OpenAcs 入门 *translated by i-robot*

前言

这是对《Getting Started With OpenAcs》的翻译，对其中的错误做了一些修改，去掉了一些不必要的介绍，比如全书的构成之类，图片也省略了，我会尽可能地把事情说清楚。不使用图片原因是因为，我觉得有图的 tutorial 学起来太偷懒了，无助于深入理解。另外与原作不同的是，翻译稿中代码得实验环境是在 windows 中的，因此省略了在 Ubuntu 系统中安装 OpenAcs 的介绍，如有需要可以查看原文。

Day 1

杂七杂八

在第一天里，我们要做到系统中正确启动运行 OpenAcs，然后创建 OpenAcs 的第一个 web 页面。在此期间，我们会学习到如何建立 OpenAcs 的测试站点，什么是 Tcl 及上哪儿学习 Tcl，如何使用 OpenAcs 建立最基本的 web 页面。

至于在 windows 上的安装，我想就不必多说了，双击打开安装包后一路回车即可。只是 OpenAcs 的 windows 安装包不好找，官网的联系始终是打不开，有需要的可以直接跟我联系或者问下 smalltalk，我也是从他那儿要来的，万分感谢。安装好之后，别忘接了先启动 postgresql 服务，再启动 OpenAcs 服务，这样才能开始开发使用。需要特别提醒的是，对于主站的操作需要身份的验证，OpenAcs 的默认账户为“admin@acme.com”，密码是“abc123”。最好再建一个自己常用的账户，并赋予其管理员权限。

那么先说一下什么是 Tcl。Tcl 就是“Tool Command Language”的缩写，据我所知这是一个语法很简单却很实用的语言，自从从 Perl 转到 Tcl 上以来，我写的代码 bug 直线减少。那么为什么 OpenAcs 实用 Tcl 作为它的默认语言呢？这就首先要搞清楚 OpenAcs 是什么。它是一个应用在 AOLServer 之上的工具集，而 AOLServer 的默认开发语言是 Tcl，所以顺理成章的 OpenAcs 也使用 Tcl 作为默认的开发语言。

原文中建议学习 Tcl 到 http://philip.greenspun.com/tcl/ 这个网站，不过我个人建议实用 TclTutor。因为 TclTutor 程序，除了有介绍 Tcl 的语法之外，还给出了大量可执行的例子，方便边学边练，有助于真正掌握 Tcl。

第一个页面

OpenAcs 里的页面页以 adp 作为后缀名的，意思是指 AOLServer Dynamic Pages，它们就像普通的 html 页面，只不过加入了一些 tags 和扩展，以便使用来自同名 tcl 文件的动态信息。其实说白了，就是对于一个页面要有至少两个文件，这两个文件名称相同，只是一个是 adp 的后缀名，另一个是 tcl。举个 helloworld 的例子如下：

helloworld.tcl

```
set hello {Hello World!}
```

helloworld.adp

```
Hey, @hello@
```

把以上两个文件放到“C:\aolserver\servers\openacs\www”里，然后再浏览器输入地址“http://localhost:8000/helloworld”回车，不出意外的话你会看见“Hey, Hello World!”。这就算是成功了。

连接信息

关于某个网页的用户连接信息，可以通过 ad_conn 来得到，这是一个很有用的函数，在之后的例子中将被频繁用到。这里先来一个粗略的例子。

connection.tcl

```
set user_id [ad_conn user_id]
set url [ad_conn url]
set session_id [ad_conn session_id]
set IP [ad_conn peeraddr]
```

connection.adp

```
<h2>Basic Connection Information:</h2>

<ul>
<li>User Id: @user_id@</li>
<li>This URL: @url@</li>
<li>This session: @session_id@</li>
<li>IP Address: @IP@</li>
</ul>
```

把这两个文件还是放到上面 helloworld 一样的目录里，然后执行“http://localhost:8000/connection”，便可以看见当前连接的信息。

但是目前的网页看上去相当的丑陋，因为没有任何修饰的东西。现在我们就来改变一下，只需要稍微加上几行，就会大有改观。将 connection.adp 的内容修改为如下，其中加粗为新添加的内容。

```
<master>
<property name="doc(title)">Basic Connection Information
</property>
<h2>Basic Connection Information:</h2>
<ul>
<li>User Id: @user_id@</li>
<li>This URL: @url@ </li>
<li>This session: @session_id@ </li>
<li> IP Address: @IP@ </li> </ul>
```

其中名为 master 的 tag 高速 OpenAcs 该页面应包含头、脚，css 和一些其他的修饰等，使用这个 tag 能使我们建立的网站有统一的风格。名为 property 的 tag 能用于设置多种页面属性，在这个例子里设置了页面的名称。

在页面之间传递变量

为了确保 tcl 文件能够得到需要的信息和参数，OpenAcs 在不同的 adp 和 tcl 文件之间应用了一个简单而强大的“契约系统”来保证数据的传输。这里之所以说是不同的 adp 和 tcl 文件，是因为此种情况下的 tcl 文件接收了来自上一个页面中 adp 文件所传输出来的参数数据。打个比方说，就是一个表单页面将表单数据发送给下一个处理页面。因此这两

者是不同名的页面文件。在这里，还是要举一个具体的例子。

namepage.tcl

```
```

namepage.adp

```
<master>

<form action="display-name-age">
<label for="name">Name</label>
<input type="text" name="name"/><br />
<label for="age">Age</label>
<input type="text" name="age"/><br />
<input type="submit">
</form>
```

display-name-age.tcl

```
ad_page_contract {
 This page will display the name and age entered on the nameage page.
 And this page is role is pretty much just to accept and validate the data
before displaying it.
}{
 age:integer
 name
}
```

display-name-age.adp

```
<master>
<property name="title">Page for @name@</property>
<ul>
<li>Name: @name@</li>
<li>Age: @age@</li>
</ul>
<if @age@ ge 18>
@name@ is an adult.
</if>
<else>
@name@ is still a minor.
</else>
```

依旧是将以上的文件放入到于 helloworld 相同的文件夹中。目前你不用担心真正的部署时该怎么办，因为明天将介绍一块内容。消化一下，确保做到了每一点。

Day 2

我们已经掌握了如何建立页面，现在我们将学习如何创建包和怎样使其在 OpenAcs 中运行起来。OpenAcs 中的包概念，就相当于一个 web 应用程序，或者说一个站点。但它比

站点的意义更为广泛，举例来说对于同一个名为 name 的包，我可以加载其为李雷和韩梅梅两个运行实例，两者互不干扰。具体的在之后的学习中我们会看到。

我的第一个包

虽然即使只是使用一些页面和文件夹便可以完成一个不错的网站设计，但是使用包可以更好地利用到 OpenAcs 的工具集。为证实这点我们将建立一个简单的名为“To Do”的包。

“To Do”所要实现的功能和目标相当简单，以下列出：

- ✓ 用户可以将需要按时完成的任务保存在一个列表中
- ✓ 用户仅可以检查并修改属于他们自己的任务条目
- ✓ 每条任务条目包括如下
  - ➢ 题目
  - ➢ 任务描述
  - ➢ 截止时间
  - ➢ 任务状态

为此我们应给用户提供一个实现以下功能的接口：

- ✓ 能够观察任务列表
- ✓ 在列表中添加新的任务条目
- ✓ 编辑或者删除任务条目

了解了目标任务之后，接下来我们就要着手开始创建这个包了。

对文件系统角度看，一个包其实就是一个包含了特定目录的文件夹，我们可以在“C:\aloserver\servers\openacs\packages\”中找到一系列已经建立并被使用的包，而我们即将建立的包也将会被 OpenAcs 自动放置在这个目录之中。而我们需要做的只是在管理站中按几个连接填一张表而已。

首先我们进入“http://localhost:8000/acs-admin”，点击位于右上角的“Developer's Admin”，在新出现的页面中点击一系列链接中的“Package Manage”。这是我们可以看见一个包含了已创建包的列表，在这个表的底下有一个名为“Create new package”的链接，点击它我们就正式进入创建包的过程了。按照以下的提示来填写该表格：

Package Key	todo
Package Name	To Do List
Package Plural	To Do Lists
Package Type	Application
OpenACS Core	Leave it unchecked
Singleton	Leave it unchecked
Auto-Mount URI	Leave it blank
Package URL	Leave the default
Initial Version	0.1d
Version URL	Leave the default
Summary	A little application to keep track of items on a "To do list"
Description	A little application to keep track of items on a "To do list"
Primary Owner	Leave the default

Primary Owner URL	Leave the default
Secondary Owner	not necessary
Secondary Owner URL	not necessary
Vendor	not necessary
Vendor URL	not necessary

最后保证“Write a package specification file for this package”这项的钩子是被打上的。按下“Create Package”按钮，一个 todo 包就建立好了。注意以上那个列表的第一项 title 的内容是小写的 todo，这与书上错误的写法不一样，因为 OpenAcs 要求一定要小写。我们再去 packages 文件夹看一下，可以发现多出了一个名为“todo”的文件夹，这就是新建立的包。其中的每个文件或者文件夹都有其特定的用途。

todo.info：这个包专有的文件，在其中包含了使该文件夹成为 OpenAcs 包所需的必要信息，包括包名、参数、依赖关系等等，通常你不需要直接对该文件进行编辑，在包管理器中对该报的修改都会间接改变该文件的内容。

sql：该文件夹保存了包的数据模型，初始化定义和针对该数据模型的一些修改等。在其中有两个子文件夹，分别是 oracle 和 postgreSQL，便于编写针对不同数据库的特定 sql 文件。在本文中，将只涉及到 PostgreSQL 部分。

tcl：该文件夹包含了与本工程有关的库文件和一些过程定义等。

www：这里就是我们要防止页面文件的地方了，与之前的例子不同，包文件的页面都是存储在相应包的 www 文件夹中的。

了解了包结构之后，我们要加载已创建好的“todo”包了。加载的意思就是发布关于该包的实例到网络上供客户端浏览和操作。

首先我们进入到“http://localhost:8000/admin/site-map”，可以看见各个已被加载的包，将页面拖拉至底部可以发现一个选择包的表单，选择“To Do List”并单击右侧的“Mount Package”按钮。之后“todo”包就被载入，并出现在上面的已加载包列表中。此时我们还没有创建任何页面，因此无法显示“http://localhost:8000/todo/”。

“To Do”包的数据结构

为了保存“To Do”列表中的信息，我们在数据库中至少应该建立一个表。在该表中，保存了如下的几点信息。

- ✓ 任务的名称
- ✓ 任务描述
- ✓ 任务截止时间
- ✓ 任务状态（停滞，完成，取消等）
- ✓ “to do”列表的创建者或拥有者
- ✓ 建立任务的时间
- ✓ 任务的最后修改时间
- ✓ 属于哪一个包实例（对于同一个包，可以有多个不同的运行实例）

接着就是建立该表的 sql 代码了，我个人的建议是使用“pgAmin III”中的查询工具，与命令行工具 psql 相比，查询工具至少可以允许方便地修改已录入的代码。

```
create table todo_item {
  item_id integer,
  title varchar(200),
  description text,
```

```
    status char(1),
    owner_id integer,
    due_date date default now(),
    creation_date date default now(),
    constraint todo_item_pk primary key (item_id),
    constraint todo_item_fk foreign key references acs_objects(obejct_id)
  };
```

录入完毕后，按 F5 键执行就创建了 todo_item 表，再将执行后的以上代码保存为“ C:\aolserver\servers\openacs\packages\todo\sql\postgresql \todo-create.sql”。这个名称并非胡乱取的，它可以保证当该包被装载到其他的 OpenAcs 中时，系统将自动自动运行该 sql 脚本并生成 todo_item 表。另外还可以建立一个名为“todo-drop.sql”的脚本，该脚本将在 todo 包被卸载的时候自动运行，我们可以在其中执行一些清理工作，比如删除 todo_item 表。

```
  drop table todo_item
```

现在可以试着往这个表里添加一些测试用的数据了。

```
  Insert into todo_item values (nextval('t_acs_object_id_seq'), 'My first item', 'my first iten description', 'p', 568);
  Insert into todo_item values (nextval('t_acs_object_id_seq'), 'My second item', 'my second iten description', 'p', 568);
  Insert into todo_item values (nextval('t_acs_object_id_seq'), 'Another item', 'iten description', 'p', 568);
```

Day 3

这个第三天其实信息量是很大的，所以看不完的话分两天甚至三天看也可以，关键是要吃透，不懂得不可以跳过，多试验下一定要做到弄懂弄明白。在今天的学习中，我们将完成昨天建立的包中的内容，包括实现用户接口使用户能够使用我们的“To Do”网页程序。记住将所有创建的页面相关文件保存到包的 www 文件夹中，只有这样才能正确显示页面。

增加一个编辑页面

接下来，我们要建立一个可以同时用于添加和修改任务条目的页面。虽然使用普通的 html 语法结合一定的 tcl 逻辑也可以完成，但是 OpenAcs 提供了更强大可靠的工具来帮助我们实现这一功能。通过使用 ad_form，我们能够指定表单中的各个单元，针对不同状态需要执行的下一个动作和输入验证等。通过以下的例子我们会掌握 ad_form 的用法，但是想知道更多的话还是应该查阅 OpenAcs 的文档。

首先我们要写一个 adp 文件

todo-ae.adp

```
<master>

<property name="doc(title)">@page_title@</property>
<formtemplate id="todo_item_form"></formtemplate>
```

然后是 todo-ae.tcl，先试用 ad_page_contract 来制定一个传入参数，以便于以后用到。

```
ad_page_contract {
    This page allows the user to add new items to their to do list or edit
existing items.
} {
    item_id:optional
}

set page_title "Add/Edit Todo Item"
set user_id [ad_conn user_id]
```

我们注意到在这里我们声明了一个叫做 item_id 的可选输入参数，通过该参数我们就可以知道当前的网页处于编辑状态还是添加状态。如果我们制订了 item_id，那么当前页面处于编辑状态；反之则处于添加新任务条目的状态。ad_form 对这种情况会进行自动的判断，从而显示出不同的页面形式。

现在我们在 todo-ae.tcl 中接着添加如下代码：

```
ad_form -name todo_item_form -export {user_id} -form {
    item_id:key
    {title:text {label {Task Title}}}
    {description:text(textarea),optional {label {Description}}}
    {due_date:date(date) {label {Due Date:}} {format {MONTH DD YYYY}}}
    {status:text(select) {label Status} {options {
      {{pending} {p}}
      {{Complete} {c}}
      {{Canceled} {x}}
    }}}
}
```

在第一行中，我们调用了 ad_form，“name”参数表示的是表单的 id 号，就是我们在 adp 文件中建立的表单模板就会被其实现。同事我们还“export”了一些参数，说是输出其实用输入来说更容易理解，就是把参数引入到表单中以 hidden 的形式保存起来。

```
ad_form -name todo_item_form -export {user_id} -form {
```

在“-form”之后，我们定义了表单中的组件，首先是一个“key”的组件，通常这是需要添加或者编辑的任务条目的唯一标识符。

```
item_id:key
```

第二个是一个简单的单行文本单元，用来输入任务名称。

```
{title:text {label {Task Title}}}
```

下面的一行表示一个大文本框，用来输入任务表述。

```
{description:text(textarea) {label {Description}}}
```

接下来是一个日期控件，它包含了两个下拉框和一个文本输入单元，分别用来输入年月日，方便和简化了日期的输入过程。同时我们还能指定日期显示的格式。

```
{due_date:date(date) {label {Due Date:}} {format {MONTH DD YYYY}}}
```

最后又是一个下拉框用来选择任务的状态，“options”参数用来指定可选择的内容。

```
{status:text(select) {label Status} {options {
     {{pending}  {p}}
     {{Complete} {c}}
     {{Canceled} {x}}
 }}}
```

输入这些完毕之后，到浏览器中输入“http://localhost:8000/todo/todo-ae”，回车后一切顺利的话就可以看到所期待的页面了。在显示的页面中，OpenAcs 对表单的每一项做了基本的验证，即一定要求填写内容，所以在每项标题的后面又打上了“(required)”标记。如果没有填写内容而按了确定，则会提醒出错。为了让某项变成可选的输入，我们需要对代码做少许修改。

```
{description:text(textarea),optional {label {Description}}}
```

就是在第一段信息的后面紧跟“,optional”。

插入新数据

ad_form 命令还负责处理在根据表单中添加的数据来处理新增加的任务，这是通过“-new_data”选项来实现的，紧跟在该选项之后的代码负责完成这项工作，在这里就是将数据的数据存储到 postgresql 中建立的“todo_item”表中。

在这里顺便提一下，函数甚至代码段作为参数是 tcl 编程的一个明显特点，在别的语言中可能被称作闭包，但这在 tcl 中很早就被频繁地使用了。我们举个例子来说一下它的基本实现。

```
proc do {code while cond} {
    uplevel 1 $code
    uplevel 1 [list $while $cond $code]
}
```

这是一个模拟 C 中“do while”的结构，其中 code 和 cond 参数都是代码段，这里的关键在于“uplevel 1”，这使得以上两个被 do 函数组织成一定执行序列的代码段，其实际执行的上下文环境是在调用 do 函数的代码中。但即使可以像这样随意创建执行结构，但依然与原生命令存在着细微的差异，这一点还是要注意的。比如还是 do 这个函数，如果想在某次循环中直接 return 推出，那么实际退出的只是 do 函数，为此我们需要执行“return -code return”。

言归正传，下面就是“-new_data”后的代码段。

```
-new_data {
  db_dml insert_item "
```

```
      insert into todo_item (item_id, title, description, status, due_date,
owner_id)
      values       (:item_id,       :title,       :description,       :status,
to_date(:due_date,'YYYY MM DD HH24 MI SS'), :user_id) "
   }
```

注意，对于“due_date”我们使用了“to_date”函数就其转换为符合 postgresql 中 date 类型的格式，即包括了年月日和小时、分钟以及秒。

好了，到目前为止“ad_form”函数的整个代码基本就能使用了，我们可以在其中试着插入一些任务数据，之后到通过“pgAmin III”来查看一下“todo_iten”中新增加了的数据。完整代码如下所示。

```
   ad_form -name todo_item_form -export {user_id} -form {
     item_id:key
     {title:text {label "Task Title"} }
     {description:text(textarea),optional {label "Description"}}
     {due_date:date(date) {label "Due Date: "} {format {MONTH DD YYYY}}}
     {status:text(select) {label "Status"}
        {options {{"Pending" "p"} {"Complete" "c"} {"Canceled" "x"}}}}
   } -new_data {
     db_dml insert_item "
       insert into todo_item
         (item_id, title, description, status, due_date, owner_id)
       values
         (:item_id, :title, :description, :status, to_date(:due_date,'YYYY MM
DD HH24 MI SS'), :user_id) "
   }
```

编辑修改任务条目

编辑修改一个已存在的任务条目，我们需要做两件事情。第一，现在该条目中的信息；第二，将修改后的数据写回至数据库表中完成更新。为此，我们要将需要修改的任务条目信息从数据库中提取出来，显示在页面上，这个功能还是由“ad_form”来完成，这一次我们用“-select_query”选项，后接必要的代码段。

```
   -select_query {
     select title,
       description,
       to_char(due_date, 'YYYY MM DD') as due_date,
       status
     from todo_item
     where item_id = :item_id and owner_id = :user_id
   }
```

该选项只有在页面可选参数“item_id”被赋值的情况下才会启用，也就是说我们要显示一个已建的任务条目，就必须先给“todo-ae”页面一个“item_id”值，比如“http://localhost:8000/todo/todo-ae?item_id=1”的地址就是用来显示“item_id”为 1 的任务条目信息的。实际中的具体值是多少，可以通过查看数据库得到。

接下来就是更新部分了，依然还是继续添加“ad_form”，这次使用的是“-edit_data”

选项。基本上这个是与“-new_data”很类似的，只是用处各异罢了。

```
-edit_data {
  db_dml update_item "
    update todo_item
      set title = :title,
      description = :description,
      status = :status,
      due_date = to_date(:due_date,'YYYY MM DD'),
    where item_id = :item_id and owner_id = :user_id "
}
```

至始至终，我们都会看见一些带冒号的字符串，比如上面出现的“:title”、“:description”和“:status”等。事实上这些是占位符，当真实执行的时候，带冒号的字符段会被同名参数的值所替换。之所以不用 tcl 中的“$”号，我觉得有两个原因。第一，是为了延后执行，因为该参数是在运行时才被确定的，因此不能在编写代码时直接赋予；第二，熟悉 tcl 代码的人都知道，在花括号即“{}”中的“$”本身就是延后赋值的，但 OpenAcs 之所以不用这个办法，我想可能是考虑避免给初学者初学者带来困扰，毕竟网页编程中的 tcl 使用者太少了。

按完确定之后

这个是要单独说一下的，因为按完确定之后，并不是就结束了。页面应该跳转至一个至少应该有提示完成信息的页面，而不是原地不动依然处于编辑状态。幸好“ad_form”还为我们提供了一个选线叫“-after_submit”，其后的代码将在按完确定按钮之后执行，我们就在这里做一些跳转等善后工作。

```
-after_submit {
  ad_returnredirect index
  ad_script_abort
}
```

其中“ad_returnredirect index”就是表示跳转到 index 页面，在这里我们还没有建立好 index 页面，所以可能会报错，不过没关系，下面我们马上就将怎么建立。最后另外要提一下的是“ad_form”命令还有一个有用的选项“-mode”，即表示表单的状态，一般有两个选择“edit”和“display”。前者表示表单处于可编辑状态，后者则是表示只读状态。为此，我们再为该选项设置一个页面参数，当从其他页面转到“todo-ae”时可根据实际情况来选择表单的状态。此时“todo-ae.tcl”中的内容如下所示：

```
ad_page_contract {
    This page allows the user to add new items to their to do list or
edit existing items.
} {
    item_id:optional
    {due_date ""}
    {form_mode "edit"}
}

set page_title "Add/Edit Todo Item"
set user_id [ad_conn user_id]
```

```
ad_form -name todo_item_form -export {user_id} -mode $form_mode -form {
    item_id:key
    {title:text {label {Task Title}}}
    {description:text(textarea),optional {label {Description}}}
    {due_date:date(date) {label {Due Date:}} {format {MONTH DD YYYY}}}
    {status:text(select) {label Status} {options {
     {{pending}  {p}}
     {{Complete} {c}}
     {{Canceled} {x}}
    }}}
} -new_data {
     db_dml insert_item "
       insert into todo_item
         (item_id, title, description, status, due_date, owner_id)
       values
         (:item_id, :title, :description, :status,
            to_date(:due_date,'YYYY MM DD HH24 MI SS'), :user_id) "
} -select_query {
     select title,
       description,
       to_char(due_date, 'YYYY MM DD') as due_date,
       status
     from todo_item
     where item_id = :item_id and owner_id = :user_id
} -edit_data {
     db_dml update_item "
       update todo_item
         set title = :title,
         description = :description,
         status = :status,
         due_date = to_date(:due_date,'YYYY MM DD'),
     where item_id = :item_id and owner_id = :user_id "
} -after_submit {
    ad_returnredirect index
    ad_script_abort
}
```

Index 页面

我们将在 index 页面中以列表的形式显示已经创建了的所有任务条目。为此，我们需要使用 OpenAcs 提供的列表创建器，这是一个强大的模板工具，允许我们快捷地建立功能丰富的列表。该命令是“tempalte::list::creae”，在我们这个例子里将只使用到其最基本的功能。

使用列表创建器的方法与之前使用的“ad_form”大同小异，它就是一个过程调用并加

上几个所需要的选项。稍显不同的是，列表创建器需要一个多行数据作为其必要的参数，该多行数据可以通过“db_multiform”得到。

首先，我们还是要建立 adp 文件，即 index.adp。

```
<master>
<property name="doc(title)">@title@</property>
<listtemplate name="todo_list"></listtemplate>
```

之后，是 index.tcl 文件。事实上，对于绝大多数的情况，页面 tcl 文件的开始部分都应该包括定义传入参数的代码，也就是我们在前一天讲到的“ad_page_contract”函数。即使是什么都没有用，最好也写一下，以便于日后修改之用，在这里就是一个空的调用 。

```
ad_page_contract {
 This page will display a list of to do items belonging to the current user.
}{
}
```

此外，我们还要定义两个即将被使用到的参数。

```
set title "My To Do List"
set user_id [ad_conn user_id]
```

现在，我们要真正开始建立列表了，这从调用列表创建函数开始。

```
template::list::create -name todo_list \
  -multirow todo_list_mr \
```

以上这行定义了列表的名称和该列表中包含的数据，该数据保存在“todo_list_mr”之中。之后要给该命令添加必要的选项了，首先是定义列表的各个列元素。每一个列元素的定义使用如下的形式。

```
element_name {
  element_option option_value
  element_option option_value
}
```

反复使用这样的形式就可以创建所需要的列表了，比如 title 列表示任务的名称，可以这样定义。

```
title {
    label "Task"
    link_url_col item_url
    link_html {title "Click to view this item details" }
}
```

要知道每一个参数的意义，可以查看 OpenAcs。这里需要注意的是“link_url_col”，它指定了点击该列表项将会跳转到的页面。完整的列表创建代码如下所示：

```
template::list::create -name todo_list \
 -multirow todo_list_mr \
 -elements {
    title {
       label "Task"
       link_url_col item_url
       link_html {title "Click to view this item details" }
    }
```

```
        due_date_pretty {
          label "Due Date"
        }
        status_text {
          label "Status"
        }
        creation_date_pretty {
          label "Creation Date"
        }
        view {
          display_template "View"
          link_url_col item_url
        }
        delete {
          display_template "Delete"
          link_url_col delete_url
        }
        completed {
          display_template "Mark Completed"
          link_url_col completed_url
        }
        cancel {
          display_template "Cancel"
          link_url_col cancel_url
        }
    }
```

不过列表的创建还没有到此完工，因为我们还要建立用来得到“multirow”的代码。事实上“multirow”是查询数据库的“todo_item”表得到的数据。我们将在“db_multirow”中定一个特定的 sql 查询代码来完成这项工作。

```
    db_multirow -extend {item_url delete_url cancel_url completed_url
status_text } todo_list_mr
    todo_list_mr \
    "select item_id,
        title,
        due_date,
        to_char(due_date, 'Month DD YYYY ') as due_date_pretty,
        creation_date,
        to_char(creation_date, 'Month DD YYYY ') as creation_date_pretty,
        status
     from todo_item
     where owner_id = :user_id"
```

可以看出，在以上命令里还包含了 item_url、delete_url、cancel_url 等 url 参数，它们将和数据库查询出来的数据一起被打包进 todo_list_mr 变量中，该变量就是创建列表要用的“multi_row”了。其中 item_url 很简单，就是在不可编辑状态下显示选

择的任务条目信息，使用以下的代码。

```
set form_mode display
set item_url "todo-ae?[export_vars -url { item_id form_mode }]"
```

将“from_mode”设置为 display 表明接下来显示的表单时不可编辑，如要编辑则需将其设置为 edit。可以看见，我们还用到了一个函数是“export_vars”，在这里其实它是返回来这么一个字符串：

```
item_id=1&&form_mode=display
```

和之前的“todo-ae?”组合成了 http 地址的结尾一段，而 item_id 和 form_mode 就是在 todo-ae.tcl 中最开始部分定义的可传入参数。

除此之外，对于查询得到的 status 状态值，我们还要做一些处理。因为在定义数据库时，status 的类型为“char(1)”，即是一个字符长度的单位，但是在实际显示的时候我们需要跟确切地显示出该状态的意义。所以，在这里需要做一个转换，通过 status 的值来得到表单定义中实际所需要的 status_text 值。

```
switch $status {
  p {set status_text "Pending"}
  c {set status_text "Completed"}
  x {set status_text "Canceled"}
  default {set status_text "Unknown" }
}
```

与 item_url 相类似的，delete、completed 和 cancel 的链接也是很简单就可以写出来的，不过实际要跳转的页面我们呆一会再写，暂且搁置一边。

```
set return_url [util_get_current_url]
set delete_url "todo-delete?[export_vars -url {item_id return_url}]"
if { $status != "c" } {
  set new_status completed set completed_url "todo-update-item?[export_vars -url {item_id new_status return_url}]"
}
if { $status != "x" } {
  set new_status canceled set cancel_url "todo-update-item?[export_vars -url {item_id new_status return_url}]"
}
```

在传入参数中除了 item_id，我们还看见定义了 return_url，也就是说我们在建立跳转页面的时候应该在 tcl 文件开始部分的 ad_page_contract 函数里加入以上这两个参数。

现在默认页面基本上建立好可以使用了，接着我们便可以在浏览器里打入“http://localhost:8000/todo”的地址，回车。正常的话就可以看见任务列表了。这里要提醒一下，我试了很多次一直提醒无法找到 index 页面，后来我使用以下这个页面才正常显示，而且之后上面的那个地址也可以使用了，我估计这是一个 bug。

http://localhost:8000/tdo//

添加排序

（这第三天的任务还真是繁重啊，真不知道写这个 tutorial 的老兄是怎么想的，到了第三天跟打了鸡血似的，忒兴奋了吧。）

现在，为任务列表的各个列添加排序显示的功能，我们需要完成如下几个工序，代码的添加依然是在 index.tcl 中完成的。

首先，在代码开始处定义传入参数的部分，新添加一个可选传入变量“orderby”，该变量指定根据哪一列进行排序。

```
    ad_page_contract {
      This page will display a list of to do items belonging to the current
user.
    } {
      orderby:optional
    }
```

接着，在列表创建器“ad_form”函数中加入选项“-order_by”，以及一段代码。其中任一列的格式为“<element name> { <order by clause> }”。当选择某一列时，实际的排序行为由该列所对应的“<order by clause>”确定。

```
    -orderby {
      title {orderby title}
      due_date_pretty {orderby due_date}
      status_text {orderby status}
      creation_date_pretty {orderby creation_date}
    }
```

第三，修改“db_multirow”函数中的代码，使其能够支持排序。在这里我不得不臆测一下“index.tcl”结构，因为 ad_form 函数是定义在 db_multirow 之前的，所以也就是说表单的建立完后，才开始提取数据库参数。这么一来，就很不合情理了，既然表达的建立是要用到数据库数据的，可偏偏提取数据的行为发生在其后，岂不是自相矛盾了。所以，我认为“ad_form”的行为其实是一个滞后的过程，这可以从其中的“-multirow”参数绑定的是变量名称而非变量本身看出些许端倪。也就是说在完成了“index.tcl”的运算之后，OpenAcs 在将数据交给“index.adp”时做了数据替换，我个人以为这些过程完全可以透明，这并不会给 OpenAcs 本身增加复杂度。

言归正传，接着完成有关于“db_multirow”函数的排序代码，先做一些预备工作。

```
     if {[exists_and_not_null orderby]} {
       set orderby_clause \
          "ORDER BY [template::list::orderby_clause -name todo_list]"
     } else {
       set orderby_clause "ORDER BY due_date asc"
     }
```

这是给“order_by”变量设置一个默认值。使其在默认情况下，一般是页面跳转到“index“时，按照任务截止日期进行排序。

```
    db_multirow -extend {
      item_url    delete_url    cancel_url    completed_url    status_text    }
todo_list_mr todo_list_mr \
      "select item_id,
         title,
         due_date,
         to_char(due_date, 'Month DD YYYY ') as due_date_pretty,
         creation_date,
         to_char(creation_date, 'Month DD YYYY ') as creation_date_pretty,
         status from todo_item
```

```
        where owner_id = :user_id
        $orderby_clause " \
        {
         ...
        }
```

完成这些之后，我们发现“http://localhost:8000/todo/”页面的任务列表各列的标题变为了链接状态，单击各个标题就会进行不同的排序。

添加一个增加任务条目用的按钮

该按钮将出现在列表的上方靠左一些的位子。在设计表单的时候，一般我们都应该建立这么一个方便用户来影响表单行为的按钮。在这个例子里，我们只需要增加一个用来添加新任务条目的按钮即可，该按钮将跳转到“todo-ae”页面上。

事实上，“ad_form”已经预计到了需要添加按钮的情况，并且视按钮作为表单的必要组成部分。因此，我们只需要在“ad_form”中添加一个如下形式的必要选项即可。

```
    -actions {
      <Action label> <Action URL> <Action title text>
      <Action label> <Action URL> <Action title text>
    }
```

在本例子中我们将使用如下的实际代码：

```
    -actions {
      "Add New Task" "todo-ae" "Click here to add a new item to the list"
     }
```

到目前为止，我们已经使用 OpenAcs 工具建立了一个基本但却有足够功能的“to do”列表程序。以下就是“index.tcl”的最终代码。

```
    ad_page_contract {
       This page will display a list of to do items belonging to the
current user.
    } {
       orderby:optional
    }

    set page_title "My To Do List"
    set user_id [ad_conn user_id]
    template::list::create -name todo_list \
       -multirow todo_list_mr \
       -elements {
          title {
             label "Task"
             link_url_col item_url
             link_html {title "Click to view this item details" }
          }
          due_date_pretty {
             label "Due Date"
          }
```

```
            status_text {
                label "Status"
            }
            creation_date_pretty {
                label "Creation Date"
            }
            view {
                display_template "View"
                link_url_col item_url
            }
            delete {
                display_template "Delete"
                link_url_col delete_url
            }
            completed {
                display_template "Mark Completed"
                link_url_col completed_url
            }
            cancel {
                display_template "Cancel"
                link_url_col cancel_url
            }
        } -orderby {
            title {orderby title}
            due_date_pretty {orderby due_date}
            status_text {orderby status}
            creation_date_pretty {orderby creation_date}
        } -actions {
            "Add New Task" "todo-ae" "Click here to add a new item to the
list"
        }

    if {[exists_and_not_null orderby]} {
        set orderby_clause "ORDER BY [template::list::orderby_clause -name
todo_list]"
    } else {
        set orderby_clause "ORDER BY due_date asc"
    }

    db_multirow -extend { item_url delete_url cancel_url completed_url
status_text } todo_list_mr todo_list_mr \
        "select item_id,
           title,
           due_date,
```

```
          to_char(due_date, 'Month DD YYYY ') as due_date_pretty,
          creation_date,
          to_char(creation_date, 'Month DD YYYY ') as creation_date_pretty,
          status
        from todo_item
        where owner_id = :user_id $orderby_clause
       " {
          set form_mode display
          set item_url "todo-ae?[export_vars -url { item_id form_mode }]"
          switch $status {
             p {set status_text "Pending"}
             c {set status_text "Completed"}
             x {set status_text "Canceled"}
             default {set status_text "Unknown" }
          }

          set return_url [util_get_current_url]
          set    delete_url    "todo-delete?[export_vars    -url    {item_id
return_url}]"

          if { $status != "c" } {
             set new_status completed
             set    completed_url    "todo-update-item?[export_vars    -url
{item_id new_status return_url}]"
          }
          if { $status != "x" } {
             set new_status canceled
             set  cancel_url  "todo-update-item?[export_vars  -url  {item_id
new_status return_url}]"
          }
       }
```

其他的一些 url

从列表中可以看出，除了已经建立的添加功能之外，还有删除和更新功能及相应的页面需要建立。我们首先要建立删除任务条目的页面“todo-delete”，并添加相关的内容。以下是“todo-delete.tcl”的代码。

```
  ad_page_contract {
   This page will delete an item from the todo list package and then return
the user
   to the specified return URL.
  } {
    item_id
    return_url
  }
```

```
   set user_id [ad_conn user_id]
   db_dml delete_item "delete from todo_item where item_id = :item_id and
owner_id = :user_id"
   ad_returnredirect $return_url
```

在“todo-delete”中，我们并不需要 adp 文件。这是因为，“todo-delete”完成的是一个后台的操作不需要显示部分，因此当删除工作完成之后我们只需要将页面重定向到某个合适的已建页面即可，为此，我们在 tcl 文件开始的传入参数部分设置了一个名为“return_url”的参数，用以在完成删除工作之后，使用“ad_returnredirect”命令来具体完成重定向工作。接下来的更新页面及其功能也是类似的形式。“todo-status-upadte.tcl”：

```
   ad_page_contract {
    This page will update an item's status from the todo list package and
then return the user
    to the specified return URL.
   } {
     item_id
     new_status
     return_url
   }

   set user_id [ad_conn user_id]
   switch $new_status {
      "pending"   {set status_code p}
      "completed" {set status_code c}
      "canceled"  {set status_code x }
      default     {set status_code p }
     }

   db_dml update_status "update todo_item
                      set status = :status_code
                      where item_id = :item_id
                      and owner_id = :user_id"

   ad_returnredirect $return_url
```

为 todo 包添加 Tcl 的 API

在介绍包结构时，我们提到了除了在包中建立页面之外，我们还可以在包目录下的 tcl 文件夹中建立包专有的过程。在“todo”包中，我们要建立一个名为“todo.tcl”的过程文件。

```
   ad_library {
      Procs for the To Do list package.
   }
```

```
namespace eval todo {}

ad_proc -public todo::get_status_label { status } {
  switch $status {
    p { set status_text "Pending" }
    c { set status_text "Completed" }
    x { set status_text "Canceled" }
    default { set status_text "Unknown" }
  }
  return $status_text
}

ad_proc -public todo::get_status_code { status_text } {
  set status_text [string tolower $status_text]
  switch $status_text {
    "pending"   {  set status_code p }
    "completed" { set status_code c }
    "canceled"  { set status_code x  }
    default     {  set status_code p  }
  }
  return $status_code
}
```

最开始的三行使用了“ad_library”过程来定义该文件为一个新的库，并提供了必要的文档信息。接下来我们为所需要建立的过程定义了一个新的命名空间，以防止与 OpenAcs 的过程名冲突。之后我们在该命名空间中建立了两个过程，分别是“get_status_label”和“get_status_code”。在完成库文件的修改后，我们要在管理页面中手动通知 OpenAcs 进行组件的更新。也就是单击“http://localhost:8000/acs-admin/apm”中相应包的“reload changed”链接，之后新建或修改后的库将被载入到系统之后可供我们使用，并且页面将跳转并显示新被载入的文件。最后，我们要在页面中具体应用这些过程来简化代码的编写。我们需要修改两处地方，第一个是将“index.tcl”中“db_multirow”函数中 siwtch 的一段改为：

```
set status_text [todo::get_status_label $status ]
```

第二个修改是将“todo-status-upadte.tcl”中的 switch 改为如下：

```
set status_code [todo::get_status_code $new_status]
```

Day 4

在第四天里我们将着重学习和应用 OpenAcs 中的对象概念和系统。根据我个人的理解，ACS 对象其实就是抽象出不同存储数据共性的部分，并将其存储入特定的表中进行统一管理，同时 OpenAcs 为这些共性的元素提供了一组可公用的操作函数。这么做的好处在与我们可以将更多的精力集中在业务逻辑上，而不用管一些比如数据建立时间、所有人等琐碎的细节。以下是一些可能的共性：

- 数据库条目的所有者信息
- 数据库条目的所有者权限
- 部分条目可能需要版本控制
- 提示
- 相关的评论、排名等服务
- 同站点中不同组使用同包的不同实例，因此产生的相关信息

如何建立对象

使用对象系统很简单，但是我们在建立数据模型的同时还需要首先做一些必要的准备工作。

1. 新建对象类型
   - 对象类型就如同面向对象编程语言中的类
2. 为我们的网页应用程序定义一个表
   - 该表格保存了与本应用程序相关的对象信息
   - 该表的主键必须同时作为外键与“acs_objects”中的主键“object_id”相关联
3. 定义与应用程序相关的过程
   - 过程可以是 tcl 函数或者在数据库中定义的过程
4. 修改程序使其具有“对象相关性”
   - 为此我们需要对以上已建立的页面 tcl 文件进行一些小范围内的修改。

定义数据模型

首先，我们将在数据库中删除已建立的“todo_item”表。

```
drop table todo_item;
```

之后，在“PgAmin III”的查询工具中运行以下代码，并将该代码存储在包的“sql\postgresql”文件夹下（之前已详细说明）。

```
create table todo_item (
  item_id integer,
  title varchar(500),
  description text,
  status char(1),
  due_date date default now(),
  constraint todo_item_pk primary key (item_id),
  constraint todo_item_fk foreign key (item_id) references
acs_objects(object_id)
);
```

```
create function inline_0 ()
returns integer as $$
  begin
    perform acs_object_type__create_type (
      'todo_item',
      'To Do Item',
      'To Do Items',
      'acs_object',
      'todo_item',
      'item_id',
      'todo',
      'f',
      null,
      null
    );

    return 0;
  end;
$$ language 'plpgsql';

select inline_0 ();
drop function inline_0 ();
```

请按照以上的代码，重新输入一边，这样有助于理解。另外，与原文中不同的是，在定义函数时我使用了“$$”取代了单引号的使用，使代码清晰易懂（使用单引号实在是一个外行的写法）。还有在 plpgsql 中，字符串是由单引号包围的而不是双引号，这一点要注意，也就是说如果按照原文的代码是无法完成创建的。这里建立的“inline_0”临时函数，其实完全是一个卖弄的噱头，大可以用 select 代替。

按 F5 之后，一切顺利的话，我们可以在“acs_object_types”表中查看到刚建立的“todo_item”对象。以上代码中使用到了一个 OpenAcs 自带的 plpgsql 过程“acs_object_type__create_type”，通过“PgAmin III”我们可以查看它的具体定义过程，经常查阅这些函数有助于我们更好地理解其中的实现。

我们还要建立两个属于个“to do”包的 plpgsql 函数，“todo_obj_item__new”和“todo_obj_item__delete”，分别用于添加和删除“todo_item”对象。

```
create or replace function todo_item__new (
  integer,
  varchar,
  text,
  char,
  integer,
  date,
  varchar,
  integer
) returns integer as $$
  declare
```

```
    p_item_id alias for $1;
    p_title alias for $2; -- default null
    p_description alias for $3; -- default null
    p_status alias for $4; -- default null
    p_creation_user alias for $5; -- default null
    p_due_date alias for $6;
    p_creation_ip alias for $7; -- default null
    p_context_id alias for $8; -- default null

    v_id integer;
    v_type varchar;
  begin
    v_type := "todo_item";

    v_id := acs_object__new(
      p_item_id,
      v_type,
      now(),
      p_creation_user,
      p_creation_ip,
      p_context_id::Integer,
      true
    );

    insert into todo_item
      (item_id, title, description, status, due_date)
    values
      (p_item_id, p_title, p_description, p_status, p_due_date);

    return v_id;
  end;
$$ language 'plpgsql';

create or replace function todo_item__delete(integer)
returns void as $$
  declare
    p_item_id alias for $1;
  begin
    delete from todo_item where item_id = p_item_id;
    perform acs_object__delete(p_item_id);
  end;
$$ language 'plpgsql';
```

将查询工具中的原有代码删除后，输入以上两个过程的代码，再按 F5，我们便成功地在 postgreql 中建立了属于 OpenAcs 数据库的两个过程。

修改页面文件

我们已经将数据模型转换为了对象模式，现在我们还要修改应用程序部分，使其能使用这种对象系统。我们从修改“todo-ae”页面开始，经修改之后将要做到以下的几点：

- ✓ 以对象的形式插入新条目
- ✓ 以对象的形式修改条目
- ✓ 当提取并显示信息时，显示的是对象的信息

修改提取数据的代码

为此，对于修改下“todo-ae.tcl”中的“ad_form”命令，我们首先修改提取部分的代码，即“-select_query”之后的代码。

```
select todo.title,
  todo.description,
  to_char(todo.due_date, 'YYYY MM DD') as due_date,
  todo.status
from todo_item todo,
  acs_objects obj
where todo.item_id = :item_id
  and obj.object_id = todo.item_id
  and obj.creation_user = :user_id
  and obj.context_id = :package_id
```

可以看到，其中最主要的变化在于从“acs_objects”中提取了一些信息。其中“context_id”表示条目被创建时所属的包实例，“creation_user”表示条目的创建者。使用这两个信息的目的在于确定取出的条目属于当前用户和现所在的包实例。

具体包实例的创建过程在之前有说到过，就是加载包的过程，只是之前我们并没有在加载的过程中命名，因此默认的实例名与包名相同。

现在我们再次进入“http://localhost:8000/admin/site-map”，这次我们还是选择加载“To Do List”，但在 URL 空格内填入“todo-school”，然后按确定就新建了一个 todo 实例。依次地，我们再尝试着建立一些诸如“todo-work”、“todo-projects”等。这说明对已同一个包，我们能建立任意多的实例。

点击任意实例行右侧的“permission”链接，还能看到用户针对于这个包实例的权限。在该页面中如果再点击“Don’t inherit Permission’s from Main Site”链接，则将移除该站点从管理站集成下来的相关权限。

修改添加新条目的代码

就是修改“-new_data”选项之后的代码为如下所示：

```
-new_data {
  set context_id [ad_conn package_id]
  set new_object_id [ db_exec_plsql do_insert {
    select todo_item__new (
      :item_id,
      :title,
      :description,
      :status,
      :user_id,
```

```
        to_date(:due_date, 'YYYY MM DD HH24 MI SS'),
        :ip_address, :context_id
      );
    }]
  }
```

可以看出来，对于 pgplsql 定义的过程，我们使用了“db_exec_plsql”函数来调用。这跟单纯使用 sql 有所不同，因为在调用 sql 进行数据操作时我们使用的是“db_dml”。

修改更新条目的代码

更新涉及到修改与对象有关的信息、跟踪该对象在何时被谁进行了修改。

```
  -edit_data {
    # update the information on our table
    db_dml todo_item_update "
      update todo_item
        set title= :title,
        description = :description,
        status = :status,
        due_date = to_date(:due_date, 'YYYY MM DD HH24 MI SS')
      where item_id = :item_id"

    # update the last modified information on the object
    db_exec_plsql to_do_list_obj_item_object_update {
      select acs_object__update_last_modified(
        :item_id,:user_id,:ip_address
      )
    }
  }
```

我们使用了“acs_object__update_last_modified”函数里自动记录条路最新的修改时间，以及由来自哪个 IP 地址的哪位用户进行的此次修改。

最后，经过修改之后的“todo-ae.tcl”代码为如下形式：

```
  ad_page_contract {
     This page allows the user to add new items to their to do list or
edit existing items.
  } {
     item_id:optional
     {due_date ""}
     {form_mode "edit"}
  }

  set page_title "Add/Edit Todo Item"
  set user_id [ad_conn user_id]
  set package_id [ad_conn package_id]
  set ip_address [ad_conn peeraddr]
```

```
ad_form -name todo_item_form -export {user_id package_id} \
  -mode $form_mode -form {
    item_id:key
    {title:text {label {Task Title}}}
    {description:text(textarea),optional {label {Description}}}
    {due_date:date(date) {label {Due Date:}} {format {MONTH DD YYYY}}}
    {status:text(select) {label Status} {options {
    {{pending}  {p}}
    {{Complete} {c}}
    {{Canceled} {x}}
    }}}
} -new_data {
    set context_id [ad_conn package_id]
    set new_object_id [db_exec_plsql do_insert {
    select todo_item__new (
                  :item_id,
                  :title,
                  :description,
                  :status,
                  :user_id,
                  to_date(:due_date, 'YYY MM DD HH24 MI SS'),
                  :ip_address,
                  :context_id
                  );}]
} -select_query {
    select todo.title,
    todo.description,
    to_char(todo.due_date, 'YYYY MM DD') as due_date,
    todo.status
    from todo_item todo, acs_objects obj
    where todo.item_id = :item_id
    and obj.object_id = todo.item_id
    and obj.creation_user = :user_id
    and obj.context_id = :package_id
} -edit_data {
    db_dml todo_item_update {
    update todo_item
    set title = :title,
    description = :description,
    status = :status,
    due_date = to_date(:due_date, 'YYYY MM DD HH24 MI SS')
    where item_id = :item_id
    }
    db_exec_plsql to_do_list_obj_item_object_update {
```

```
        select
          acs_object__update_last_modified(:item_id, :user_id, :ip_address)
        }
    } -after_submit {
        ad_returnredirect index
        ad_script_abort
    }
```

接下来我们还要修改“index.tcl”中的代码，使其能够使用对象系统。

修改 index.tcl

index 页面显示任务列表，并且允许添加新的任务。我们也只需少许修改就能使其使用新的数据模型。

第一个变化发生在“db_multirow”的命令中，需要修改一下查询数据库的 sql 语句，添加对“acs_objects”表的查询。

```
    select todo.item_id,
      todo.title,
      todo.due_date,
      to_char(todo.due_date, 'Month DD YYYY ') as due_date_pretty,
      obj.creation_date,
      to_char(obj.creation_date, 'Month DD YYYY ') as creation_date_pretty,
      todo.status
    from todo_item todo, acs_objects obj
    where obj.context_id = :package_id
      and obj.creation_user = :user_id
      and obj.object_id = todo.item_id
    $orderby_clause
```

然后还要修改排序部分的代码，其实只是添加了表名前缀。

```
    -orderby {
      title {orderby todo.title}
      due_date_pretty {orderby todo.due_date}
      status_text {orderby todo.status}
      creation_date_pretty {orderby obj.creation_date}
    }
```

与书上有所不同的是，我们可能还需要修改生成“order_clause”的代码，至少我在试验的时候在此处发生了报错。因为需要显式地标明表名前缀。

```
    if {[exists_and_not_null orderby]} {
        set orderby_clause "order by [template::list::orderby_clause -name
todo_list]"
    } else {
        set orderby_clause "order by todo.due_date asc"
    }
```

到此为止，我们修改完了所有的代码，使站点使用了 OpenAcs 的对象系统，只是没有增加任何其他新的性能，与原来完全一致。

整合其他服务

通过使用对象系统，我们还能整合 OpenAcs 提供的一些有用服务。在这里我们将举两个服务例子。第一个是有关于评论的，虽然我们可以自己建立数据模型再添加代码来完成这个任务，但是 OpenAcs 提供了更简单易用的方案。为此，我们要使用到“General Comments”服务，该服务为每个 ACS 对象提供了相关的评论保存设置等功能，而我们要做的只是仅仅添加少许代码。

在此之前还需要确定是否已经安装了该评论服务，我们首先要进入链接“http://localhost:8000/acs-admin/install”，在“Installed Packages”标签下列出了所有已安装的服务。如果是使用了 Windows 的 OpenAcs 安装包，该服务是默认已被安装了的，但如果使用了其他的安装办法而导致该服务未被安装，那么则应点击列表上方的“Install from Repository”链接，在跳转出来的页面中有大量的服务可供选择，我们勾上“General Comments”前的选择方框，再点击接近页面底部的“Install or Upgrade checked application”。接下来的是一个用于确定安装的界面，我们点击“Install above package”，安装完毕之后点击“Click here to restart your server now”来重启 OpenAcs 服务。

为任务条目添加评论服务

当我们以“display”模式显示任务条目的具体信息的同时，也要显示关于该条目的评论；而在“Edit”模式下，评论将被隐藏。为此我们需要增加用于添加评论的链接和显示评论的部分。在“todo-ae.tcl”中“ad_form”命令之后，添加如下代码：

```
if { [string equal $form_mode "display"] } {
  set comment_add_url \
    "[general_comments_package_url]comment-add?[export_vars {
      { object_id $item_id }
      { return_url [util_get_current_url]} }]"
  set comments_html \
   [general_comments_get_comments \
     -print_content_p 1 $item_id [util_get_current_url]]
}
```

以上代码的解析如下：

- ✓ 首先，确定任务条目具体信息显示页面是否处于“display”模式，如果是则可以开始提取相关的评论。
- ✓ 第二，增加“添加评论”的链接。
  - ● 我们使用“general_comments_package_url”命令来得到操作评论的页面地址，在 OpenAcs 中应尽量避免使用完全地址。
  - ● “export_vars”命令将参数已特定地格式输出用于组成具体的 url 地址，这在前面具体说明过
- ✓ 最后，提取并显示与当前任务条目相关的评论。

我们还需要修改“todo-ae.adp”文件，添加评论部分的 html 代码。

```
<if @form_mode@ eq "display">
  <a href="@comment_add_url@">Add a comment</a>
  <p>
  @comments_html;noquote@
</if>
```

以上代码中，我们再次确定了页面的状态，只有在“display”模式下，才会显示添加条目用的链接并显示已有的评论。注意到我们是如何使用评论变量的，在这里使用了“@comment_html;noquote@”，之所以使用 noquote 是为了使数据以 html 结构被浏览器解析，避免错误显示如下形式的文字。

```
<h4>My Comment</h4> Yes this is my comment!
```

许可

使用许可服务可能是我们采用 ACS 对象系统能得到的最大好处了，它能很方便地为我们的新建站点整合如 OpenAcs 许可系统。接下来，我们将建立一个小测试来试验该系统。

首先除了 OpenAcs 内置的“admin@acme.com”用户之外我们还要另外建立一个用户，通过点击以下的链接“http://localhost:8000/acs-admin/users/user-add”，填写相关资料就可以建立新的账户。系统会要求发送一封邮件给新建的用户，但是由于没有配置好邮件系统，发送会遭遇失败，但这不影响新账户的建立。

然后是修改代码，使只有被选择的任务条目的创建者才能读取和修改该条目。在“todo-ae.tcl”中，我们要修改的是“ad_form”中选项“-new_data”之后的代码。将以下代码插入到对象建立之后。

```
permission::toggle_inherit -object_id $new_object_id
permission::grant -party_id $user_id \
    -object_id $new_object_id \
    -privilege "general_comments_create"
```

第一行代码用于收回新建对象从“todo”包中继承下来的许可权限。第二个命令赋予该对象的创建者拥有添加评论的许可。

现在再在“todo-ae.tcl”的开始部分，即“ad_page_contract”命令之后，添加许可检测代码。

```
if { [exists_and_not_null item_id] \
  && [acs_object::object_p -id $item_id]} {
  if {[string equal $form_mode "edit"]} {
    permission::require_permission -object_id $item_id -privilege write
  } else {
    permission::require_permission -object_id $item_id -privilege read
  }
}
```

上面这段代码使得用户必须要有一定的许可权限才能对对象进行读写操作。例如当任务条目处于“display”状态，则需要用户有“read”的许可；而处于“edit”状态，则需要用户有“write”的许可。现在如果我们退出“admin@acme.com”账户，换用上面新建的账户重新登录，会发现 todo 页面没有了之前建立的任务条目。这是因为新账户不是这些条目的创建者，因此没有查看的许可。

另外，我们还要在“todo-delete.tcl”和“todo-update_status.tcl”为对象的删除和更新功能添加许可检测。添加的位置依然是在文件的“ad_page_contract”命令之后。以下加粗的部分就是新添加的许可验证代码。

todo-delete.tcl

```
ad_page_contract {
  This page will delete, an item from the todo list package and then
return the user to the specified return URL.
```

```
} {
  item_id return_url
}

permission::require_permission -object_id $item_id -privilege delete

set user_id [ad_conn user_id]
db_transaction {
  db_exec_plsql delete_item { select todo_item__delete ( :item_id ) }
}
ad_returnredirect $return_url
```

todo-update-status.tcl

```
ad_page_contract {
  This page will update an item's status from the todo list package and then return the user to the specified return url.
} {
  item_id new_status return_url
}

permission::require_permission -object_id $item_id -privilege write

set user_id [ad_conn user_id]
set status_code [todo::get_status_code $new_status]
db_transaction {
  db_dml update_status "update todo_item set status = :status_code where item_id = :item_id"
}
ad_returnredirect $return_url
```

这里修改了很少部分的代码，也仅仅达到了许可控制的目的。现在可以刷新页面，并尝试着用不同的账户登录来操作“to do list”。

本文对象系统介绍也到此为止了，它是一个强大的工具并且整合了其他的 OpenAcs 服务为我们提供了大量不同的功能，在开发的过程中应该多加利用。