使用 Ruby on Rails 在 Tivoli Directory Server V6.0 内创建 LDAP 目录服务

用 Net::LDAP Ruby gem 添加、更新、搜索和删除目录项

级别： 中级

Deepak Vohra, Web 开发人员, 独立顾问

2009 年 6 月 01 日

在本文中，学习如何使用 Net::LDAP 库在 IBM® Tivoli® Directory Server V6.0 目录服务器创建一个 LDAP 目录服务，亲历用 Net::LDAP Ruby 库添加、修改、搜索和删除目录项的各个示例。以 Tivoli Directory Server V6.0 作为后端数据库创建一个 Ruby on Rails 应用程序。

简介

Tivoli Directory Server 是一个基于 LDAP V3 的目录服务器，可用来创建目录服务。所谓的目录服务是指一个能存储、检索和修改网络资源信息的应用程序。LDAP 是一种用于访问目录服务的轻量协议，建立在目录项的基础之上。一个项是一组属性，由全局惟一的专有名称（DN）标识。每个目录项属性都具有一个类型和一个或多个值。在本文中，学习如何使用 Net::LDAP 库在 Tivoli Directory Server V6.0 目录服务器创建一个 LDAP 目录服务，用 Tivoli Directory Server V6.0 作为后端数据库创建一个 Ruby on Rails 应用程序。

先决条件和系统需求

本文要求您对 LDAP 和脚本语言有基本的理解。本文主要面向的是对 LDAP 有基本认识并需要使用 Ruby 脚本语言编写目录服务的读者。为了进行本文中的示例操作，必须要安装 Tivoli Directory Server V6.0。下载 本文中的这个示例 LDAP 应用程序。

有关 Ruby on Rails

Ruby on Rails 是一种开源 Web 框架，用于基于数据库开发 Web 应用程序。Ruby on Rails 提供了开发的简便性、对 Ajax 的支持以及 Model-View-Controller（MVC）框架。Ruby 是一种解释性的、面向对象的脚本语言。*解释性* 意味着 Ruby 应用程序的运行无需首先对之进行编译。Ruby 内的变量不具有类型；一个 Ruby 变量可以包含任何类型的数据。Ruby 内的变量也无需变量声明。作为一种面向对象的语言，Ruby 具有诸如类、方法和继承之类的特性。

要开发一个 Ruby on Rails 应用程序，必须具备 Web 服务器和数据库。Rails 包括一个内置的 Web 服务器，称为 WEBrick。Rails 默认配置有 MySQL 数据库。Rails 是如下这些子项目的综合。

***Model* — 活动记录（Active Record）**
: Model 模型化 MVC 应用程序内的业务对象。在 Rails 内，模型通常基于活动记录模式，该模式提供了业务对象和数据库间的对象关系映射（ORM）。

***View* — 动作视图（Action View）**
: *View* 是 Ruby on Rails Web 应用程序的用户界面。视图由 RHTML 或 RXML 呈现。RHTML 是 Ruby 嵌入的 HTML，RXML 则是 Ruby-生成的 XML。视图内包含对控制器内定义的方法（动作）的链接，控制器动作由这些链接调用。

Controller — 动作控制器（Action Controller）
: Controller 设置视图所需的实例变量并呈现一个视图。 控制器是一个类，它扩展 `ApplicationController` 类并包含动作（方法）。控制器使用公共方法（动作）集成模型和视图。

模型提供数据，控制器提供业务逻辑来处理数据，视图展示数据。

MVC Rails 应用程序包含如下的 Ruby、RHTML 和配置文件：

- app/views 目录内的 View 模板（.rhtml 文件）
- app/models 目录内的 Model 类
- app/controllers 目录内的 Controller 类
- config 目录内的数据库配置文件（database.yml）

Ruby on Rails 提供用来生成模型和控制器脚本的命令。常用的 Ruby on Rails 命令如下所示。

表 1. 常见 Ruby on Rails 命令

命令（变量以斜体表示）	描述
`rails` *`applicationname`*	创建具有特定名称的 Rails 应用程序
`ruby script/server`	启动 http://localhost:3000 处的 Ruby Rails Web 服务器 WEBrick
`ruby script/generate model` *`modelname`*	生成具有特定模型名的一个 model 类
`ruby script/generate controller` *`controllername controlleraction1 controlleraction2`* ...	生成具有特定名称的一个 controller 类，如指定，还会生成控制器动作；还会生成对应于控制器动作的 view 模板（.rhtml 文件）

Ruby on Rails 提供各种 Ruby 库来开发 LDAP 应用程序。比如：

- Ruby/ActiveLDAP 提供一个面向对象的 LDAP 接口，并且 LDAP 项映射到 Ruby 对象。
- Net::LDAP 是一个具有 LDAP 支持的 Ruby 库，并且支持 LDAP 客户机特性。

在本文中，您将学习使用 Net::LDAP 库在 Tivoli Directory Server V6.0 目录服务器内创建一个目录服务。此外，您还将使用 Net::LDAP Ruby 库添加、修改、搜索和删除一个目录项。

安装

本节展示如何安装 Rails 框架 Ruby 以及 RubyGems。RubyGems 是标准的 Ruby 包管理器，与 Ruby 应用程序和库联合使用。此外，您还将安装 Net::LDAP，它是面向 LDAP 的 Ruby 类库。示例应用程序使用的是 Windows® 平台。安装的顺序是先安装 Ruby，然后安装 Rails，最后安装 Net::LDAP。

基本的 Ruby 安装

要安装 Ruby 和 RubyGems 的最新版本：

1. 下载 Ruby Windows Installer 应用程序。本文用 Ruby V1.8.5-21 开发这个示例目录服务应用程序。
2. 双击 **ruby185-21.exe** 应用程序。启动 Ruby Setup Wizard。
3. 单击 **Next** 并接受许可协议。选择安装默认组件，包括 RubyGems 包管理器，并单击 **Next**。
4. 指定 Ruby 安装的目录（默认为 c:/ruby）并单击 **Next**。
5. 单击 **Install**。Ruby 和 RubyGems 就安装好了。

Rails

安装 Rails：

1. 更改目录（Cd）到 c:/ruby 目录，Ruby 将安装在此目录内。
2. 运行如下命令来安装 Rails 和依赖项：`c:/ruby>gem install rails --include-dependencies`。

Net::LDAP 包

用如下代码安装 ruby-net-ldap gem：

```
C:/ruby>gem install ruby-net-ldap
```

Net::LDAP 类提供了 LDAP 客户机协议的一种 Ruby 实现。Net::LDAP 类可被用来进行绑定、搜索、添加、修改、删除和重命名操作。Net::LDAP 类方法如表 2 所示。

表 2. Net::LDAP 类方法

方法	描述
add(args)	添加一个新目录项。参数为： • :dn — 新目录项的 DN（专有名） • :attrs — 新目录项的属性，作为一个 hash 指定。
add_attribute(dn, attribute, value)	向属性添加一个值。如果所指定的属性还没有定义，就会创建一个新属性。
authenticate(username, password)	指定 LDAP 服务器的身份验证凭证。
bind(auth=@auth)	连接到 LDAP 服务器并基于在 open 或 new 方法内指定的身份验证凭证请求身份验证。如果与 LDAP 服务器的连接建立，就会返回 true。
bind_as(args={})	作为一个指定用户绑定。
delete(args)	针对指定的 DN 删除目录项，DN 是所支持的惟一参数。
delete_attribute(dn, attribute)	删除一个属性及其所有值。dn 参数指定此目录项，attribute 参数指定要删除的那个属性。
get_operation_result()	返回绑定、搜索、添加、修改、重命名和删除操作的操作结果代码和消息。

modify(args)	修改指定目录项的属性值。将如下参数作为一个 hash：
	• :dn — 要修改的目录项的 DN。
	• :operations — 具体的修改，其中每个修改由数组指定，该数组由如下元素组成：
	  ○ Operator — 可以是 :add、:replace 或 :delete
	  ○ Attribute name — 要修改的属性
	  ○ Attribute value — 该属性的值

new(args = {})	创建类型 Net::LDAP 的一个对象，但不会打开与服务器的连接。参数可以是： • :host - LDAP 服务器主机，默认为本地主机。 • :port - LDAP 服务器端口，默认为 389。 • :auth - 包含授权参数的一个 hash。

创建一个 Rails 应用程序

在本节内，创建一个 Rails 应用程序来用 Net::LDAP Ruby 库创建一个目录服务。使用 rails 命令来创建一个称为 netldap 的 rails 应用程序：

```
c:/ruby>rails netldap
```

一个具有 Rails 应用程序完整目录结构的 Rails 应用程序将被创建。我们将作为一个控制器脚本运行这个 Net::LDAP Ruby on Rails 应用程序，来用不同的控制器动作分别创建一个目录项、修改一个目录项、搜索一个目录项和删除一个目录项。此外还将对应于每个控制器动作创建 RHTML 视图模板以便为目录项输入数据。

创建一个称为 directory 的控制器脚本，包含控制器动作 add_entry、modify_entry、 search_entry 和 delete_entry：

```
C:/ruby/netldap>ruby script/generate controller directory
add_entry modify_entry search_entry delete_entry
```

一个称为 directory_controller.rb 的控制器脚本在 controllers 目录内创建。此控制器脚本包含控制器动作 add_entry、modify_entry、search_entry 和 delete_entry。视图模板 add_entry.rhtml、modify_entry.rhtml、earch_entry.rhtml 和 delete_entry.rhtml 在 views 文件夹内创建。

在接下来的小节内，修改控制器动作和视图模板来添加一个目录项、修改目录项、搜索目录项和删除目录项。

创建一个目录项

下一步是在 Tivoli Directory Server 内创建一个目录项。一个目录项包含属性和属性值。目录项的 DN 代表的是此目录项的专有名称。一个 DN 包含相对专有名和基础 DN。在本例中，我们在 cn=localhost root/base DN 内创建一个目录项。

如果还未启动，请启动 Tivoli Directory Server 实例。

每个目录项由 dn 属性标识。 objectClass 属性指定数据类型以及每项内必须和可选的属性。对象类形成了一个类的等级结构； 常用的对象类有 top、organization 和 organizationalPerson。所有对象类均是对象类 top 的子类。我们将用 top、person 和 organizationalPerson 对象类创建一个目录服务：

- top 不具有任何必须属性。
- person 具有必须属性 cn 和 sn。
- organizationalPerson 不具有任何必须属性。在对象类 organizationalPerson 的目录项内可以指定的属性还有 title 和 telephoneNumber。

修改 add_entry.rhtml 视图模板来输入此目录项的数据。用 FormTagHelper 类的 form_tag 方法定义一个表单。用 text_field(object_name, method, options = {}) 方法定义此表单内的一个字段。方法参数 object 代表的是此表单模板的一个对象。method 参数代表的是作为此表单对象的一个属性的表单字段。比如，如下的文本字段：

```
text_field("directory_entry", "title", "size" => 20)
```

转变成 HTML 表单文本字段：

```
<input type="text" id="directory_entry_title" name="directory_entry[title]" size="20"
value="#{@directory_entry.title}" />
```

将姓、名、职务、电话号码、部门和传真号的文本字段添加到 add_entry.rhtml。

文本字段	属性
First name	gn
Last name	sn
Title	title
Telephone number	telephoneNumber
Department	physicalDeliveryOfficeName
FAX number	facsimileTelephoneNumber

清单 1 给出了 add_entry.rhtml。

清单 1. add_entry.rhtml

```
<html>
<body>

<div>
    <table border='0' cellspacing='0' cellpadding='5'>
       <tr>
         <caption>
           Add Directory Entry
         </caption>
       </tr>
          <!-- start_form_tag -->
     <%= form_tag :action => "add_entry" %>
       <tr>
          <td>First Name*</td>
          <td><%= text_field(:add_entry, :gn) %></td>
       </tr><tr>
           <td>Last Name*</td>
           <td><%= text_field(:add_entry, :sn) %></td>
       </tr>

       <tr>
           <td>Title</td>
           <td><%= text_field(:add_entry, :title) %></td>
       </tr><tr>
           <td>Telephone Number</td>
           <td><%= text_field(:add_entry, :telephoneNumber) %></td>
       </tr>

       <tr>
           <td>Department</td>
           <td><%= text_field(:add_entry, :physicalDeliveryOfficeName) %></td>
       </tr>


       <tr>
            <td>Fax Number</td>
            <td><%= text_field(:add_entry, :facsimileTelephoneNumber) %></td>
       </tr>


       <tr>
            <td><input type="submit" value="Submit"></td>
       </tr>
     <%= end_form_tag %>
     </table>
        </div>
* indicates a required field.
</body>
</html>
```

修改控制器动作 add_entry 并检索参数值。检索名（:gn）和姓（:sn）并定义变量 cn。

```
values = params[:add_entry]
    gn=values[:gn]
    sn= values[:sn]
    cn=gn+sn
```

定义此目录项的专有名 dn，它由 rdn 和基础 DN 组成。

```
dn="cn="+cn+",cn=localhost"
```

检索其他表单字段的值。

```
title=values[:title]
telephoneNumber=values[:telephoneNumber]
physicalDeliveryOfficeName=values[:physicalDeliveryOfficeName]
facsimileTelephoneNumber=values[:facsimileTelephoneNumber]
```

定义一个变量 attr，它包含此目录项的不同属性。

```
attr = {
   :cn => cn,
   :objectclass => ['top', 'person', 'organizationalPerson'],
   :sn => sn,
   :title => title,
   :telephoneNumber => telephoneNumber,
   :physicalDeliveryOfficeName => physicalDeliveryOfficeName,
   :facsimileTelephoneNumber => facsimileTelephoneNumber
    }
```

打开与 Tivoli Directory Server 的连接并使用 add() 方法将此目录项添加到服务器。

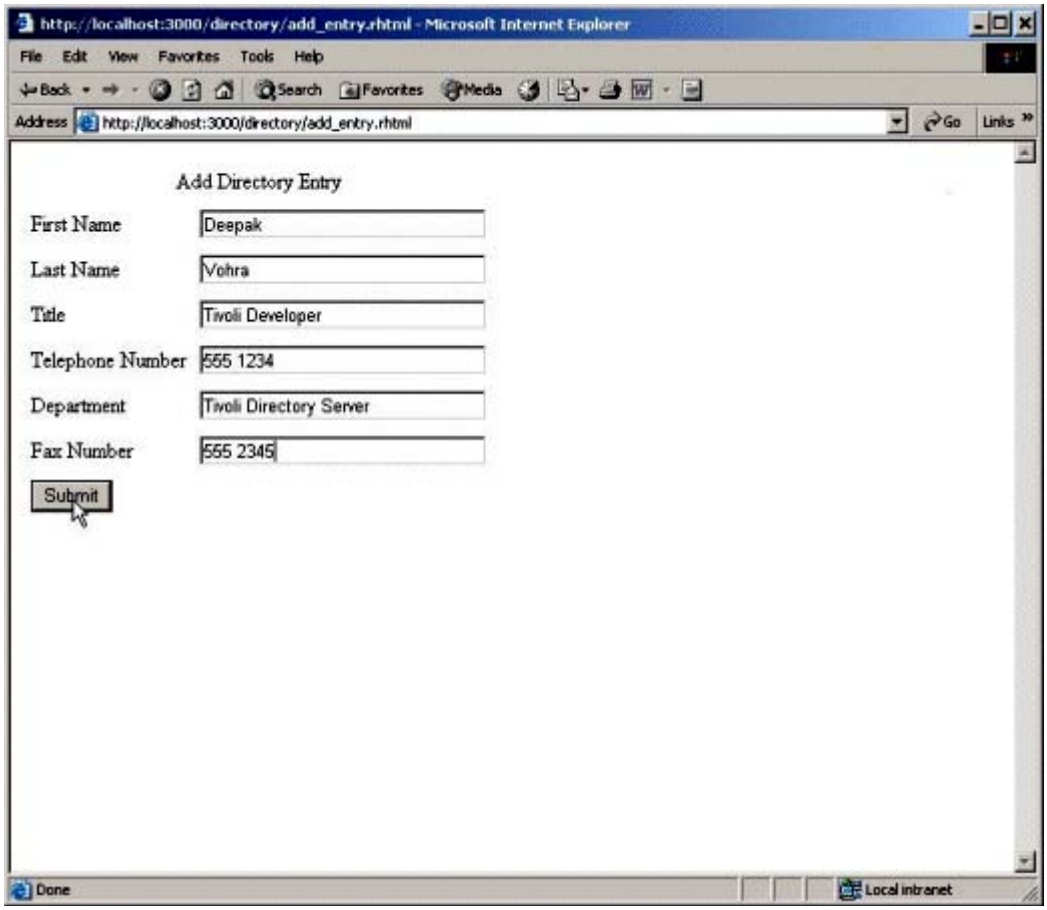
```
Net::LDAP.open( :host => 'localhost', :port => 389,:base =>
'cn=localhost', :auth => { :method => :simple, :username => 'cn=root',

 :password => 'tivoli' } ) do |ldap|
    ldap.add( :dn => dn, :attributes => attr )
end
```

用 add_entry.rhtml 模板创建一个目录项。用如下命令启动 WEBrick 服务器：

```
C:/ruby/netldap>ruby script/server
```

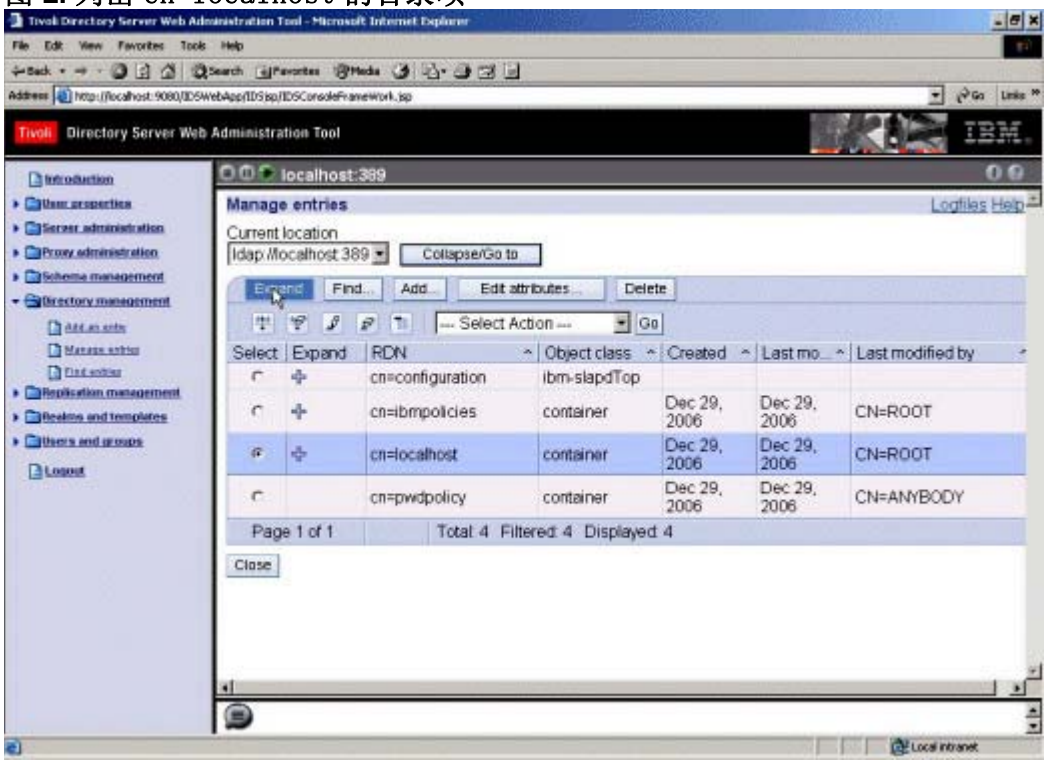
用 URL http://localhost:3000/directory/add_entry.rhtml 调用 add_entry.rhtml 视图模板。为不同属性指定值并单击 **Submit**，如下所示。

图 1. 添加一个目录项



在 Tivoli Directory Server 实例内的 cn=localhost 目录项内创建了一个目录项。在 Web Administration Tool 中选择 cn=localhost 目录项并单击 **Expand**，如下所示。

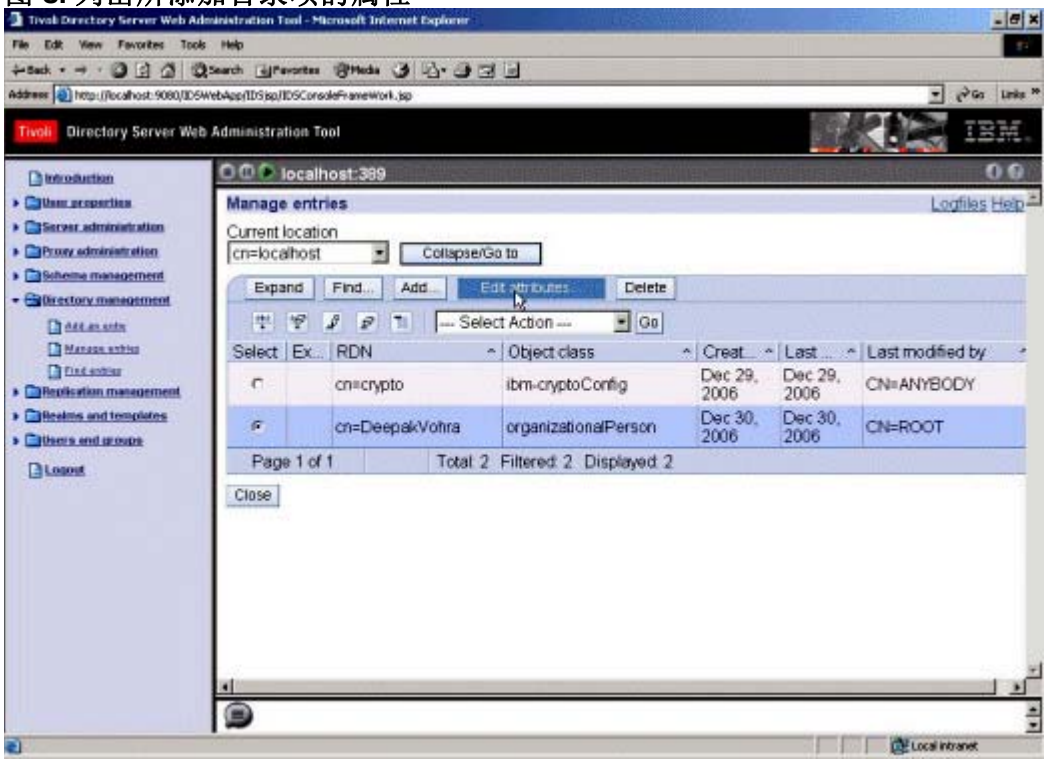
图 2. 列出 cn=localhost 的目录项



用 Ruby on Rails 所添加的这个目录在 cn=localhost 目录项内的目录项中列出。要列出此目录项的属性，选

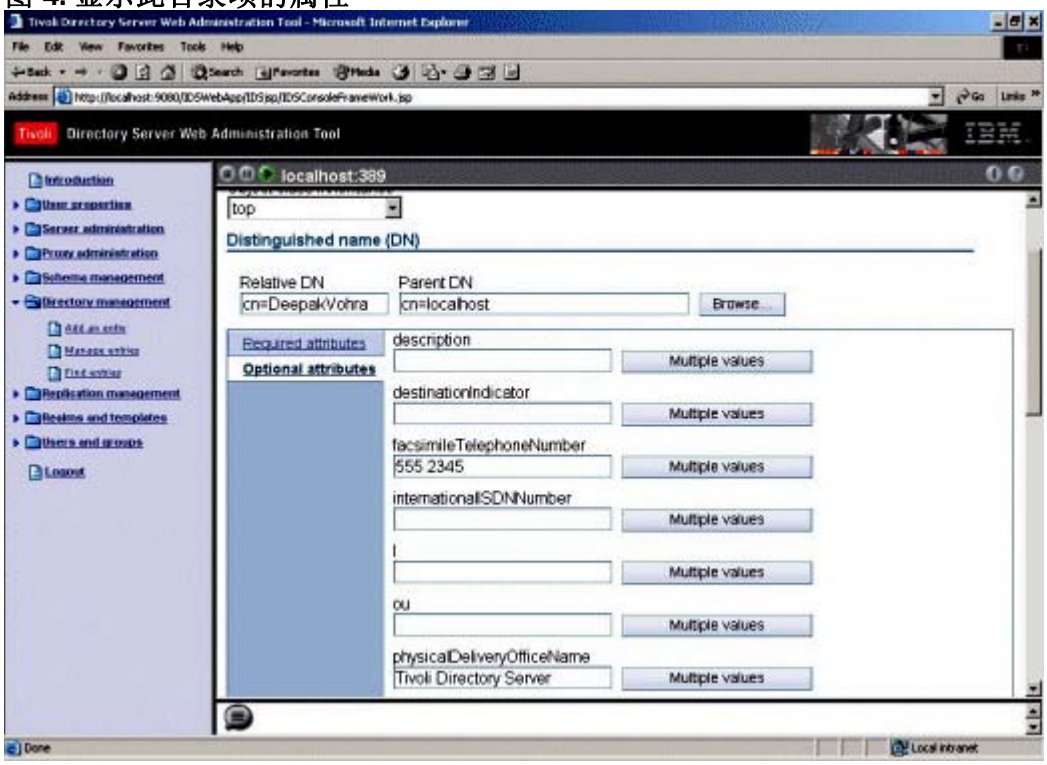
择此目录项并单击 **Edit attributes**。

图 3. 列出所添加目录项的属性



所添加的目录项的属性就会显示出来。

图 4. 显示此目录项的属性



修改一个目录项

在本章节内，我们将修改一个目录项。要修改的数据在 modify_entry.rhtml 内输入。与添加目录项类似，同样要用 FormTagHelper 类的 form_tag 方法向 modify_entry.rhtml 视图模板添加一个表单。用 FormHelper 类的 text_field 方法添加表单字段。清单 2 给出了这个 modify_entry.rhtml 视图模板。

清单 2. modify_entry.rhtml

```
<html>
<body>

<div>
    <table border='0' cellspacing='0' cellpadding='5'>
       <tr>
         <caption>
             Modify Directory Entry
         </caption>
       </tr>
    <!-- start_form_tag -->
     <%= form_tag :action => "modify_entry" %>
       <tr>
            <td>First Name*</td>
            <td><%= text_field(:modify_entry, :gn) %></td>
       </tr><tr>
            <td>Last Name*</td>
            <td><%= text_field(:modify_entry, :sn) %></td>
       </tr>

       <tr>
            <td>Title</td>
            <td><%= text_field(:modify_entry, :title) %></td>
       </tr><tr>
            <td>Telephone Number</td>
            <td><%= text_field(:modify_entry, :telephoneNumber) %></td>
       </tr>

       <tr>
            <td>Department</td>
            <td><%= text_field(:modify_entry, :physicalDeliveryOfficeName) %></td>
       </tr>


       <tr>
            <td>Fax Number</td>
            <td><%= text_field(:modify_entry, :facsimileTelephoneNumber) %></td>
       </tr>


       <tr>
            <td><input type="submit" value="Submit"></td>
       </tr>
   <%= end_form_tag %>
  </table>
        </div>
* indicates a required field.
</body>
</html>
```

在 modify_entry.rhtml 模板被提交时，会调用控制器 directory 的 modify_entry 控制器动作。修改 modify_entry 控制器动作。

检索表单字段 :gn 和 :sn 的值，并定义一个变量 cn。

```
values = params[:modify_entry]
    gn=values[:gn]
    sn= values[:sn]
    cn=gn+sn
```

目录项由一个专有名标识。定义要修改的这个目录项的 DN。

```
dn="cn="+cn+",cn=localhost"
```

检索其他表单字段的值。

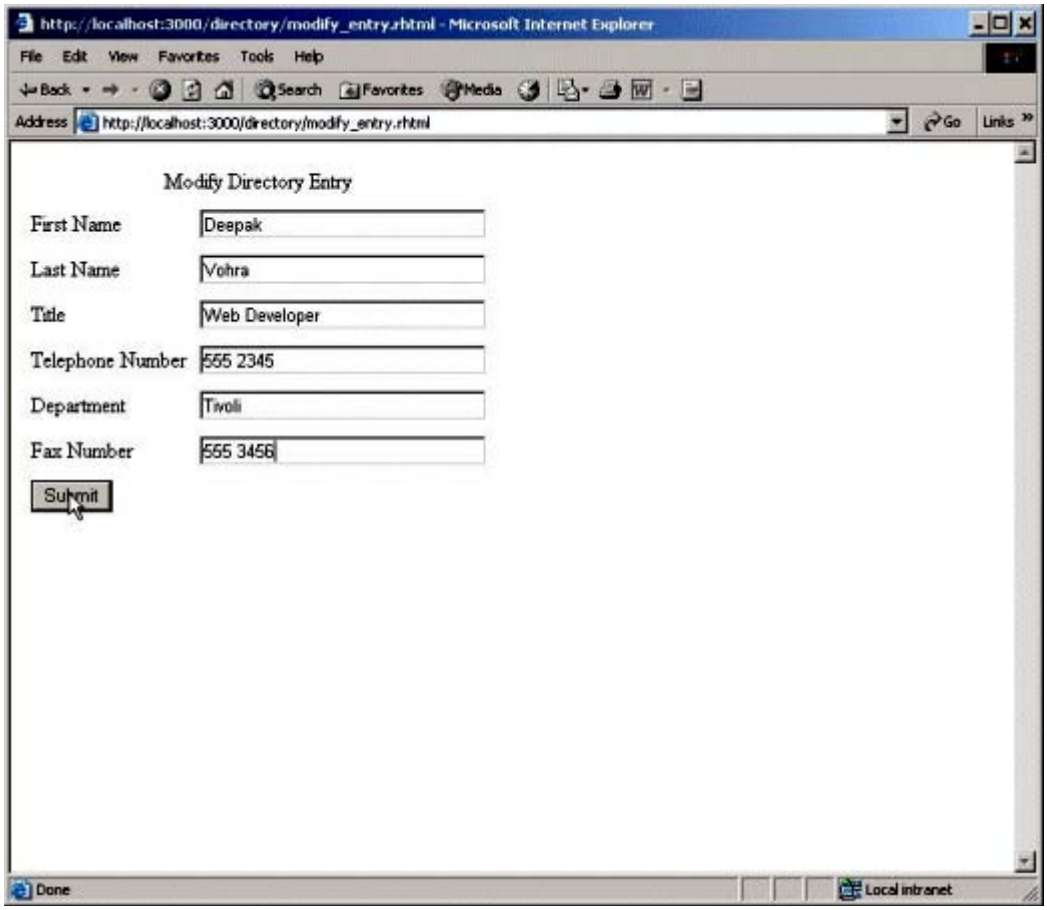
```
title=values[:title]
telephoneNumber=values[:telephoneNumber]
physicalDeliveryOfficeName=values[:physicalDeliveryOfficeName]
 facsimileTelephoneNumber=values[:facsimileTelephoneNumber]
```

打开与 Tivoli Directory Server 的连接并用 replace_attribute 方法替换这些属性值。比如，:title 属性被替换如下：

```
Net::LDAP.open( :host => 'localhost', :port => 389,:base =>
'cn=localhost', :auth => { :method => :simple, :username => 'cn=root',

:password => 'tivoli' } ) do |ldap|

    ldap.replace_attribute dn, :title, title
end
```

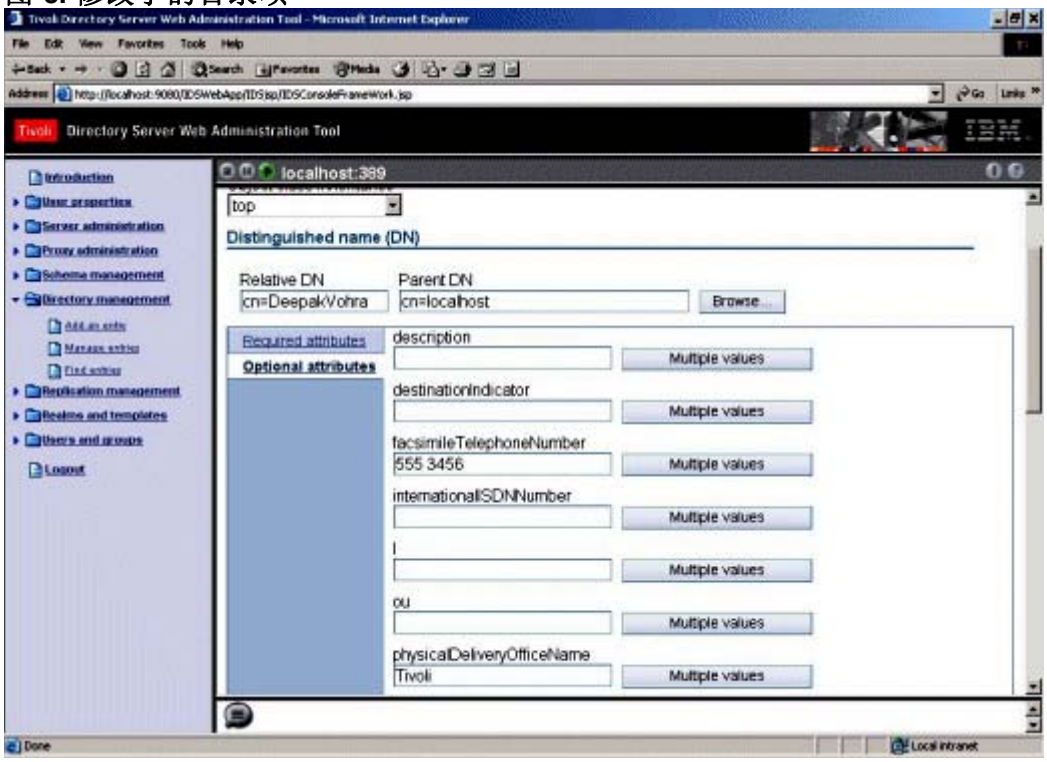
目录项在 modify_entry.rhtml 视图模板中被修改。启动 WEBrick Web 服务器并使用 URL http://localhost:3000/directory/modify_entry.rhtml 调用 modify_entry.rhtml 视图模板。指定要修改的目录项以及被修改的属性。单击 **Submit**，如下所示。

图 5. 修改目录项



此目录项得到了修改，如 Web Administration Tool 所示。

图 6. 修改了的目录项



搜索一个目录

本节展示了如何搜索一个目录项。目录搜索的结果显示在使用 Ajax Web 技术输入目录项数据的同一个页面中。此 Ajax Web 技术由原型库实现。这个原型库包括类 PrototypeHelper 来创建一个表单，这个表单可使用 Ajax 进行异步更新。将这个原型库包含在 search_entry.rhtml 视图模板内。

```
<%= javascript_include_tag "prototype" %>
```

用 PrototypeHelper 类的 form_remote_tag 方法添加一个使用 Ajax 提交的表单。form_remote_tag 的 :update 选项指定了由服务器响应进行更新的那个表单元素。:url 选项指定了此表单被提交到的那个 URL，而控制器动作则由 :action 参数指定。

```
<%=form_remote_tag(:update=>"directory_entry",
:url=>{:action=>:search_entry}) %>
<% end_form_tag %>
```

用 FormTagHelper 类的 text_field_tag 指定此目录项的输入字段。而要更新的表单元素则作为 div 指定。

```
<div id="directory_entry"></div>
```

search_entry.rhtml 视图模板如清单 3 所示。

清单 3. search_entry.rhtml

```
<html><head>
    <title></title>
    <%= javascript_include_tag "prototype" %>
  </head>

<body>
     <caption>
      Search  Directory Entry
     </caption>
    <%=form_remote_tag(:update=>"directory_entry",
           :url=>{:action=>:search_entry}) %>
     <table>
           <tr>
     <label>First Name*</label>
    <%=text_field_tag:firstName %></tr>
       <tr><label>Last Name</label>
    <%=text_field_tag:lastName %></tr>
      <%=submit_tag "Search" %>
    </table>
     <caption>
         <b> Directory Entry Table</b>
     </caption>

   <div id="directory_entry"></div>
   <% end_form_tag %>
        * indicates a required field.
</body>

</html>
```

在 search_entry.rhtml 表单被提交时，会调用此目录控制器的 search_entry 控制器动作。修改 search_entry

动作。检索 :gn 和 :sn 字段的值，并定义要搜索的目录项的基础 DN。

```
gn=values[:gn]
 sn= values[:sn]
cn=gn+sn
treebase= "cn="+cn+",cn=localhost"
```

指定要检索的目录项的属性。

```
attrs = ["cn", "sn","title", "telephoneNumber","physicalDeliveryOfficeName",
"facsimileTelephoneNumber"]
```

打开与 Tivoli Directory Server 的连接并使用 Net::LDAP 类的 search 方法搜索这个特定的目录项。

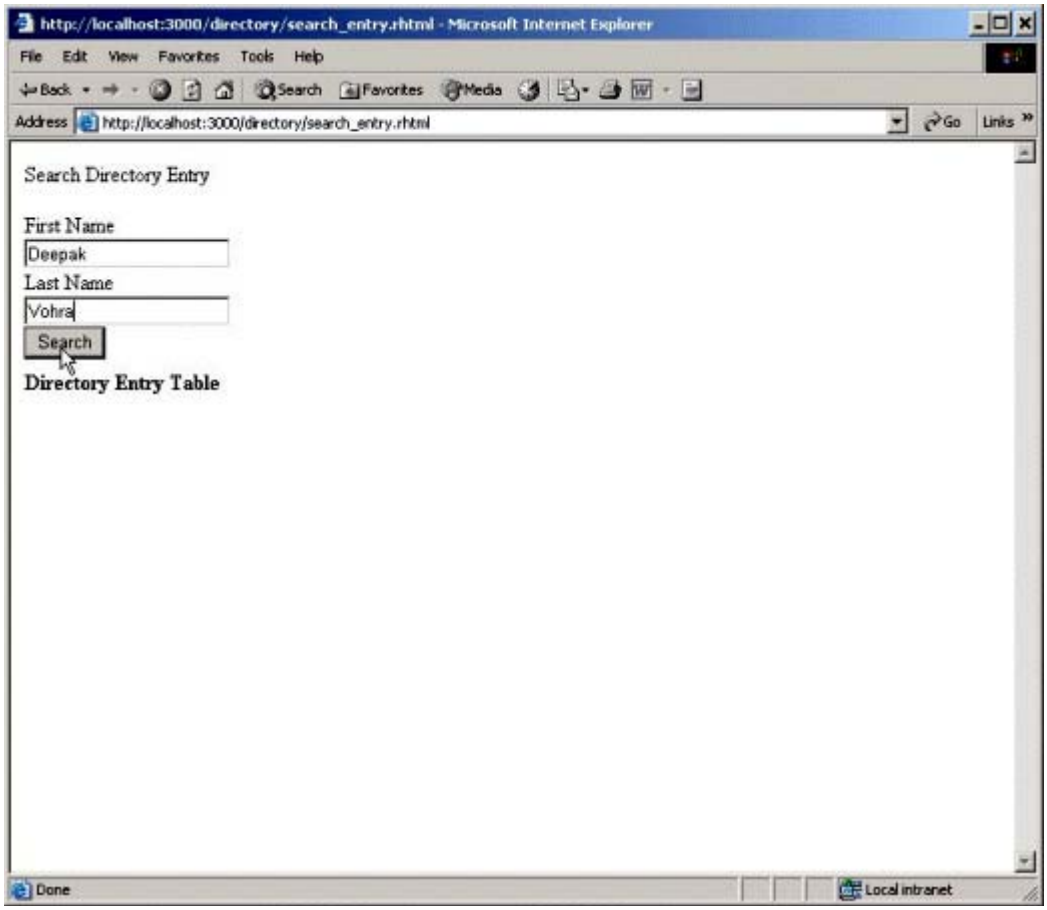
```
Net::LDAP.open( :host => 'localhost', :port => 389,:base =>
'cn=localhost', :auth => { :method => :simple, :username =>
'cn=root',

:password =>'tivoli' } ) do |ldap|
  ldap.search( :base => treebase,  :attributes => attrs,
:return_result => true ) do |directory|
end
```

search 方法返回一个结果集。迭代此结果集并创建一个 HTML 表来作为响应发送给 search_entry.rhtml 视图模板。比如，针对 cn 属性的一行就被添加到了此表中。

```
directoryEntry+="<tr>"
directoryEntry+="<td>cn</td>"
directoryEntry+="<td>"+"#{directory.cn}"+"</td>"
directoryEntry+="</tr>"
```

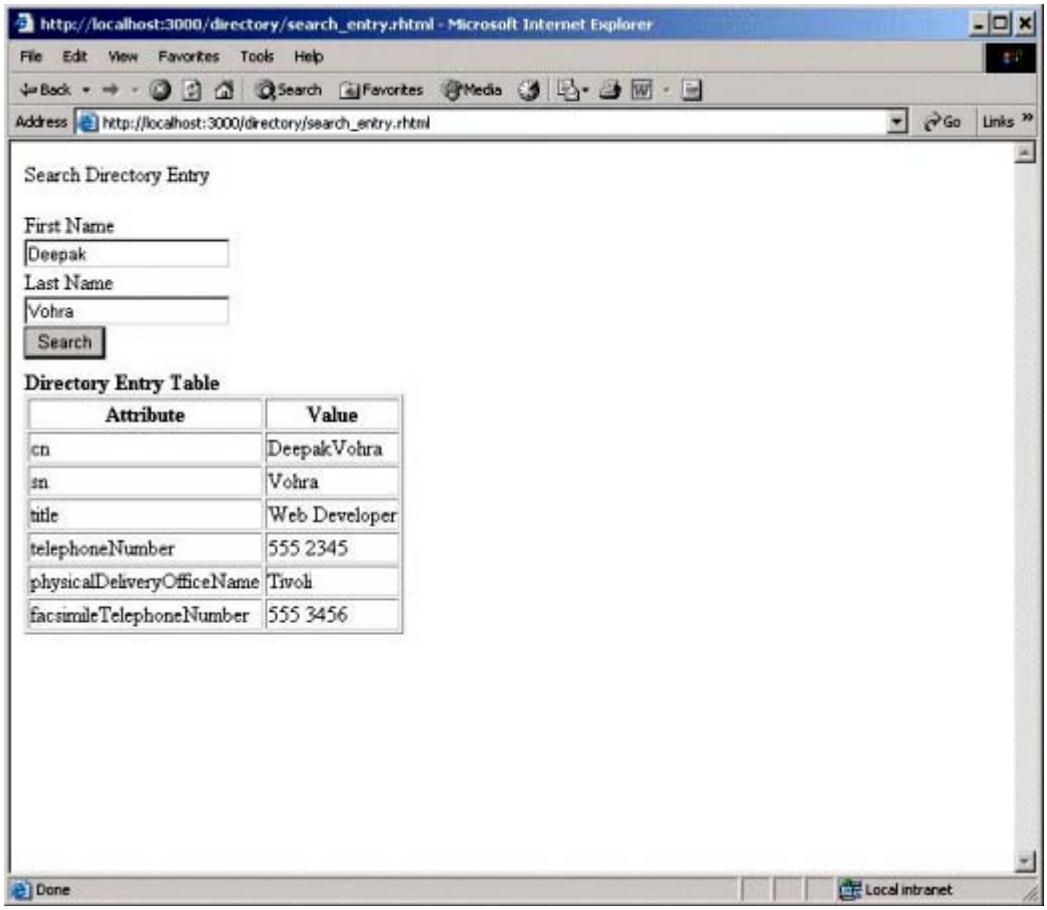
用 URL http://localhost:3000/directory/search_entry.rhtml 调用 asearch_entry.rhtml 视图模板来搜索此目录项。指定 :gn 和 :sn 属性的值以形成目录项的 rdn，并单击 **Search**。

图 7. 搜索一个目录项



此目录项属性即被列出。

图 8. 目录项搜索结果



删除一个目录项

在本节中，我们将删除一个目录项。目录项由 DN 标识，而 DN 由 rdn 和基础 DN 组成。要被修改的目录项的 RDN 在 delete_entry.rhtml 视图模板内指定。form_tag 方法被用来创建一个表单，text_field 标记被用来创建一个表单文本字段。 delete_entry.rhtml 视图模板包含名和姓的输入字段，如下所示。

清单 4. delete_entry.rhtml

```
<html>
<body>

<div>
                <table border='0' cellspacing='0' cellpadding='5'>
                        <tr>
                                <caption>
                                        Delete Entry
                                </caption>
                        </tr>
                        <!-- start_form_tag -->
                        <%= form_tag :action => "delete_entry" %>
                        <tr>
                                <td>First Name*</td>
                                <td><%= text_field(:delete_entry, :gn) %></td>
                        </tr><tr>
                                <td>Last Name*</td>
                                <td><%= text_field(:delete_entry, :sn) %></td>
```

```
                        </tr><tr>
                                <td><input type="submit" value="Submit"></td>
                        </tr>
                        <%= end_form_tag %>
                </table>
        </div>
* indicates a required field.
</body>
</html>
```

当 delete_entry.rhtml 表单被提交时，会调用 directory 控制器的 delete_entry 控制器动作。delete_entry 控制器动作检索这些表单字段的值并创建要删除的那个目录项的 DN。

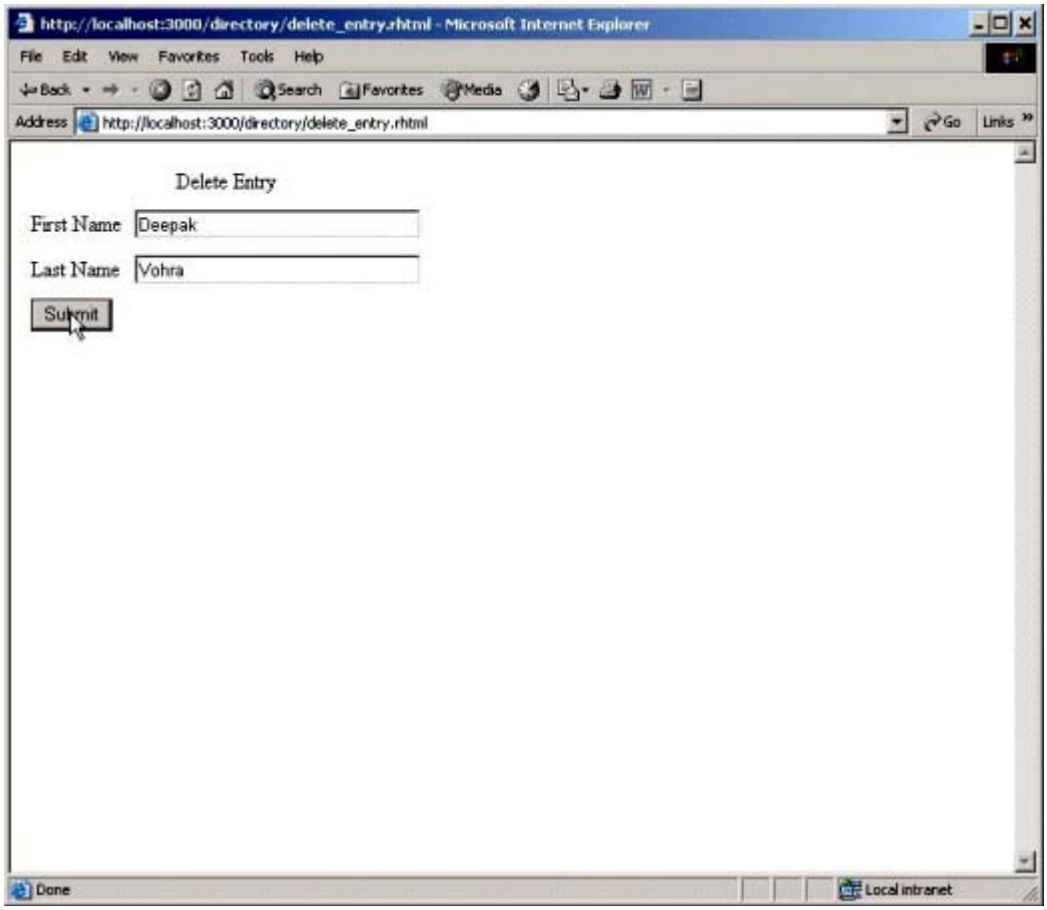
```
values = params[:delete_entry]
    gn=values[:gn]
    sn= values[:sn]
    cn=gn+sn
    dn="cn="+cn+",cn=localhost"
```

打开与目录服务器的连接并用 Net::LDAP 类的 delete 方法删除此目录项。

```
Net::LDAP.open( :host => 'localhost', :port => 389,
:base => 'cn=localhost', :auth => { :method => :simple, :username =>
'cn=root', :password => 'tivoli' } ) do |ldap|
     ldap.delete :dn => dn
 end
```

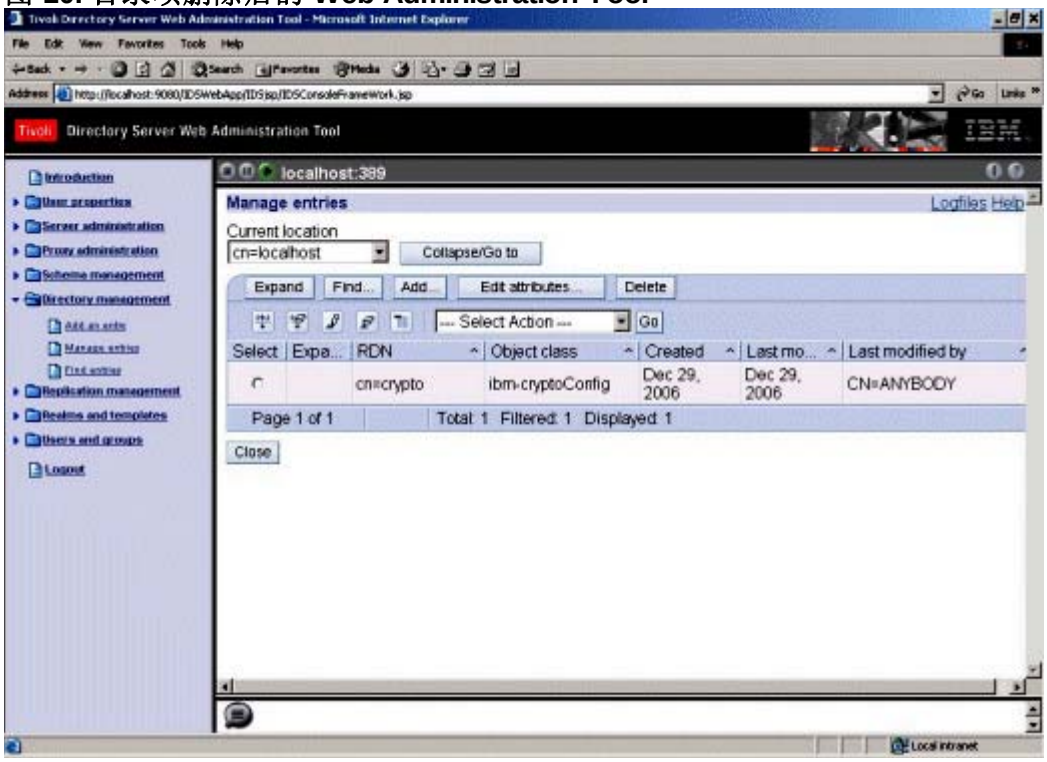
要删除一个目录项，用 URL http://localhost:3000/directory/delete_entry.rhtml 调用 delete_entry.rhtml 视图模板。指定要删除的那个目录项的 :gn（名）和 :sn（姓）属性并单击 **Submit**。

图 9. 删除一个目录项



此目录项即被删除，如 Web Administration Tool 内所示。

图 10. 目录项删除后的 Web Administration Tool



directory_controller.rb 控制器脚本如下所示。

清单 5. directory_controller.rb

```
require 'net/ldap'

class DirectoryController < ApplicationController

def add_entry

    values = params[:add_entry]
    gn=values[:gn]
    sn= values[:sn]
    cn=gn+sn

     dn="cn="+cn+",cn=localhost"

     title=values[:title]
     telephoneNumber=values[:telephoneNumber]
     physicalDeliveryOfficeName=values[:physicalDeliveryOfficeName]
     facsimileTelephoneNumber=values[:facsimileTelephoneNumber]

     attr = {
   :cn => cn,
   :objectclass => ['top', 'person', 'organizationalPerson'],
   :sn => sn,
   :title => title,
   :telephoneNumber => telephoneNumber,
   :physicalDeliveryOfficeName => physicalDeliveryOfficeName,
   :facsimileTelephoneNumber => facsimileTelephoneNumber
    }



Net::LDAP.open( :host => 'localhost', :port => 389,:base =>
'cn=localhost', :auth => { :method => :simple, :username => 'cn=root',

:password => 'tivoli' } ) do |ldap|

    ldap.add( :dn => dn, :attributes => attr )
end
end



def delete_entry


values = params[:delete_entry]
    gn=values[:gn]
    sn= values[:sn]
    cn=gn+sn

    dn="cn="+cn+",cn=localhost"

Net::LDAP.open( :host => 'localhost', :port => 389,:base =>
'cn=localhost', :auth => { :method => :simple, :username => 'cn=root',

:password => 'tivoli' } ) do |ldap|
```

```
     ldap.delete :dn => dn
 end

 end

def search_entry

 gn=params[:firstName]
    sn= params[:lastName]
    cn=gn+sn

    treebase= "cn="+cn+",cn=localhost"
    attrs = ["cn", "sn","title", "telephoneNumber","physicalDeliveryOfficeName",
             "facsimileTelephoneNumber"]

 directoryEntry="<table border><tr><th>Attribute</th>
                 ;<th>Value</th></tr>"

Net::LDAP.open( :host => 'localhost', :port => 389,:base =>
'cn=localhost', :auth => { :method => :simple, :username =>
'cn=root',

 :password => 'tivoli' } ) do |ldap|


  ldap.search( :base => treebase,  :attributes => attrs,
:return_result => true ) do |directory|

directoryEntry+="<tr>"
directoryEntry+="<td>cn</td>"
directoryEntry+="<td>"+"#{directory.cn}"+"</td>"
directoryEntry+="</tr>"

directoryEntry+="<tr>"
directoryEntry+="<td>sn</td>"
directoryEntry+="<td>"+"#{directory.sn}"+"</td>"
directoryEntry+="</tr>"

directoryEntry+="<tr>"
directoryEntry+="<td>title</td>"
directoryEntry+="<td>"+"#{directory.title}"+"</td>"
directoryEntry+="</tr>"

directoryEntry+="<tr>"
directoryEntry+="<td>telephoneNumber</td>"
directoryEntry+="<td>"+"#{directory.telephoneNumber}"+"</td>"
directoryEntry+="</tr>"


directoryEntry+="<tr>"
directoryEntry+="<td>physicalDeliveryOfficeName</td>"
directoryEntry+="<td>"+"#{directory.physicalDeliveryOfficeName}"+"</td>"
directoryEntry+="</tr>"

directoryEntry+="<tr>"
directoryEntry+="<td>facsimileTelephoneNumber</td>"
directoryEntry+="<td>"+"#{directory.facsimileTelephoneNumber}"+"</td>"
directoryEntry+="</tr>"
```

```
end

directoryEntry+="</table>"
render:text=> directoryEntry

end

 end

def modify_entry

 values = params[:modify_entry]
    gn=values[:gn]
    sn= values[:sn]
    cn=gn+sn

    dn="cn="+cn+",cn=localhost"

     title=values[:title]
     telephoneNumber=values[:telephoneNumber]
     physicalDeliveryOfficeName=values[:physicalDeliveryOfficeName]
     facsimileTelephoneNumber=values[:facsimileTelephoneNumber]


Net::LDAP.open( :host => 'localhost', :port => 389,:base =>
'cn=localhost', :auth => { :method => :simple, :username => 'cn=root',

:password => 'tivoli' } ) do |ldap|

    ldap.replace_attribute dn, :title, title
    ldap.replace_attribute dn, :telephoneNumber, telephoneNumber
    ldap.replace_attribute dn, :physicalDeliveryOfficeName, physicalDeliveryOfficeName
    ldap.replace_attribute dn, :facsimileTelephoneNumber, facsimileTelephoneNumber

  end

 end

end
```

结束语

在本文中，您安装了 Ruby on Rails 和 Net::LDAP Ruby gem。了解了如何在 Tivoli Directory Server V6.0 内创建一个目录项，然后修改此目录项，搜索此目录项和删除此目录项，这些均使用 Ruby 脚本语言实现。

就开发 LDAP 目录服务方面的应用而言，Ruby 完全可以与 PHP 这种最为常用的脚本语言媲美。根据 TIOBE Programming Community Index 的统计，过去一年来，Ruby 的使用已经有所增加。

下载

描述	名字	大小	下载方法
LDAP 应用程序	os-ldap-tivoli-ruby_ldap-app.zip	29KB	**HTTP**

关于下载方法的信息

参考资料

学习

- “Introduction to LDAP, Part 2: LDAP and WebSphere” 阐释了 LDAP 如何为您的 WebSphere Application Server 提供优秀的身份验证机制。
- 获得对 Tivoli Directory Server 的大致介绍，了解其特性、优势、系统要求、产品支持等。
- 学习并下载 Ruby on Rails，这是一种为简化程序员工作和提高其效率而进行了优化的开源 Web 框架。
- “An introduction to Ruby on Rails for DB2 developers” 阐释了如何使用 Ruby on Rails Web 框架加速基于 DB2 的 Web 应用程序的开发。
- 查阅 TIOBE Programming Community Index，看看 Ruby 的使用在过去一年内有何提高。
- “使用 Ruby on Rails 快速开发 Web 应用程序” 介绍了 Ruby on Rails，详细讨论了各组件并展示了它是如何工作的。
- 要收听针对软件开发人员的有趣访谈和讨论，请查看 developerWorks podcasts。
- 随时关注 developerWorks 技术活动和网络广播。
- 了解 developerWorks on Twitter。
- 查阅最近将在全球举办的面向 IBM 开放源码开发人员的研讨会、交易展览、网络广播和其他 活动。
- 访问 developerWorks 开放源码专区 获得丰富的 how-to 信息、工具和项目更新，帮助您用开放源码技术进行开发，并与 IBM 产品结合使用。
- 查看免费的 developerWorks 演示中心，观看并了解 IBM 及开源技术和产品功能。

获得产品和技术

- 利用 IBM 试用软件 改进您的下一个开放源码开发项目，这些软件可以通过下载获得。
- 下载 IBM 产品评估试用软件 或 IBM SOA Sandbox for Reuse 并试用这些来自 DB2®、Lotus®、Rational®、Tivoli® 和 WebSphere® 的应用程序开发工具和中间件产品。

讨论

- 参与 developerWorks blogs 并加入 developerWorks 社区。

关于作者

Deepak Vohra 是一名经过 Sun 认证的 Java 程序员和 Web 组件开发人员。Deepak 曾在 XML Journal、onjava.com 和 WebLogic Developer's Journal 上发表过文章。