第七讲 Python Web编程

清华大学计算机系

什么是Web编程

- 客户端、服务器编程 **Client/Server (CS)**
- 浏览器、服务器编程 **Browser/Server (BS)**

B/S结构 (Browser/Server)	C/S结构 (Client/Server)
浏览器端： HTML/CSS/JavaScript/ VBScript 服务器端： ASP(.NET)/PHP/JSP	C/S结构： VB/VC/VC#/Delphi/ Java/.Net系列

数据库支持： SQL Server/Oracle/Sybase/MySQL/Informix

两大语法体系
Basic系： VB/VBScript/ASP(VBScript)/VB.Net/VBA
C系： Java/JavaScript/C++/C#/PHP/JSP/ASP(JScript)

BS、CS比较

- 客户端要求
  C/S要求较高、B/S要求较低。
- 软件安装
  C/S每一个客户端都必须安装和配置软件
  BS 使用浏览器访问，不用安装任何专门的软件,易推广。
- 软件升级和维护
  C/S每一个客户端都要升级程序。BS客户端不必安装及维护
- 安全性
  - ✓ C/S面向相对固定的用户群，对信息安全的控制能力很强。
  - ✓ B/S架构管理模式是基于浏览器完成的，对安全性产生了很大的隐患。

BS、CS比较

- 响应速度
  C/S快，基本没有延迟。B/S慢，有延迟
- 交互性
  C/S交互性强，客户端有一套完整的应用程序
  B/S有一定的交互能力
- 易用性
  - ✓ C/S架构的管理模式在易用性方面要远远优于B/S架构。操作方便，直观、简单、比较人性化。
  - ✓ B/S易用性较差。兼容性问题。

Web介绍

- Web全称为World Wide Web
- Web是Internet提供的一种服务
- Web是存储在全世界Internet计算机中、数量巨大的文档的集合
- Web是一种超文本信息系统
- Web与平台无关
- Web是分布式的 、具有新闻性、动态的、交互的

Web工作原理

Web服务器向浏览器提供服务的过程：

- **用户提交请求**：在浏览器中指定一个URL(Uniform Resource Locator，统一资源定位器），浏览器便向该URL所指向的Web服务器发出请求。
- **服务器解析**：Web服务器（也称为HTTP服务器）接到浏览器的请求后，把URL转换成页面所在**服务器的文件路径名**。
- **服务器响应**：如果URL指向的是普通的HTML（Hypertext Markup Language，超文本标记语言）文档，Web服务器直接把它传送给浏览器。
- **客户端解析**：浏览器解析服务器的返回结果，显示给用户

静态页面和动态页面

- 早期的Web页面是静态的，用html代码书写。
- 随着ASP和java技术的发展产生了动态网页。
- 服务器可以访问数据库，存取服务器的有关资源。
- 动态网页的实现一般采用客户端编程和服务器端编程两种设计方法。

- 客户端编程就是从浏览器下载服务器的程序来执行相关动态工作。
- 服务端编程就是将程序员编写的代码保存在服务器中。代码在服务器端执行，把数据（HTML/XML等）传回浏览器。

静态页面

1 静态网页

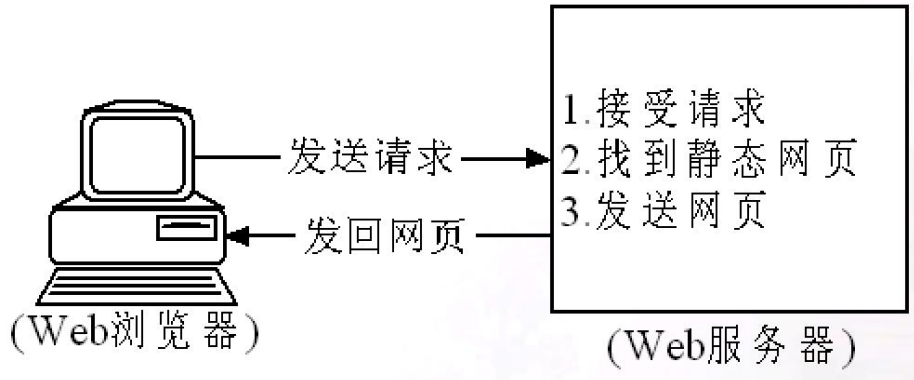
- ✓静态网页是标准的HTML文件，其文件扩展名是.htm或.html，它可以包含HTML标记、Java小程序、客户端脚本以及客户端ActiveX控件
- ✓但不包含任何服务器端脚本
- ✓静态网页的工作原理：
  - 用户提交请求
  - 服务器解析：服务器的查找文件路径名
  - 服务器响应：服务器返回静态文件
  - 客户端解析

动态页面

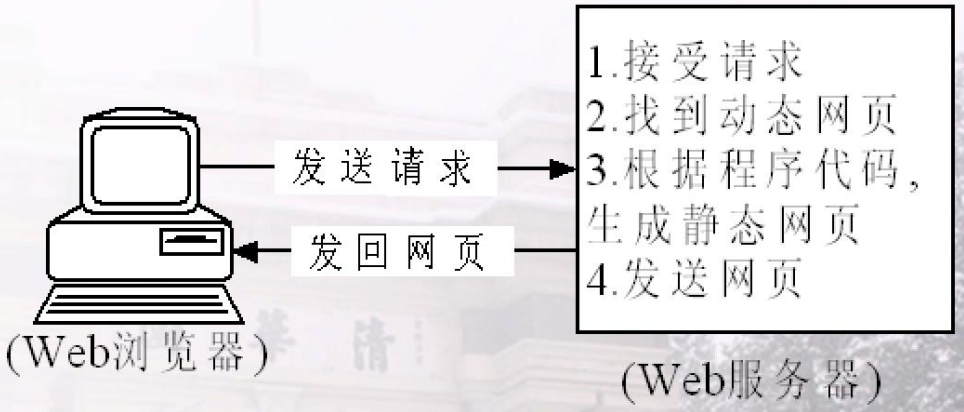
2 动态网页

- 所谓动态网页，就是该网页文件含有服务器端脚本，这种网页的后缀一般根据不同的程序设计语言来定
  - ✓ 如ASP文件的后缀为.asp
  - ✓ Java编程. jsp
  - ✓ Python编程 .py
- 动态网页的工作原理与静态网页有很大的不同。
  - 用户提交请求
  - 服务器解析：服务器查找文件路径名
  - 服务器响应：服务器执行服务器语言，返回结果
  - 客户端解析

动态和静态页面区别



静态网页工作原理



动态网页工作原理

客户端编程

- 轻载语言
  - ✓**HTML（Hypertext Markup Language，超文本标记语言）**
  - ✓**CSS（Cascading Style Sheets，层叠样式表单）**
  - ✓脚本语言**JavaScript、VBScript。**

HTML概述

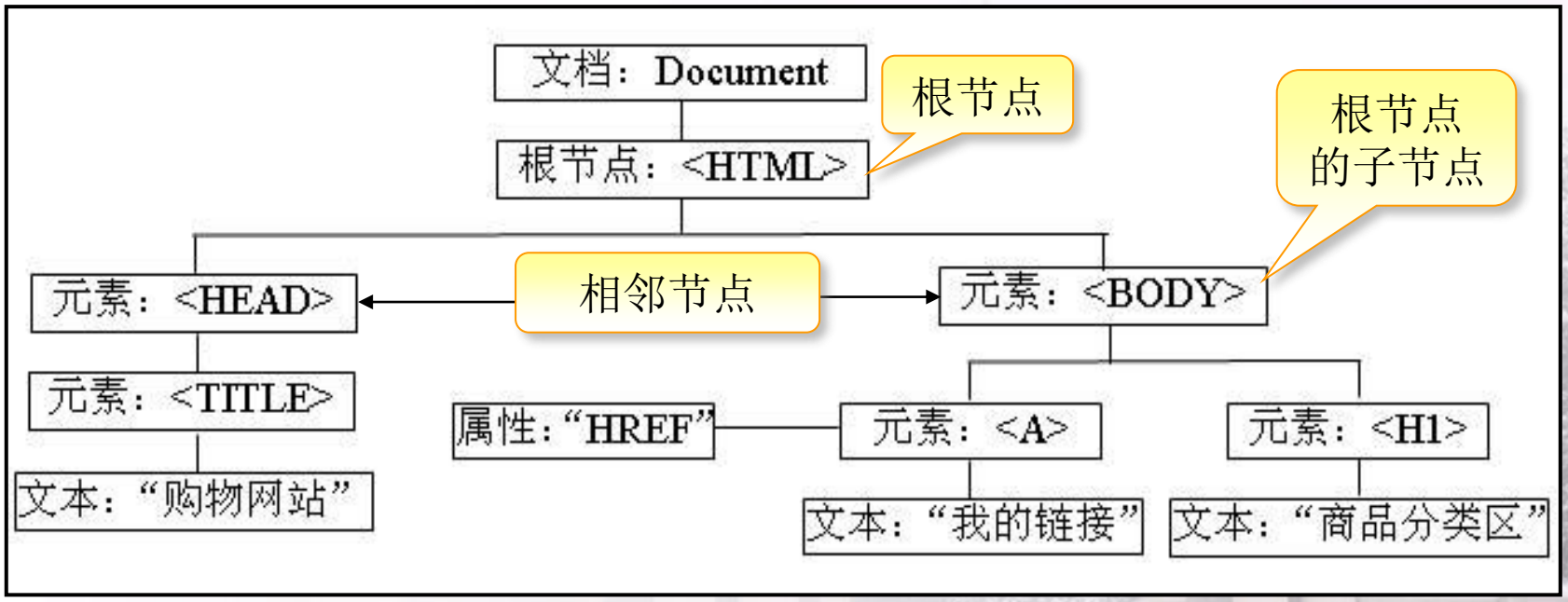
- **Hyper Text Markup Language超文本标记语言，是一种描述文档结构的标注语言。**
- **“.html”或者“.htm”作为后缀。**
- **当用户浏览WWW上的信息时，浏览器会自动解释这些标记的含义，并按照一定的格式在屏幕上显示这些被标记的文件。**
- **HTML的优点是跨平台性。**

HTML网页基本结构

- HTML文件是标准的ASCII文件，像是加入了许多被称为链接签（tag）的特殊字符串的普通文本文件。
- 从结构上，HTML文件由元素（element）组成。
- 组成HTML文件的元素有许多种，用于组织文件的内容和指导文件的输出格式
- 绝大多数元素是“容器”， 即它有起始标记和结尾标记（start tag、end tag），中间部分是元素体。

```
<HTML>
        <HEAD>
                <TITLE></TITLE>
        </HEAD>
        <BODY></BODY>
</HTML>
```

HTML文档的树状结构



HEAD头元素

head中的<title>和<meta>

<TITLE>标记用于网页命名，显示在浏览器的标题栏中。

<meta>设定与网页内容相关的各种信息，具体由其两个参数http-equiv和content决定。

http-equiv指定浏览器的编码种类；

charset=gb2312表示用GB码显示。

```
<HTML>
<HEAD>
 <meta http-equiv="Content-Type"  content="text/html; charset=gb2312">
<TITLE>清华大学计算机系</TITLE>
</HEAD>
<BODY   bgcolor= yellow>
<P>这是一HTML的测试文件</P>
</BODY>
</HTML>
```

HTML的常用标记

- HTML的常用标记有一些共同特点：都放在BODY标记里面。
- 常用的标记有字体标记、图片标记、超级链接、列表、表格和表单等
- 注意：
  - ✓（1）HTML文件不区分大小写；
  - ✓（2）HTML文件可以双击执行

文本标记

- 段落标记 ** … **
  - ✓** ** 表示标记段落。
  - ✓** **标记常带参数**align**，用于控制对齐方式。
  - ✓可取**left**，**center**，**right**。
- 标题标记 **<hn>…</hn>**
  - ✓**<h1>…</h1>**到**<h6>…</h6>**6种，用于表示文章中的各种题目。
  - ✓字体大小**<h1>**到**<h6>**顺序减小。

字体/字型标记

字型标记 <b>、<i>、<u>等 。

字体标记font

- <FONT FACE="隶书" SIZE="5" COLOR="Blue">
- </FONT>

图片标记

IMG标记<img src=“图象资源地址” >

IMG的属性有

- **width，height**
- **alt用来为图象指定描述性文字，当图象不能正常显示时出现；**
- **border表示边框的宽度，0表示无边框。**

```
<HTML>
        <BODY>
        <IMG SRC=“myimage.jpg” WIDTH=“200” HEIGHT=“100” BORDER=“10” />
        </BODY>
</HTML>
```

超级链接

Anchor用于定义超链接之间的关系。通过点击超链接，可以在各个页面之间转换，将网页组织起来。

<a>的最常用参数**href**用于指明所链接对象的**URL**。

URL可以指向一个页面、一副图片或一篇文章中作好标记的段落等。

```
<HTML>
        <BODY>
          <A HREF="http://www.tsinghua.edu.cn">清华大学</A>
        </BODY>
</HTML>
```

列表

HTML中列表分为：有序列表、无序列表

- 有序列表显示时，会自动按照顺序编号；
- 无序列表显示时，所以行前面都有小圆圈，不分先后

```
<HTML><BODY>
                    有序列表
                    <OL>
                      <LI>第一</LI>
                      <LI>第二</LI>
                    </OL>
                    无序列表
                    <UL>
                      <LI>清华大学</LI>
                      <LI>北京大学</LI>
                    </UL>
</BODY></HTML>
```

基本表格

- <TABLE>是表格的基本标记。
- <TR>代表表格的行，<TD>代表表格的列。
- BORDER属性，指定围绕表格的外边框的宽度（只能用像素）。当表格用来使版面显得整齐，也就是用来“定位”，那么一定将边框设为0。

```
<HTML><BODY>
    <TABLE BORDER="1">
      <TR>  <TD>第一行第一列</TD><TD>第一行第二列</TD> </TR>
      <TR>  <TD>第二行第一列</TD><TD>第二行第二列</TD> </TR>
      <TR>  <TD>第三行第一列</TD><TD>第三行第二列</TD> </TR>
    </TABLE>
</BODY></HTML>
```

表单

- 表单的功能是收集用户信息，实现系统与用户交互。
  - ✓比如**E-mail**信箱的注册页面就是一个十分典型的表单页面。
- 表单信息的处理过程如下：
  - ✓当单击表单中的提交按钮时，表单中的信息就会上传到服务器中
  - ✓然后由服务器端的应用程序进行处理，处理后将用户提交的信息存储在服务器端的数据库中，或者将有关信息返回到客户端浏览器上。

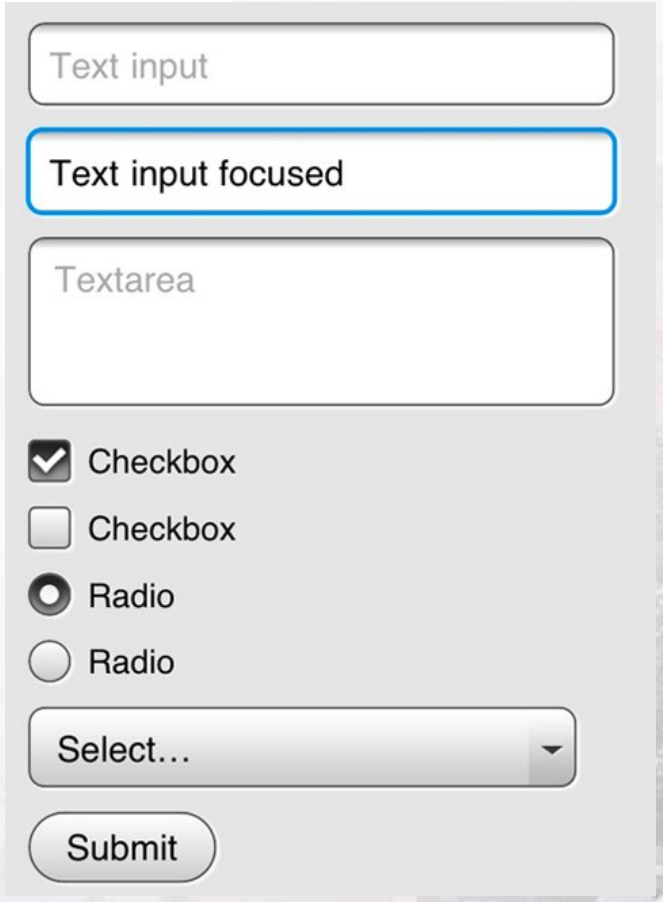
表单头及其属性

METHOD属性：说明从客户端浏览器将输入信息传送给Web服务器的方式，有两种方式：POST和GET。

```
<HTML><BODY>
        <FORM METHOD="Post" ACTION="do_submit.htm" >
        用户名：  <INPUT TYPE="Text" NAME="UserID"><BR>
        密码：    <INPUT TYPE="Password" NAME="UserPWD"><BR><BR>
                  <INPUT TYPE="Submit" VALUE="提交" NAME="B1">
                      <INPUT TYPE="Reset" VALUE="重写" NAME="B2">
        </FORM>
</BODY></HTML>
```

表单头及其属性

- 常见的表单控件包括文本框、文本域、密码框、多选框、单选框和下拉列表框，等等。



HTML中的INPUT标记

- <INPUT TYPE="…" VALUE ="…" NAME="…">
- TYPE属性：说明信息输入的类型。例如是文本框、单选按钮或多选按钮。它的取值如下：
  - ✓ – TYPE = "TEXT" 表示单行文本框
  - ✓ – TYPE = "PASSWORD" 表示密码输入框
  - ✓ – TYPE = "RADIO" 表示单选按钮
  - ✓ – TYPE = "CHECKBOX" 表示多选按钮
  - ✓ – TYPE = "SUBMIT" 表示提交按钮
  - ✓ – TYPE = "RESET" 表示重置按钮
- NAME属性：表示表单提交时作为输入信息的命名（相当于变量名）
- VALUE属性：表示按钮上的标题或者文本框的内容（相当于变量值）

如何让HTML显示更美观呢？

如何使得显示和数据分离呢？

CSS

- **Cascading Style Sheets 层叠样式表单**
- 告诉浏览器如何显示页面
- 美化页面

CSS

- 语法 - 属性：值
  - **Font-size: large**
  - **Color: red**
- 例子：
  - h4 { font-size: 12px; color:red; }
  - h2 {
    background: #f0f0f0; 背景颜色
    margin: 15px 0; 上下边界15 左右0
    }

CSS样式选择

- Id 和 class

```
<style>
    #id {background-color:blue}
    .class1 {color:red;}
    .class2 {width:200px;height:100px}
</style>
<div id="id"> 蓝色理想 </div>
<div class="class1"> 红色理想 1</div>
<div class="class1"> 红色理想 2</div>
```

- Id唯一
- Class可重用

CSS引用方式

- 内部引用

```
<head>
<style type="text/css"> body { background-color: blue; } </style>
</head>
```

- 外部引用（独立文件引用）

```
<head>
<link rel="stylesheet" type="text/css"  href="位置/CSS文件名.css" />
/*文件位置就是所处在的文件夹相对与当前网页的相对路径*/
</head>
```

有什么办法对HTML中的内容进行动态改变呢？

Javascript

- JavaScript是一种网页编程技术，大部分使用者将它用于创建动态交互网页
- JavaScript是internet上最流行的脚本语言，它可以在包括IE, Mozilla, Firefox, Netscape, 和 Opera的所有主流浏览器中工作
- JavaScript是一种基于对象和事件驱动的解释性脚本语言
- JavaScript是一种使用简单，功能强大的编程语言，是搭配服务器端技术的主要客户端编程语言

Javascript功能

- 实现客户端动态效果

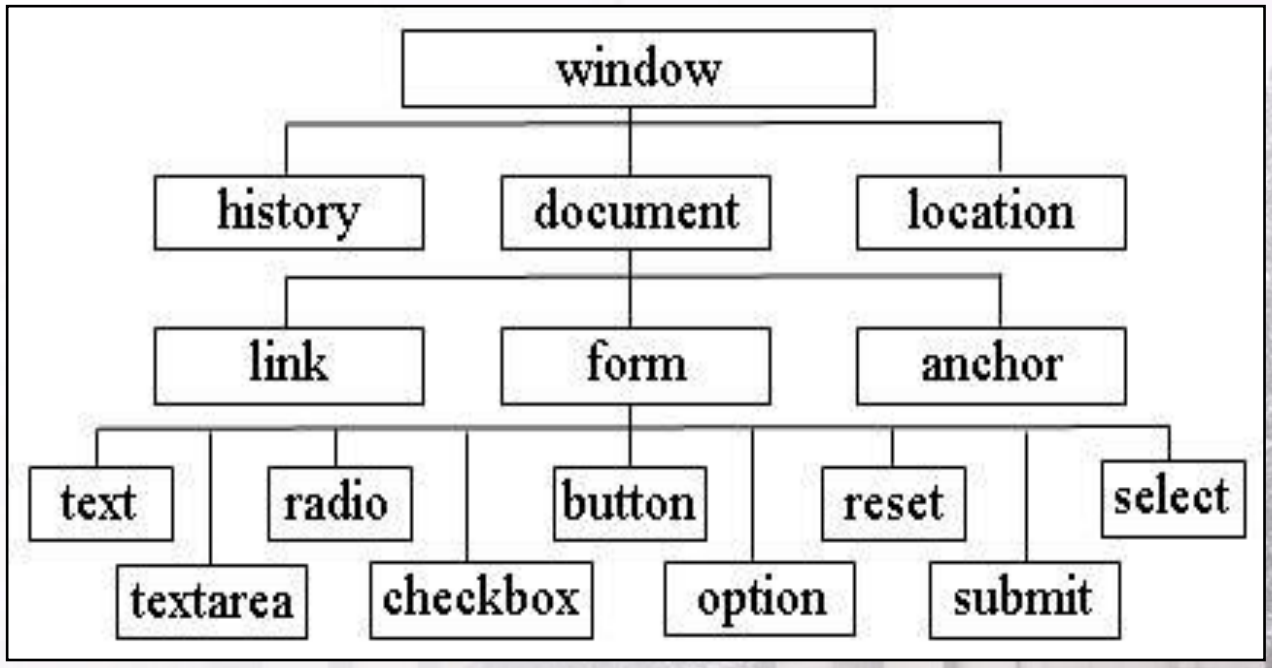
- JavaScript 为 HTML 设计者提供了一个编程工具
- JavaScript 可以在HTML页面中插入动态文本
- JavaScript 可以对事件进行反应
- JavaScript 可以读写HTML元素
- JavaScript 可以被用来提交、验证数据

JavaScript特点

- 简单易用
  - ✓可以使用任何文本编辑工具编写
  - ✓只需要浏览器就可以执行程序
- 动态语言 解释执行
  - ✓事先不编译，逐行执行
  - ✓无需进行严格的变量声明
- 基于对象
  - ✓内置大量现成对象，编写少量程序可以完成目标
- 事件驱动
  - ✓采用事件驱动方式，能响应键盘事件，鼠标事件以及浏览器窗口事件等，并执行指定得操作

JavaScript中的主要对象

Window对象
Document对象
location对象
history对象
frame对象
Frames数组
form对象
Forms数组
elements对象
text对象
button对象
……



适用场合

- 适合做哪些事情
  - ✓客户端数据计算
  - ✓客户端表单合法性验证
  - ✓浏览器对象的调用
  - ✓浏览器事件的触发
  - ✓网页特殊显示效果制作
- 不适合做哪些事情
  - ✓大型应用程序
  - ✓图像、多媒体处理
  - ✓网络实时通讯应用

直接嵌入式

- 在网页中直接嵌入JavaScript

```
<script language="JavaScript">
<!-
/*
程序功能：书写方法说明
开始和结束的标志的书写方法
单行和多行JavaScript注释的写法
*/
function sayHello(){  //在HTML文档中显示hello
        document.write("hello");
}
sayHello();
-->
</script>
```

脚本开始声明

HTML注释

JavaScript多行注释

JavaScript单行注释

语句结尾

脚本结束声明

文件调用式

➢在网页中调用独立JavaScript文件

```
<script language="JavaScript" src="test1-3.js"></script>
```

HTML文件

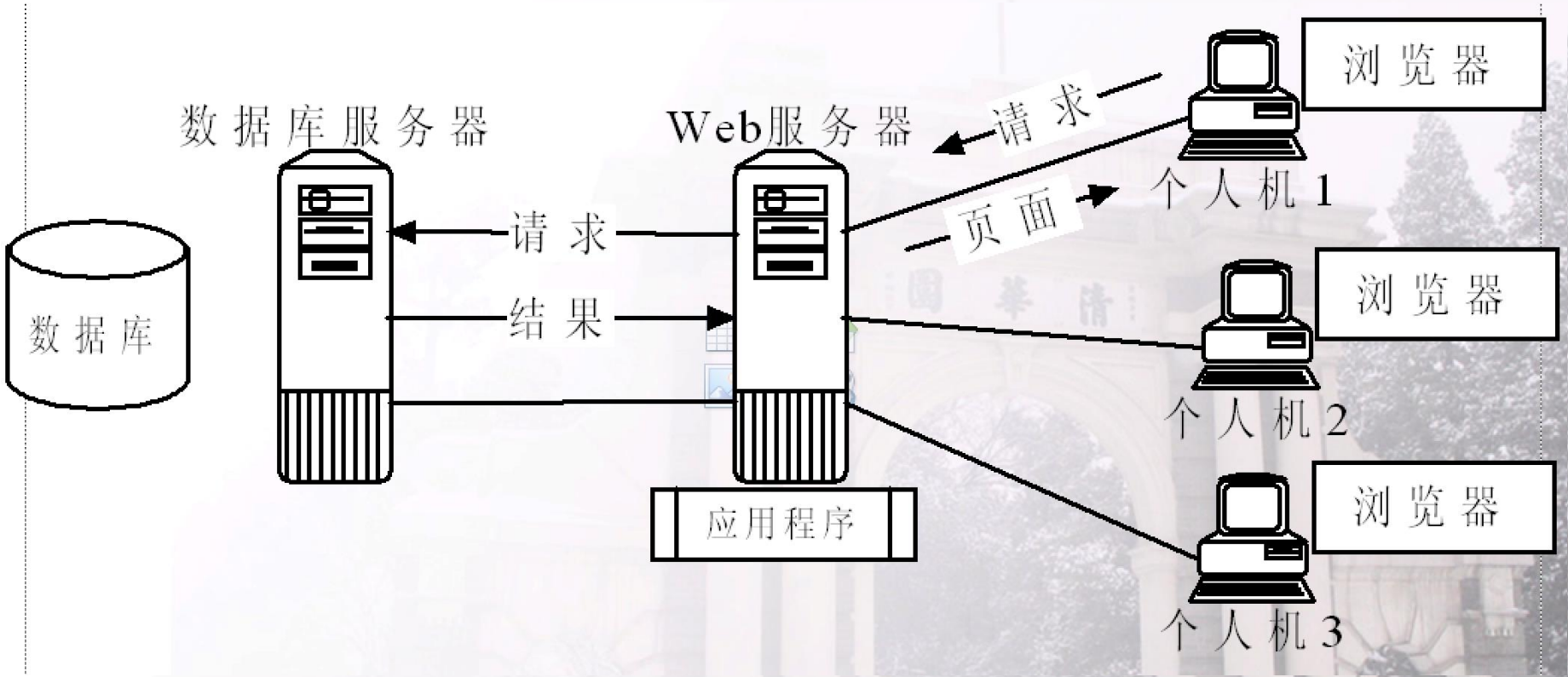
```
<!--
function sayHello(){
        //在HTML文档中显示hello
        document.write("hello from js");
}
sayHello();
-->
```

JS脚本文件中不需要脚本开始和结束声明

服务器端

- 高级语言
  - ✓Java
  - ✓C
  - ✓Python
- 交互式数据处理
- 一般与数据库打交道

BS框架


数据库服务器
Web服务器
请求
浏览器
个人机1
请求
页面
数据库
请求
结果
浏览器
个人机2
浏览器
应用程序
个人机3

服务器客户端数据交换

- **Request对象从客户端获得信息**
  - ✓**QueryString**
  - ✓**Form**
  - ✓**Cookies**

服务器客户端数据交换

- **Response对象从服务器端返回客户端信息**
  - ✓**Write**
  - ✓**Redirect**

客户端服务器端交换数据语言

- JSON
- XML

JSON

- JSON (JavaScript Object Notation)
- 例如返回搜索引擎结果
  - ✓<title, url, snippet>
  - ✓JSON结构：

```
{“answers": [
{“title": “1", “url": “1.html", “snippet": “1.
snippet"},
{“title": “2", “url": “2.html", “snippet": “2.
snippet"},
 “title": “3", “url": “3.html", “snippet": “3.
snippet"}
]}
```

JSON

- **JavaScript Object Notation**
- 一种轻量级的数据交换格式。
- 易于人阅读和编写，也易于机器解析和生成。
- 完全独立于语言的文本格式
- 客户端用**Javascript**解析显示

JSON优点

- 体积小、传输快、省带宽
- 易于机器的解析和生成
- 支持多语言
- 能够通过JavaScript中eval()函数解析JSON

使用XML

- XML (eXtensive Markup Language)
- 例如返回搜索引擎结果
  - ✓ <title, url, snippet>
  - ✓ XML结构：

```
<?xml version="1.0" encoding="utf-8"?>
<answers>
    <answer>
      <title> 1 </title>
      <url> 1.html </url>
      <snippet>1.snippet</snippet>
    </answer>
    <answer>
      <title> 2 </title>
      <url> 2.html </url>
      <snippet>2.snippet</snippet>
    </answer>
    …
</answers>
```

JSON和XML比较

```
{ cn: {
        lang: 'zh_cn',
       name: '中文’  },
    en: {
        lang: 'us_en',
        name: '英文’   }
 }
```

```
<?xml version="1.0" encoding="utf-8"?>
 <languages>
    <cn>
        <lang>zh_cn</lang>
        <name>中文</name>
    </cn>
    <en>
        <lang>us_en</lang>
        <name>英文</name>
    </en>
 </languages>
```

JSON和XML比较

- XML需要创建ActiveX对象，需要DOM解析，JSON则完全就是一个JS对象
- XML可读性更好

如何进行服务器端编程？

如何进行网络编程？

网络编程的痛苦

1. 从头开始编写网络应用程序。
2. 从头编写另一个网络应用程序。
3. 从第一步中总结（找出其中通用的代码），并运用在第二步中。
4. 重构代码使得能在第 2 个程序中使用第 1 个程序中的通用代码。
5. 重复 2-4 步骤若干次。
6. ☹ 意识到你发明了一个框架。

Web框架

- 提供通用Web开发模式的模型
- 提供频繁进行的编程作业的快速解决方案
- MVC
  - ✓模型（Model）：封装数据和所有基于对数据的操作
  - ✓视图（View）：封装数据显示，即用户界面
  - ✓控制器（Control）：封装外界作用于模型的操作和对数据流向的控制

Web框架功能

- 提供URL映射
  - ✓ 如何根据一个URL 找到需要执行的代码
- 通过模板系统分离内容和显示
- 对用户提交的东西转换成容易操控的数据结构
- 更高的抽象

常用web框架

- PHP：FleaPHP，CakePHP，Joomla
- JAVA：Struts，Spring
- Python：Django，Quixote
- Ruby：Ruby On Rails

Python的web编程框架

Django

- 由 堪萨斯（Kansas）州 Lawrence 城中的一个网络开发小组编写的
- 2003开始开发，2005年完成
- 来源于一个著名的爵士乐吉他演奏家Django Reinhardt。
- Adrian 和 Simon 开发

Django特点

- 提供强大的网络编程框架
- 一流的 URL 设计
- 通过模板系统分离内容和显示

Django 获取

- Django-1.11.5.tar.gz - Python 2
- Django-2.1.1.tar.gz - Python 3
  - ✓ https://www.djangoproject.com/download/
- Document
  - ✓ https://docs.djangoproject.com/

- ✓ Python Install
  - ✓ https://pypi.python.org/pypi/setuptools#downloads
  - ✓ python setup.py install

安装

- 安装
- python setup.py install
- 创建目录> **python django\bin\django-admin.py startproject hwtest**
- 启动**server> python hwtest\manage.py runserver**
  - ✓ **Validating models...**
  - ✓ **0 errors found**
  - ✓ **Django version 1.8, using settings 'hw.settings'**
  - ✓ **Development server is running at http://127.0.0.1:8000/**
  - ✓ **Quit the server with CTRL-BREAK.**

✓python manage.py runserver IP:Port

Start App (Django 2.1.1)

- 在**hwtest**目录下运行
  - ✓**python manage.py startapp mytest**
  - ✓**python manage.py runserver**

文件作用介绍

- __init__.py：
  - ✓当成一个开发包 (即一组模块)所需的文件
- manage.py：
  - ✓命令行工具
  - ✓提供简单化的django-admin.py 命令
- settings.py: django的配置文件
- urls.py
  - ✓url 映射处理文件

settings.py

```
ROOT_URLCONF = 'hw.urls‘
```

- ✓*URLconf* 就像是 Django 所支撑网站的目录。
- ✓它的本质是 URL 、视图函数之间的映射表。

```
TEMPLATE_DIRS = (
    'hw/templates',
)
STATICFILES_DIRS = (
  'hw/static',
)
```

Django的一些设置

urls.py

```
from django.conf.urls.defaults
import * urlpatterns = patterns('',
# Example: # (r'^mysite/', include('mysite.apps.foo.urls.foo')),
# Uncomment this for admin:
# (r'^admin/', include('django.contrib.admin.urls')), )
url(r'^$', 'hw.hw.index'),
url(r'^hw/', 'hw.hw.indexhw')
```

第一行从 django.conf.urls.defaults 模块引入了所有的对象，其中包括了叫做 patterns 的函数。

第二行调用 patterns() 函数并进行url映射

urls.py (Django 2.1.1)

```
from django.contrib import admin
from django.urls import path
from django.conf.urls import url
from mytest import views

urlpatterns = [
    path('admin/', admin.site.urls),
    url(r'^$',views.index),
    url(r'^hw/',views.indexhw),
]
```

匹配规则

- r'^hw/$' 中的 r 表示 '^hw/$' 是一个原始字符串。
  - ✓避免正则表达式有过多的转义字符。
- 不必在 '^hw/$' 前加斜杠（/）来匹配 /hw/
  - ✓Django 会自动在每个表达式前添加一个斜杠。
- 上箭头 ^ 和美元符号 $ 符号非常重要。
  - ✓^对字符串的头部进行匹配
  - ✓$对字符串的尾部进行匹配。

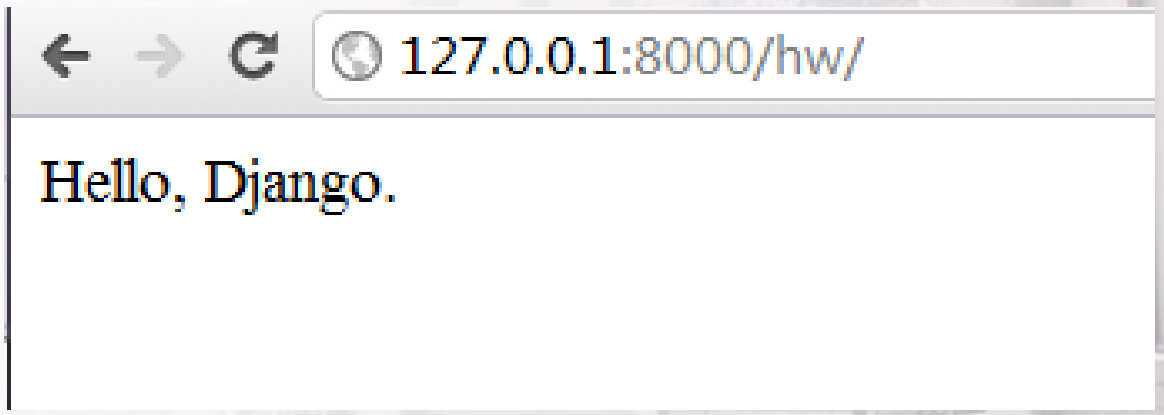
Hello World

hw.py

```
from django.http import HttpResponse
def index(request):
    return HttpResponse("Hello, Django.")
def indexhw(request):
    return HttpResponse("Hello World Demo!")
```

urls.py

```
url(r'^$', 'hw.hw.index')
url(r'^hw/', 'hw.hw.indexhw')
```



Django是怎么处理请求的？

- django-admin.py startproject 创建的 settings.py 和 urls.py
  - ✓ 系统自动生成的 settings.py -- ROOT_URLCONF 默认设置是 hw.urls 。
- python manage.py runserver 从同一目录载入文件 settings.py 。
  - ✓ 该文件包含了这个特定的Django实例所有的各种可选配置
  - ✓ 最重要的配置是 ROOT_URLCONF
  - ✓ ROOT_URLCONF 告诉Django哪个Python模块用作本网站的URLconf
- Django 根据 ROOT_URLCONF 的设置，访问 /hw/urls.py ，
- 从URL Patterns查找， 找到匹配的URLpatterns，调用相关联的view函数，并把 HttpRequest 对象作为第一个参数

什么是Python视图

- 视图函数（或 视图 ）是一个接受 Web 请求并返回 Web 响应的 Python 函数。
- 该响应可以是
  - ✓一份网页的 HTML 内容
  - ✓一次重定向
  - ✓一条 404 错误
  - ✓一份 XML 文档
  - ✓一幅图片
  - ✓或其它任何东西

静态页面实现

settings.py

```
STATIC_ROOT = ‘static/’  1.10
STATIC_URL = ‘static/’
STATICFILES_DIRS = (                    1.4
    ‘hw/static/’, #任意目录，注意是/ 不是\
)
```

urls.py

```
from django.conf import settings

url(r'^static/(?P<path>.*)$', 'django.views.static.serve', {'document_root':
    settings.STATIC_ROOT}),
```

静态页面实现

settings.py for django2.1.1

```
STATIC_URL = '/static/'

STATICFILES_DIRS = [
os.path.join(BASE_DIR, 'static'),
]
```

动态页面实现

- 视图是一个简单的 Python 方法/函数，它接受一个请求对象，负责实现：
  - ✓任何业务逻辑（直接或间接）
  - ✓响应对象，它将所表示的结果返回到这个框架中
- 在 Django 中，当一个 URL 被请求时，所调用的 Python 方法称为一个视图view，这个视图加载并被执行

例子：显示时间

time.py

```
from django.http import HttpResponse
import datetime
def current_datetime(request):
    now = datetime.datetime.now()
    html = "<html><body>It is now %s.</body></html>" % now
    return HttpResponse(htm)
def hours_ahead(request, offset):
    offset = int(offset)
    dt = datetime.datetime.now() + datetime.timedelta(hours=offset)
    html = "<html><body>In %s hour(s), it will be %s.</body></html>" % (offset, dt)
    return HttpResponse(html)
```

urls.py

```
url(r'^time/$', 'hw.time.current_datetime'),
url(r'^time/plus/(\d{1,2})/', 'hw.time.hours_ahead'),
```

如何进行数据交互?

Httprequest对象

```
def hours_ahead(request, offset):
    offset = int(offset)
    dt = datetime.datetime.now()+datetime.timedelta(hours=offset)
    html = "<html><body>In %s hour(s), it will be
            %s.</body></html>"  % (offset, dt)
    return HttpResponse(html)
```

request 是一个 HttpRequest 对象

每一个视图 *总是* 以一个HttpRequest 对象作为它的第一个参数。

Httprequest对象属性

属性	描述
path	表示提交请求页面完整地址的字符串， 不包括域名，如 "/music/bands/the_beatles/" 。

method	表示提交请求使用的HTTP方法。 它总是大写的。例如：
	`if request.method == 'GET':`
	`    do_something()`
	`elif request.method == 'POST':`
	`    do_something_else()`

GET	一个类字典对象，包含所有的HTTP的GET参数的信息。 见 QueryDict 文档。
POST	一个类字典对象，包含所有的HTTP的POST参数的信息。 见 QueryDict 文档。 通过POST提交的请求有可能包含一个空的 POST 字典， 也就是说， 一个通过POST方法提交的表单可能不包含数据。 因此，不应该使用 if request.POST 来判断POST方法的使用， 而是使用 if request.method == "POST" （见表中的 method 条目）。 注意： POST 并 不 包含文件上传信息。 见 FILES 。
REQUEST	为了方便而创建，这是一个类字典对象，先搜索 POST ， 再搜索 GET 。 灵感来自于PHP的 $_REQEUST 。 例如， 若 GET = {"name": "john"} ， POST = {"age": '34'} ， REQUEST["name"] 会是 "john" ， REQUEST["age"] 会是 "34" 。 强烈建议使用 GET 和 POST ， 而不是 REQUEST 。 这是为了向前兼容和更清楚的表示。

Httprequest对象方法

方法	描述
`__getitem__(key)`	请求所给键的GET/POST值，先查找POST，然后是GET。 若键不存在，则引发异常 `KeyError` 。 该方法使用户可以以访问字典的方式来访问一个 `HttpRequest` 实例。 例如， `request["foo"]` 和先检查 `request.POST["foo"]` 再检查 `request.GET["foo"]` 一样。
`has_key()`	返回 `True` 或 `False` ， 标识 `request.GET` 或 `request.POST` 是否包含所给的 键。
`get_full_path()`	返回 `path` ，若请求字符串有效，则附加于其后。 例如，`"/music/bands/the_beatles/?print=true"` 。
`is_secure()`	如果请求是安全的，则返回 `True` 。 也就是说，请求是以HTTPS的形式提交的。

例子

```
<form action="/hw/formsubmit/" method="post">
<input type="text" name=“your_name" />
    <select multiple="multiple" name="bands">
        <option value='Beatles'>The Beatles</option>
        <option value='Who'>The Who</option>
        <option value='Zombies'>The Zombies</option>
   </select>  <br/>
    <input type="submit" />
</form><br/>
```

若用户输入了 "John Smith“ 在多选框中同时选中了 The Beatles 和 The Zombies, 然后点击 Submit, Django的request对象将拥有:

```
request.GET {}
request.POST   {'your_name': ['John Smith'], 'bands': ['beatles', 'zombies']}
request.POST['your_name']    'John Smith'
request.POST.getlist('bands')    ['beatles', 'zombies']
```

例子

reply.py

```
from django.http import HttpResponse
from django.views.decorators.csrf import csrf_exempt
text = """<form action="/hw/formsubmit/" method="post">
    <input type="text" name="your_name" /><br />
     <select multiple="multiple" name="bands">
            <option value='Beatles'>The Beatles</option>
            <option value='Who'>The Who</option>
            <option value='Zombies'>The Zombies</option>
   </select>  <br/>  <input type="submit" />  </form>  br/>
   <input type="text" value="%s ; %s" width="400"/>
"""
@csrf_exempt
def index(request):
    return HttpResponse(text % (request.POST['your_name'],
     request.POST.getlist('bands')))
```

urls.py

```
url(r'^hw/formsubmit/$',
'hw.reply.index'),
```

HttpResponse

- 与Django自动创建的 HttpRequest 对象相比，HttpResponse 对象则是由程序员创建的。
- 程序员创建的每个视图都需要实例化，处理和返回一个 HttpResponse 对象。

- 可以返回HTML
- 可以返回JSON
- 可以返回XML

例子：交互

add.py

```
from django.http import HttpResponse
from django.views.decorators.csrf import csrf_exempt
text = """<form method="post" action="/add/">
    <input type="text" name="a" value="%d"> + <input
      type="text" name="b" value="%d">
    <input type="submit" value="="> <input type="text"
      value="%d"></form>"""
@csrf_exempt
def index(request):
    if request.POST.has_key('a'):
        a = int(request.POST['a'])
        b = int(request.POST['b'])
    else:
        a = 0
        b = 0
    return HttpResponse(text % (a, b, a + b))
```

urls.py

```
url(r'^add/', 'hw.add.index'),
```

如何实现数据显示？

127.0.0.1:8000/list/

通讯录

姓名	地址
张三	zhangsan@gmail.com
李四	lisi@thu.edu.cn
王五	wangwu@sina.cn

list.html

```
<h2>通讯录</h2>
<table border="1">
    <tr>  <th>姓名</th>
          <th>地址</th>
    </tr>
    <tr>  <td>张三</td>
          <td>zhangsan@gmail.com</td>
    </tr>
    <tr>   <td>李四</td>
           <td>lisi@thu.edu.cn</td>
    </tr>
    <tr>    <td>王五</td>
           <td>wangwu@sina.cn</td>
     </tr>
</table>
```

模板

- 页面频繁修改，底层 Python 代码是否需要修改？
- Python 代码编写和 HTML 设计是两项不同的工作
  - ✓ 大多数专业的网站开发环境都将他们分配给不同的人员来完成。
  - ✓ HTML/CSS 编写人员
  - ✓ Python编写人员
- 模板进行HTML/Python分离

数据显示分离

list1.html

```
<h2>Textbook 1</h2>
<table border="1">
  <tr><th>Name</th><th>Email
     Address</th></tr>
 {% for user in address %}
  <tr>
  <td>{{ user.name }}</td>
  <td>{{ user.email }}</td>
  </tr>
{% endfor %}
</table>
```

{{}} 表示引用一个变量
{%%} 表示代码调用

https://docs.djangoproject.com/en/dev/ref/templates/builtins/?from=olddocs

list.py

```
from django.shortcuts import render_to_response

address=[
    {'name':'San Zhang',
     'email':'zhangsan@gmail.com'},
    {'name':'Si Li', 'email':'lisi@thu.edu.cn'},
    {'name':'Wu Wang', 'email':'wangwu@sina.cn'}
    ]

def index(request):
    return render_to_response('list1.html',
     {'address': address})
```

urls.py

```
url(r'^list/', 'hw.list.index'),
```

模板文件存放位置

settings.py

```
TEMPLATE_DIRS = (
    # Put strings here, like "/home/html/django_templates" or
    "C:/www/django/templates".
    # Always use forward slashes, even on Windows.
    # Don't forget to use absolute paths, not relative paths.
     'hw/templates',
)
```

如何使用模板

- 方法1
  - ✓ 直接渲染
    - render_to_response('list1.html', {'address': address})
- 方法2
  - ✓ 加载模板文件创建 Template对象
    - t = loader.get_template('list2.html')
  - ✓ 调用 Template 对象的 render() 方法并提供给他变量(内容)
    - c = Context({'address': address})
    - response.write(t.render(c))

方法2

lists.py

```
from django.template import loader, Context
from django.http import HttpResponse

address=[
    {'name':'Zhang San',
     'email':'zhangsan@gmail.com'},
    {'name':'Li Si', 'email':'lisi@thu.edu.cn'},
    {'name':'Wang Wu', 'email':'wangwu@sina.cn'}
    ]
def index(request):
    response = HttpResponse(mimetype='text/html')
    t = loader.get_template('list2.html')
    c = Context({ 'address': address})
    response.write(t.render(c))
    return response
```

list2.html

```
<h2>Textbook 2</h2>
<table border="1">
  <tr><th>Name</th><th>Email
     Address</th></tr>
 {% for user in address %}
  <tr>
  <td>{{ user.name }}</td>
  <td>{{ user.email }}</td>
  </tr>
{% endfor %}
</table>
```

urls.py

```
url(r'^list2/$', 'hw.lists.index'),
```

模板语法

- 变量(variable)：
  - ✓ 用两个大括号括起来的文字
  - ✓ 例如 {{ person_name }}
- 模板语法标签(template tag)
  - ✓ 被大括号和百分号包围的文本
  - ✓ 例如 {% if ordered_warranty %}
- for标签(tag): {% for item in item_list %}
- If标签 {% if ordered_warranty %}
- {% else %}

If else

```
{% if athlete_list %}
    <p>Here are the athletes</p>
{% else %}
    <p>No athletes are available.</p>
    {% if coach_list %}
        <p>Here are the coaches</p>
    {% endif %}
{% endif %}
```

and or not

```
{% if athlete_list and coach_list %}
    Both athletes and coaches are available.
{% endif %}
{% if not athlete_list %}
    There are no athletes.
{% endif %}
{% if athlete_list or coach_list %}
    There are some athletes or some coaches.
{% endif %}
{% if not athlete_list or coach_list %}
    There are no athletes or there are some coaches.
 {% endif %}
```

在python中空

- ✓列表 ([])
- ✓tuple(())
- ✓字典({})
- ✓字符串('')
- ✓零(0)
- ✓ None 对象

在逻辑判断中都为假
其他的情况都为真。

for

```
<ul>
   {% for athlete in athlete_list %}
        <li>{{ athlete.name }}</li>
   {% endfor %}
</ul>
{% for athlete in athlete_list reversed %}
   反向迭代
 {% endfor %}
```

Thanks
Questions?