**)
Execute na linha de comandos o ficheiro
Agora no seu navegador *Web* aceda ao ficheiro que acabou de criar.
Altere o ficheiro para mostrar outras *Strings*.

Do exercício anterior é importante reter a necessidade do programa imprimir um cabeçalho com informação do tipo de ficheiro que será criado dinamicamente. Através da interface CGI é possível não só criar ficheiros de texto (*plain*, HTML, JS, etc) como também ficheiros binários (imagens, vídeos, etc).

Exercício 17.2

Altere o ficheiro anterior adicionando o comando **env**

```
#!/bin/bash

echo Content-type: text/plain
echo ""

echo "Hello World"

env
```

No seu navegador *Web* aceda ao ficheiro.

Deste exercício podemos retirar ensinamentos sobre o tipo de variáveis que o servidor *Web* envia através da interface CGI para o programa externo.

17.2.2 Servidores Aplicacionais

Na secção anterior abordámos a interface CGI que se popularizou nos finais do século passado como a ferramenta para desenvolver conteúdos dinâmicos para a *Web*. Servidores como o *Apache* e o *Microsoft IIS* permitem a execução de programas externos usando a interface CGI, e ainda hoje diversos sites *Web* fazem uso desta tecnologia. No entanto, o crescente dinamismo da *Web*, e a necessidade de criação de aplicações *Web* que requerem múltiplas interações com o utilizador tornam a interface CGI extremamente ineficiente e até mesmo insegura (já que os programas correm com as mesmas permissões do servidor *Web*).

Servidores Aplicacionais como o *Glassfish*, *JBoss*, *.NET* permitem ao programador ultrapassar muitas destas dificuldades ao incorporarem em si próprios código desenvolvido por programadores externos. Não estamos mais na situação de o servidor executar um programa externo, mas na de o meu programa incluir o servidor *Web*.

Neste capitulo, vamos abordar um servidor aplicacional *Python* especifico de seu nome *CherryPy*.

O *CherryPy* é um servidor aplicacional maduro usado tanto para pequenas aplicações como para grandes (ex.: *Hulu*, *Netflix*). O *CherryPy* pode ser usado sozinho (*stand-alone*) ou através de um servidor *Web* tradicional via interfaces Web Server Gateway Interface (WSGI). No âmbito desta disciplina, vamos apenas usar o *CherryPy* como servidor *stand-alone*.

Para instalar o *CherryPy* pode recorrer ao gestor de pacotes da sua distribuição Linux ou ao **`pip`**.

No ubuntu pode executar:

```
sudo apt-get install python-CherryPy
```

Em alternativa pode executar:

```
pip install --user CherryPy
```

O *CherryPy* é composto por 8 módulos:

- **`CherryPy.engine`** Controla início e o fim dos processos assim como o processamento de eventos.
- **`CherryPy.server`** Configura e controla a WSGI ou servidor HTTP.
- **`CherryPy.tools`** Conjunto de ferramentas ortogonais para processamento de um pedido HTTP.

- **`CherryPy.dispatch`** Conjunto de *dispatchers* que permitem controlar o roteamento de pedidos para os *handlers*.

- **`CherryPy.config`** Determina o comportamento da aplicação

- **`CherryPy.tree`** A árvore de objetos percorrida pela maioria dos *dispatchers*

- **`CherryPy.request`** O objeto que representa o pedido HTTP

- **`CherryPy.response`** O objeto que representa a resposta HTTP

Comecemos por criar uma aplicação semelhante ao *script* CGI anterior:

Exercício 17.3

Crie no seu próprio computador o seguinte ficheiro

```
import cherrypy

class HelloWorld(object):
    def index(self):
        return "Hello World!"
    index.exposed = True

cherrypy.tree.mount(HelloWorld(), "/")
cherrypy.server.start()
```

No seu *Web* browser aceda à aplicação usando o endereço `http://localhost:8080/`

Revendo cada linha do exercício anterior, começamos por identificar a importação do modulo *CherryPy*. De seguida temos a declaração de uma classe de seu nome *HelloWorld*. Esta classe é composta por um método chamado *index* que retorna uma *String*. Uma linha final na classe *HelloWorld* determina que o método *index* deverá ser exposto ao cliente *Web*. Uma alternativa a esta linha seria um decorador da função index: ***`@CherryPy.expose`***.
Por fim, o modulo *CherryPy* inicia um servidor a partir da class *HelloWorld*.

Quando um cliente *Web* acede ao servidor aplicacional *CherryPy*, este procura por uma *class*/método que possa atender ao pedido do cliente. Neste exemplo básico, existe apenas uma classe e método que irá servir ao cliente a *String* "Hello World".

O *CherryPy* disponibiliza através do **`CherryPy.request.headers`** as variaveis enviadas pelo cliente ao servidor.

Exercício 17.4

Altere o programa anterior para mostrar o nome do servidor ao qual o cliente fez um pedido HTTP

```
...
        host = cherrypy.request.headers['Host']
        return "You have successfully reached " + host
```

Qualquer objeto associado ao objeto raiz é acessível através do sistema interno de mapeamento *URL*-para-objeto. No entanto, tal não significa que um objeto esteja exposto na *Web*. É necessário que o objeto seja exposto explicitamente como visto anteriormente.

Exercício 17.5

Crie um novo programa com o seguinte conteúdo

```
import cherrypy

class Node(object):
    @cherrypy.expose
    def index(self):
        return "Eu sou um objecto Folha"

class Root(object):
    def __init__(self):
        self.node = Node()

    @cherrypy.expose
    def index(self):
        return "Eu sou o objecto Raiz"

    @cherrypy.expose
    def page(self):
        return "Eu sou um método da Raiz"

if __name__ == '__main__':
    cherrypy.tree.mount(Root(), "/")
    cherrypy.server.start()
```

Aceda a cada um dos recursos a partir do seu navegador *Web*:
/
/page
/node/

Acrescente agora uma nova **`class`** de nome **`HTMLDocument`** que retorne o conteúdo de ficheiro HTML lido do disco.

Mais uma vez, é importante reter alguns aspetos do exercício anterior. O método **`index`** serve os conteúdos na raiz do URL (/) e cada método tem que ser individualmente exposto.

17.2.3 Formulário HTML

O protocolo HTTP faz uso de dois métodos principais para a troca de informação entre cliente e servidor (**`GET`** e **`POST`**). O método **`GET`** já foi extensivamente usado nos capítulos e secções anteriores, e permite ao cliente *Web* solicitar um documento que resida no

servidor *Web*. Por sua vez o método **POST** permite enviar informação do cliente *Web* para o servidor *Web*. O envio de informação pode ser feito através de um formulário HTML.

Exercício 17.6

Crie uma página HTML com o código para formulário seguinte:

```
<form action="actions/doLogin" method="post">

    <p>Username</p>
    <input type="text" name="username" value="" size="15" maxlength="40"/>

    <p>Password</p>
    <input type="password" name="password" value="" size="10" maxlength="40"/>

    <p><input type="submit" value="Login"/></p>
    <p><input type="reset" value="Clear"/></p>
</form>
```

Não esquecer de completar a página com o código HTML apropriado.
Crie um novo método na sua aplicação:

```
    import os.path
    from cherrypy.lib.static import serve_file

    @cherrypy.expose
    def form(self):
        return serve_file(os.path.join(os.path.dirname(os.path.abspath(__file__)),
                          "formulario.html"),
                          content_type='text/html')
```

No exercício anterior permitimos ao nosso servidor aplicacional servir uma página HTML com o conteúdo de um formulário usando do método **serve_file**. No entanto, a

submissão do formulário de *login* necessita ainda da implementação de mais um método.

Exercício 17.7

Crie um novo objeto (*actions*) e método na sua aplicação, não se esqueça de associar o novo objeto à Raiz.

```
class Actions(object):
    @cherrypy.expose
    def doLogin(self, username=None, password=None):
        return "TODO: verificar as credenciais do utilizador " + username
```

Abra o formulário através do endereço `http://localhost:8080/form/` preencha o mesmo e submeta.

Importante referir que os argumentos *username* e *password* chegam até à nossa aplicação *Web* através de um mapeamento direto do nome das variáveis do formulário HTML para os argumentos do nosso método **`doLogin`** (também mapeado diretamente).

17.3 Serviços Web

Na secção anterior abordamos como um cliente *Web* pode interagir com uma aplicação *Web* alojada no servidor. Nesta secção vamos abordar como duas aplicações podem interagir entre si através do protocolo HTTP.

O primeiro desafio que se coloca é como escrever uma aplicação *Python* capaz de aceder a uma página *Web* via o protocolo HTTP. Para tal vamos fazer uso da biblioteca **`urllib2`** cuja documentação completa encontra-se disponível em `http://docs.python.org/2/library/urllib2.html`.

A biblioteca **`urllib2`** permite-nos aceder a uma página *Web* de forma muito semelhante à que utilizamos em *Python* para aceder a um ficheiro/documento.

```
import urllib2

f = urllib2.urlopen('http://www.python.org')
```

Exercício 17.8

Faça um pedido **GET** ao endereço `http://www.ua.pt`
A sua aplicação deverá ler por completo o conteúdo da página da Universidade de Aveiro.

O uso directo do método **urlopen** permite-nos obter o conteúdo de um recurso HTTP através do método **GET**, no entanto se pretendermos enviar algum conteúdo para uma aplicação *Web*, é necessário usar o método **POST** como visto anteriormente.

O método **POST** possibilita o envio de informação codificada no corpo do pedido **POST**. A codificação dos dados segue um de dois *standards* definidos pelo World Wide Web Consortium (W3C), o **application/x-www-form-urlencoded** e o **multipart/form-data**. O primeiro formato é o usado por omissão e permite o envio de informação trivial como variáveis não muito extensas. O segundo é apropriado para o envio de variáveis mais extensas assim como de ficheiros.

O *Python* possui na biblioteca **urllib** o método **urlencode** que permite converter de forma fácil um dicionário *Python* numa *String* codificada em **application/x-www-form-urlencoded**.

```
import urllib

data = urllib.urlencode({"nome": "Ana", "idade": 20})
```

Munidos da *String* codificada de dados podemos construir um objecto **Request** que é usado como argumento de **urlopen**.

```
req = urllib2.Request(url)
req.add_data(data)

f = urllib2.urlopen(req)
```

Exercício 17.9

Fazendo uso da aplicação *Web* desenvolvida anteriormente implemente uma aplicação capaz de fazer login

Os exercícios anteriores demonstraram como criar uma aplicação *Web* capaz de interagir com um cliente (*Web browser*), mas a sua utilidade pode ser transposta para a comunicação entre duas aplicações.

Exercício 17.10

O *Google* dispõe de uma Application Programming Interface (API) que permite converter um endereço em coordenadas (latitude e longitude). Neste exercício deverá usar a API do google com base no seguinte código para encontrar as coordenadas de qualquer cidade passada ao seu programa através de um argumento de linha de comando.

```
serviceurl = 'http://maps.googleapis.com/maps/api/geocode/json?'

url = serviceurl + urllib.urlencode({'sensor':'false', 'address': address})
f = urllib.urlopen(url)
```

17.4 Para Aprofundar

Exercício 17.11

Recuperando o exercício de aprofundamento do guião anterior.
Construa uma aplicação *Web* que aceda ao ficheiro disponível em `http://www.ipma.pt/resources.www/internal.user/pw_hh_pt.xml`, contendo os dados meteorológicos observados nas principais cidades Portuguesas.
A sua aplicação deverá receber o nome da cidade por método `POST` pelo que deve construir um formulário com a lista de cidades possíveis usando uma *dropbox*.
À submissão do formulário deverá seguir-se a impressão dos dados da cidade indicada no formulário.

Glossário

API Application Programming Interface

CERN Conseil Européen pour la Recherche Nucléaire

CGI Common Gateway Interface

HTML	HyperText Markup Language
HTTP	HyperText Transfer Protocol
JS	JavaScript
URL	Uniform Resource Locator
URI	Uniform Resource Identifier
W3C	World Wide Web Consortium
WWW	World Wide Web
WSGI	Web Server Gateway Interface

Referências

[1] W3C. (1999). Html 4.01 specification, endereço: `http://www.w3.org/TR/1999/REC-html401-19991224/`.