

# 使用道语言作网站编程

傅利民(phoolimin@gmail.com)

September 19, 2009

本文档将不介绍如何进行网站编程，而仅介绍如何将道语言用作服务器端脚本语言。服务端脚本程序主要是通过公共网关协议（Common Gateway Interface, CGI）来与网站服务器程序协作。脚本程序通过CGI获取HTTP请求的有关信息，然后决定如何回应。一般情况下，脚本程序还需要查询数据库并将结果以恰当的方式反馈给客户端网页浏览器。

## 1 使用CGI模块

DaoCGI是一个用来解析HTTP请求信息的模块，它将解析的结果以结构化的数据（道语言变量）形式供脚本程序使用。

载入DaoCGI:

```
load DaoCGI;
```

载入后，当前的名字空间将增加如下全局变量:

变量名	类型	含义
HTTP_ENV	map<string,string>	HTTP请求的环境变量
HTTP_GET	map<string,string>	GET请求的变量
HTTP_POST	map<string,string>	POST请求的变量
HTTP_GETS	map<string,list<string>>	GET请求的变量
HTTP_POSTS	map<string,list<string>>	POST请求的变量
HTTP_COOKIE	map<string,string>	请求所附带的cookies
HTTP_FILE	map<string,stream>	请求所上传的文件

对于HTTP\_GETS和HTTP\_POSTS，里面可能包含同一个键的多个值。对于HTTP\_FILE其元素的键是文件的上传名，

```
<input type="file" name="上传名" id="filename"/>
```

而元素的值是流对象，其中内容是上传的文件的数据。  
一个简单的CGI程序，

```
load DaoCGI;
stdio.println( 'content-type: text/plain\n' );
stdio.println( 'You are from:', HTTP_ENV[ 'REMOTE_ADDR' ] );
stdio.println( 'Welcome to visit:', HTTP_ENV[ 'REQUEST_URI' ] );
```

## 2 数据库处理

目前道语言还只有处理MYSQL数据库的模块，其处理方式是将道语言类映射到数据库表上，建立类成员到表域名的对应关系，然后通过类实例操作数据库表里的记录。请参看更详细的文档使用*DaoDataModel*模块处理数据库。

## 3 简单的模型(Model)与视图(View)分离

通常情况下，服务端脚本程序要负责生成HTML代码以在网页浏览器里显示信息。这些信息或数据一般被保存在服务器上的文件或数据库里，脚本程序也要负责这些数据的处理，此即为模型部分。当这些数据需要被显示在客户端的浏览器里时，脚本程序需要生成合适的HTML代码以达到恰当可视化效果，这就是视图部分。

显然程序的模型部分和视图部分最好是分离的，这样对一个部分的修改将不影响另一个部分。这使得程序的维护和升级更加简单安全。道语言的*DaoDataModel*比较适合处理程序的模型部分，而道语言的一些字符串方法将使得视图部分也比较简单。下面将以一个例子来作介绍。

假设有有一个名为*Friend*的数据库表，它保存着包含如下信息的地址簿：

```
id : INTEGER PRIMARY KEY AUTOINCREMENT
name : CHAR(50)
phone : CHAR(20)
city : CHAR(50)
street : VARCHAR(100)
```

其对应的道语言类是:

```
typedef tuple<id:string,name:string,phone:string,city:string,street:string> friend_t;

class Friend
{
    my id : INT_PRIMARY_KEY_AUTOINCREMENT;
    my name : CHAR50;
    my phone : CHAR20;
    my city : CHAR50;
    my street : CHAR100;

    routine AsTuple() => friend_t {
        return ( (string) id, name, phone, city, street );
    }
}
```

这里还定义了一个元组类型`friend_t`，它包含了带名称的元素。这个元组类型将是程序的模型部分与视图部分传递数据的基本数据结构。

再假设，相关数据库被打开为:

```
global model = DataModel( 'dbname', 'host', 'user', 'password' );
```

现在在模型部分，数据库的查询将变得很简单。例如，如果要显示某个城市里的所有朋友，可以这样做:

```
routine FriendInCity( city : string ) => list<friend_t>
{
    friends = {};
    friend = Friend();
    hd = model.Select( Friend ).Where().EQ( 'city', city );
    while( hd.Query( friend ) ) friends.append( friend.AsTuple() );
    hd.Done();
    return friends;
}
```

显然这个函数也可被定义为`Friend`类的成员方法。

数据视图的创建可以使用模板`HTML`代码进行简化。例如，如果需要在一张表里显示所有被查询的朋友，可以定义如下`HTML`模板:

```
<tr class="myrowstyle"><td class="mycellstyle">@(id)</td>
<td>@(name)</td><td>@(phone)</td>
<td>@(city)</td><td>@(street)</td></tr>
```

`HTML`模板可用来决定显示那些数据，以何种方式显示。数据文本的字体颜色

等也可被包含在模板里，不过，使用层叠样式表CSS来设置它们更方便。

视图的创建可以是：

```
routine ViewFriends( friends : list<friend.t>, rowtpl = '' )
{
    html = '<table class="mytablestyle">\n';
    for( friend in friends ) html += rowtpl.expand( friend, '@' );
    html += '</table>';
    return html;
}
```

这里rowtpl.expand()方法将展开字符串rowtpl，并替换里面的占位变量，替换内容为元组friend里的跟占位变量同名的元素的值。expand()还有一个重载的函数，用来接受映射表为参数。

最后，查询数据库并创建视图可以用，

```
row = '<tr class="myrowstyle"><td class="mycellstyle">@(id)</td>
<td>@(name)</td><td>@(phone)</td>
<td>@(city)</td><td>@(street)</td></tr>';

friends = FriendInCity( 'Beijing' );
view = ViewFriends( friends, row );
```

通过改变模板HTML代码（和CSS），视图的形式和效果可以很容易地被改变。