ArcGIS Server Java

ADF 案例教程

V9.3.1

吴泳锋 王坚

2009-10-23

意见建议欢迎 Email: warrenwyf@gmail.com

前言

如果你开始学 ArcGIS Server Java ADF，你会在第一时间发现相关资料的缺乏。手上最靠谱的资料就是自带的 Help 文件，可是当你虔诚地点开 “Open Java Help” 以后一看，章节倒是不少，可是每一篇都是那么出人意料的短小精悍——这可不是因为英文比较精炼的缘故。好吧，吐血之余如果你能通过 Google 在浩瀚的互联网挖掘一下，你应该会如获至宝地发现前辈高人 CJ 的 Java ADF 教程，恭喜你，你基本已经找到中文 Java ADF 开发资料一大半的内容了。

相比 .Net ADF 的开发资料，你不禁会发出一声感慨：“真是穷的穷死，富的富死啊！”。唉，不用多抱怨这个社会的不公了，想想 CJ 写那篇教程（2007.8 左右）之前的岁月，想想那个时候受苦受难的弟兄们，现在好歹算是解放了吧。

转眼解放也 2 年了，Java ADF 的资料还是一如既往的少——同时，ArcGIS Server 已经从 9.2 更新到了 9.3.1，CJ 的教程是基于 9.2 的，有一些内容已经和 9.3 版本对不上了——这些都成为了我写这篇教程的原因之一。

这篇教程在写完初稿以后进行修改的时候，得到了当初帮助我接触 Java ADF 的同事的斧正，并给了我很多他积累的文章。因此虽然本文使用“我”的称谓，但是或许这指代的是执笔的我，也有可能是我的同事，读者不必深究。

我一直不喜欢看那些可以充当手册的教材，而是比较喜欢听谁来给我把知识像故事一样地娓娓道来——至今我还非常想念当年在实验室的一个师兄，感谢他在带我入门 GIS 世界时候的循循善诱和耐心教导，顺便也遥祝他在新加坡的学习生活愉快。

希望这篇教程也能给你带来类似的感觉！

目录

前言 1

第一章 JSF 基础 6

一 ADF 选择了 JSF 6

二 JSF 的直观印象 6

第二章 ADF 的体系结构 9

一 ADF 是什么 9

二 ADF 的物理组成 11

三 ADF 的逻辑结构 13

四 ADF 是怎样开始工作的 15

第三章 ADF 常用组件及原理分析 20

一 ADF 页面结构 20

二 ADF 中的服务 22

三 几个 ADF 常用组件 25

四 Map 组件详解 27

五 EsriMap 详解 31

六 给地图加上弹出气泡【案例】 36

第四章 Command、Tool 和 Button 40

一 Command 40

二 自定义 Command 使地图等比放大【案例】 42

三 WebContext 的 refresh 方法 44

四 Tool 46
五 自定义 Tool 实现点击地图居中【案例】 48
六 Button 50
七 自定义 Button 实现前面的 Command 和 Tool【案例】 51
第五章 Task 54
一 Task 的请求 54
二 Task 的执行 58
三 Task 的渲染 59
四 定制 Task 布局的高级话题 61
第六章 查询及结果 65
一 文本查询 65
二 几何查询 66
三 Where 条件查询 67
四 QueryResult 68
五 GraphicElement 69
六 GraphicElement 的符号化 70
七 GraphicRenderer 73
第七章 编辑工具 77
一 编辑的条件 77
二 添加编辑工具 78
三 使用编辑工具 80

第八章　自定义的请求和响应 83
四　发送 AJAX 请求 83
五　监听 AJAX 请求并响应 84
六　自定义 AJAX 请求获得地图范围【案例】 87
七　通过 AJAXRenderer 进行交互 90
第九章　自定义 Functionality 94
一　GISFunctionality 详解 94
二　自定义一个地图放大 5 倍的 Functionality【案例】 96
第十章　调用 ArcObjects 100
一　Internet 和 Local 方式连接服务的区别 100
二　IServerContext 详解 101
三　调用 AO 实现缓冲功能【案例】 102
第十一章　有关 Web Service 105
一　通过 SOAP SDK 访问 ArcGIS Web Service 105
二　两种方式自定义 Web Service 107
三　实现 Web Service 的功能 108
第十二章　扩展 ArcGIS Server 112
一　Utility Objects 112
二　使用 Utility Objects 提高查询性能【案例】 114
三　Server Object Extensions 115
四　创建一个自定义的 Server Object Extension【案例】 116

结语.....120

附录.....121

一　配置 ArcGIS Server Java 开发环境（Eclipse 3.3 为例）.....121

二　配置 ArcGIS Server Java 开发环境（NetBeans 6 为例）.....122

第一章 JSF 基础

一 ADF 选择了 JSF

为什么 ESRI 会选择 JSF 作为 ArcGIS Java ADF 的基础框架，想来你也不会很关心，因为这已经是一个既成事实了。我以前是用 PHP 写 Web 应用的，也没用过 Struts、Spring 之类的框架，因此也就无从比较 JSF 与它们的优劣。有句话说的好，生活就像某件事情，如果你不能去反抗，那就闭上眼就享受吧。如果你要使用 ArcGIS Java ADF 开发，那么你还可以对框架挑三拣四么？好了，既然我们都无能为力，那么就尽量去享受它吧。

为了给自己再找点心理上的安慰，证明自己没有吃亏，你可以和我一样无聊地去 Google 搜一下关于 JSF 和 Struts、Spring 的相关页面数量，并把它作为一个框架流行度的大体指标。我在 2009.3.16 这一天得到的结果是 JSF：约 15700000，Struts：约 14500000——至于 Spring，这个名字实在是有太多的含义了，忽略了吧——这时，你发现原来全世界有这么多人和你一样在“享受”着 JSF，是不是心里踏实了很多呢？

好了，如果你还没有接触过 JSF，那就让我们开始试着去享受 JSF 吧。

二 JSF 的直观印象

如果我是一个项目经理，我当然最喜欢的是又会写代码，又会做美工，顺便最好还能陪我对战几把 Warcraft 的全能型选手——最好要的薪水也可以少一点。不过，自古文武全才型的选手就比较难得，手下能有一些懂一点美工的程序员或者懂一点代码的美工就已经很不错了。

因此，各种 Web 开发框架都希望做到的事情是尽量把页面设计和后台逻辑分离开。这里不仅仅涉及到 MVC 的实现，还在于开发者角色的分离。做美工的希望面向的是一个纯粹的页面设计工作，直接和 HTML 和 CSS 打交道会让他们非常舒服和得心应手；而程序员（不管是写页面上的 JavaScript 还是后台的 Java /PHP/.Net…）一般都不喜欢去掺合页面的设计。

JSF 一方面有自己的标签库，使得美工可以使用类 HTML 的标签来设计页面，这样可以实现和程序员角色的分离；另一方面，JSF 希望不仅做到这样的分离，而且希望带给程序员在进行 Web 开发时，来回徜徉于可能远隔千里之外的浏览器和服务器之上的时候，能够有更好的体验。

让我们看一下下面在一个 JSF 页面中的一段标签：

```
<h:commandButton value="确定" actionListener="#{user.verify}" />
```

在这里出现的“#{…}”中包含的内容是在 JSF 页面中绑定的服务器端的 JavaBean 对象，在这里是“user”。“user.verify”是“user”对象的一个监听。我们看一下与其相对应的 Java 代码吧：

```
public void verify(ActionEvent e)
{
 this.outcome = "login";
}
```

这样的组合让 JSF 应用在页面点击那个“commandButton”的时候，服务器会监听到浏览器发来的请求，然后执行相应的代码，在这里让“user”对象的 outcome 属性值变成“login”。

至于页面怎么向服务器发送请求，服务器怎么监听到请求等等细节，JSF 都已经封装了起来，无需你去多考虑和写多余的代码。你只需要派美工做好设计，然后使用 JSF

相应的标签组织好页面；然后派程序员把后台的功能具体实现了，JSF 会帮你把浏览器和服务器联系起来，完成你设计的功能。

嗯，到这里你或许会已经对 JSF 有一个直观的印象了：MVC、页面绑定服务器端对象。当然，JSF 更多的优点和特色还有很多，建议你找一本 JSF 的教材，内容肯定会很详细，我在这里就不多讲了。不过，关于 JSF 应用中比较重要的比如配置文件、事件处理、请求的生命周期等内容，我也会在后续的 ADF 讲解中根据需要做一些详细的解释的。好了，JSF 这个幕后英雄就先了解到这里吧，让我们先尽早地接触到 ADF 再说。

第二章 ADF 的体系结构

在正式进入 ADF 之前，我想说几句关于知识储备的废话。也许你是一个很有经验的 Web 开发者，也有可能你正想通过对 ADF 的学习使自己能够慢慢进入 WebGIS 世界。因此，在 ADF 的剖析过程中，我会尽可能地顺便介绍与之相关的内容，但是如果你有相关的知识储备当然更好。以我的感觉，在进入 ADF 之前至少应该对 B/S 架构、DHTML、AJAX 有一定了解，不求精通，起码要看得懂 JavaScript、HTML/XML、Java 语言写的代码。

一 ADF 是什么

说了半天的 ADF，突然发现到底 ADF 是什么还一直没提。突然想到有人在网上恶搞了“卧槽泥马”源自《战国策》，居然还有报纸以此引经据典拽了一把酸文，捧腹之余顿感正本清源的重要。我们的 ADF 全称是 ArcGIS Java Web Application Developer Framework，一般简称 Java ADF，在这里我就直接称呼 ADF 了。从这个名字也可以看到 ADF 的实质，它是为了开发 Web 应用而生的一套开发框架，开发人员可以在 ADF 的基础上相对比较容易地开发出 WebGIS 应用。相比 ArcGIS Server 其它的客户端 API，用 ADF 开发的应用功能最强大，对浏览器要求最低。

让我们从帮助文档里的一张图开始，先感性认识一下 ADF 包含的内容吧。虽然我看过不少描述 ADF 的图表，但是我认为，这张图，是最简洁明了的。

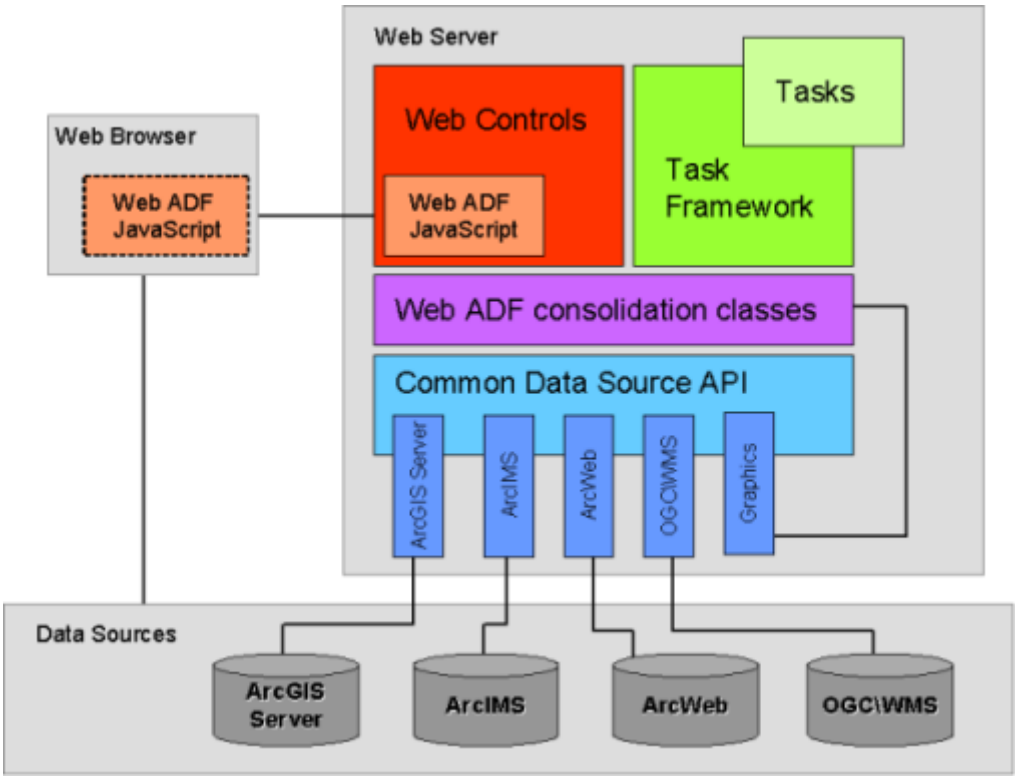


图 1　　ADF 的组成部分

大概来讲，ADF 主要是由一些 JSF 组件、实现 GIS 相关功能的 Java 类库、支持 AJAX 的 JavaScript 脚本库和一些帮助文档及例子的集合。

在图 1 中，首先你需要把最下面的那些“Data Sources”理解成服务。不管是 ArcGIS Server 或者是 ArcIMS 等发布的服务都可以作为 ADF 的数据源。在这些服务之上，ADF 内部通过不同的 API（“Common Data Source API”）去使用这些服务，当然，这些 API 上层会有相应的类库（“Web ADF consolidation classes”）进行支持。在这之上就是与开发者关系最直接的 ADF 的“Web Controls”（Map、TOC 等基本 GIS 功能）和“Task Framework”（完成比如查询、分析等定制的任务），事实上，我们可以把这两部分看成一个整体——Web 组件。这些 Web 组件包含了 JSF 组件以及相应的 AJAX 操作所需的 JavaScript 脚本库（“Web ADF JavaScript”）；它们都会在用户请求的时候被 ADF 通过一定的规则渲染成对应的 DHTML 后输出到浏览器。图 1 的左上角就是浏

览器端，事实上在浏览器中已经基本看不到 ADF 的影子了，因为用户看到的页面不过是一些 HTML 和 JavaScript 脚本的组合 ,页面会和服务器上的某些 ADF 对象进行交互，但是这对浏览器来说基本是透明的。

二 ADF 的物理组成

现在，ADF 对你来说或许就像远处的一个美女，氤氲中你依稀可以看到她的体态轮廓；当然，你同时也迫切希望能更深一步地了解她。现在我们要做的，就是走近她，慢慢揭开她的面纱。

我现在感觉自己有点像一个导演——当然，我可不想是拍《无极》那种烂片的水平——面对一个“她”，目前正在设计一个镜头，目的是怎样可以比较艺术化地表现出她的一切。这里，我决定从她最活色生香的胴体——物理组成入手，先带你去一窥它的全貌。

现在，让我们首先新建一个 ADF 的模板工程，这个模板工程是 ESRI 使用 ADF 开发的一个简单的 Web 应用供你参考 ,它包含了一些最常用和最基本的 GIS 功能 ;当然，它也必不可少会将 ADF 这个框架包含在这个应用中。

对了，迄今为止我们也没有讲怎么搭建 ADF 的开发环境，并且新建 ADF 模板工程来测试我们的开发插件是否安装成功。这些内容在我很久以前的文章中写过，我就不想在这里再重复一遍，分散我们对 ADF 的注意力了。这些内容，不管是基于 Eclipse 还是 NetBeans ,都可以从 ADF 的帮助文档中找到 ;你也可以参考附录的文档 :《配置 ArcGIS Server Java 开发环境（Eclipse/NetBeans 》——好了，现在我认为你已经配置好了开发环境，并且学会如何新建一个 ADF 模板工程了。另外，下面的讲解基于的 IDE 是

Eclipse。

让我们打开刚刚新建的 ADF 模板工程，展开项目的目录结构。我对其中一些重要的部分重点用图 2 表示了出来，下面让我们一级一级地展开。

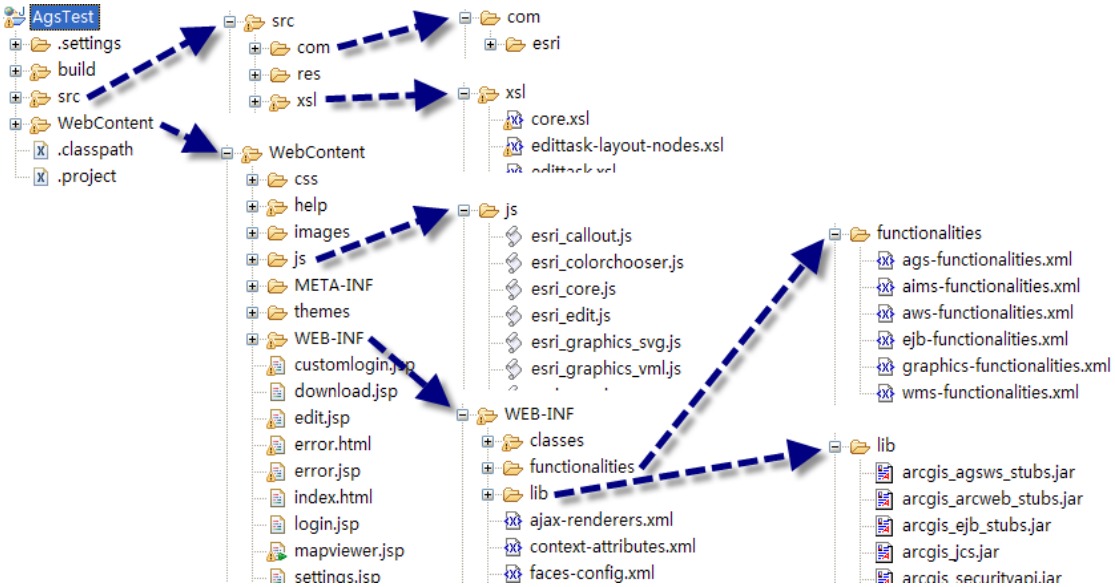


图 2　　ADF 模板工程的目录结构

在一个 ADF 的工程中，与开发相关的主要是“/src”目录和“/WebContent”目录。“/src”目录主要存放 Java 源代码相关内容，这些源代码最终会被编译后放到“/WebContent/WEB-INF/classes”目录中去；“/WebContent”目录主要存放最后需要部署的 Web 应用的内容。

在“/src”目录中，纯 Java 的代码一般会放到你自己的命名空间下，比如“/src/com/...”目录；“/src/xsl”是另外一个很重要的目录，其中包含许多 xsl 文件，ADF 通过这些 xsl 来渲染 ADF 组件 具体它是怎么工作的我们后续会做更详细的介绍。

现在转到“/WebContent”目录 ,ADF 本身带了一些“css”、“images”、“themes”目录内容来定义默认模板工程的样式；当然，还有一些“*.jsp”页面——事实上，在你自己做应用的时候基本上不需要用到这些页面。在此要详细解释的是“/WebContent/js”

和“/WebContent/WEB-INF”两个目录。

“/WebContent/js”目录包含了 ADF 的 JavaScript 函数库，主要是 ADF 在跟服务器做 AJAX 交互的时候需要用到，当然，你自己定义的 JavaScript 脚本也可以放到这个目录下；“/WebContent/WEB-INF”目录存放所有不希望被 Web 访问者访问到的文件和库，包括一些“*.xml”配置文件、“/WebContent/WEB-INF/classes”下的类文件（从“src”编译生成）、“/WebContent/WEB-INF/lib”下的 Java 库。

在这里我们稍微总结一下 ADF 的物理组成：ADF 的核心主要包括“/WebContent/WEB-INF/lib”目录下的 Java 库（底层功能、ADF 的 JSF 组件等）、“/WebContent/js”目录下的 JavaScript 脚本库（支持 AJAX）和“/WebContent/WEB-INF/classes/xsl”目录下的 XSL 样式表语言文件（渲染 ADF 组件）；除此之外，ADF 外围还有一些模板页面、CSS 文件、图片、主题等可用可不用的附赠品；最后，作为开发框架，ADF 还包括帮助文件和例子，虽然这里并没有介绍它们——因为它们并不会被放到工程中去。

三 ADF 的逻辑结构

尼采说：“我没有肉体，我只有灵魂”——照卫斯理的科幻那他就是一种不知道比地球人高级多少的外星生物形态——咱们就不搬出马老先生的辩证唯物主义去批判他了，反正咱 ADF 不一样，它可是有胳膊有腿、有思想有能力的四有框架。上面刚介绍完了 ADF 的物理组成，根据由表及里、由浅入深的原则，现在让我们开始认识一下 ADF 的逻辑结构。

在 ADF 的帮助中有一张 MVC 示意图，其中把视图和控制器合并在了一起，因为

ESRI 认为 ADF 的组件不仅仅在视图层，它们还影响到了 Web 应用流程，有控制器的功能。这种划分我感觉不太清晰，所以这里还是把控制器独立了出来。

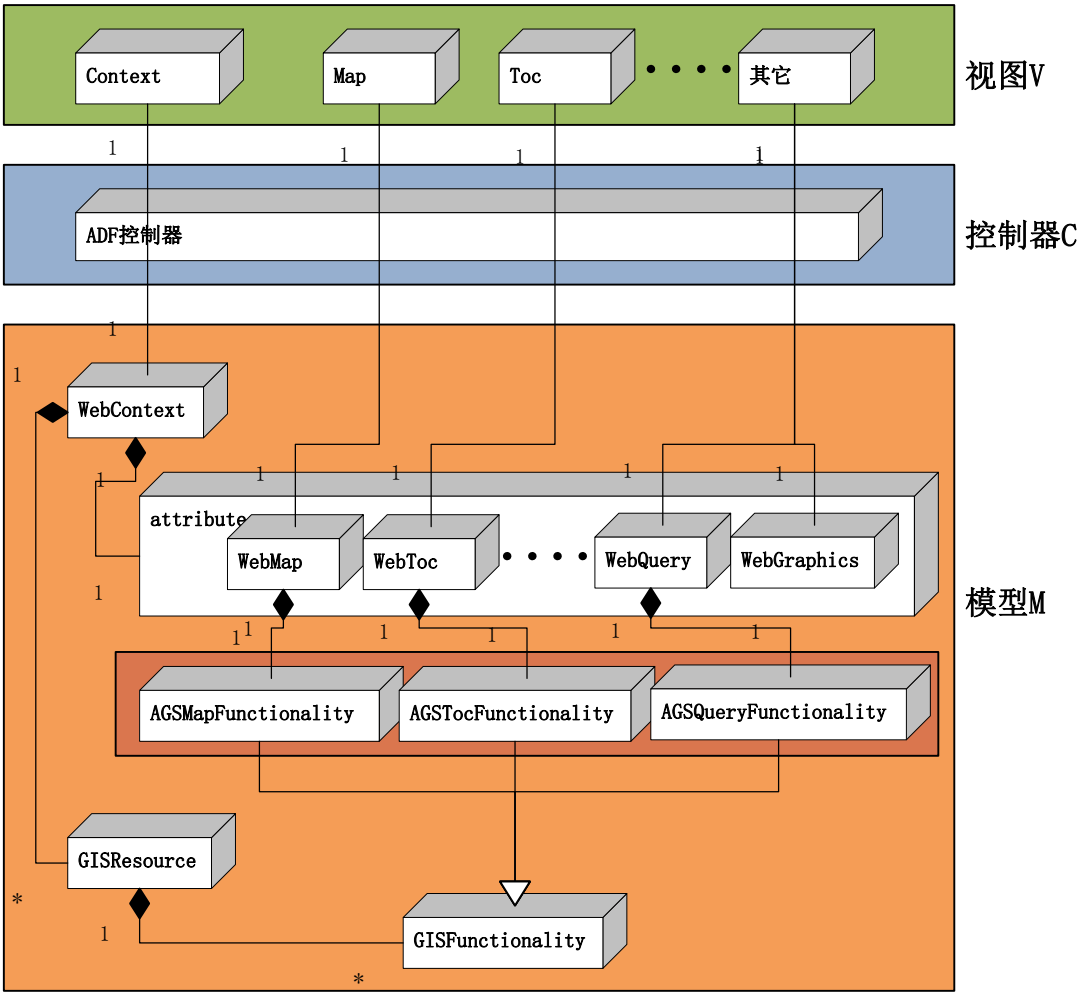


图 3　　ADF 的 MVC 示意图

如图 3 所示是我绘制的 MVC 示意图。ADF 在视图层的表现是一系列的 JSF 组件；它们是通过控制器层的 ADF 组件渲染器输出的 .最下层的是模型层。在 ADF 的模型中，WebContext 是最为重要的对象，不管是 GISResource（各种服务资源）还是其它的 WebControl（WebMap、WebToc 等等），都是在 WebContext 中进行管理的。WebContext 可以看作在用户会话中存在的管理 ADF 的一个环境，至于这个

WebContext 是从何而来，马上在下一小节中会详细叙述。

浏览器中用户所能看到的 Map、Toc 等，在服务器上都是以 WebMap、WebToc 等 JavaBean 对象存在于用户的会话中；同时，这些对象作为 WebContext 的属性（attribute）被其包含。而 Web 应用中所用到的 GIS 服务（如 Map Service、Geoprocessing Service 等）在 ADF 中都是作为 GISResource（如 AGSMapResource、AGSGPResource 等）来进行连接使用。这些不同的 GISResource 各自包含了各种不同的功能（GISFunctionality），这些不同的功能又会各自对应到 WebMap、WebToc 等对象上去。为什么需要在 ADF 的 WebControl 和 GISResource 之间加上 GISFunctionality 这一层呢？这主要是 ADF 为了统一不同服务数据源考虑，比如虽然 AGSMapResource 和 AGSLocalMapResource 是两种不同的 GISResource，但是它们都可以在 Map 组件中进行地图浏览，因此 ADF 定义了 AGSMapFunctionality 这个功能，AGSMapResource 和 AGSLocalMapResource 都分别实现了这个功能，Map 只要调用这个功能就可以实现地图的浏览而不必关心后台的服务到底是什么类型的。

上面这个 MVC 示意图目前有个大概印象就可以了，因为你还没有真正地使用 ADF，都理解了也不太可能；在后续的环节中，经常会过来看看这张图可能会有更加深入的理解，先就认个脸熟吧。

四 ADF 是怎样开始工作的

ADF 现在已经存在于我们的工程中了，接下来的问题就是它是怎么工作的了。换句话说，当我们的开发任务完成以后，把这个 ADF 项目部署到一个 Java Web 服务器上，当有浏览器对它发生请求的时候，我们的服务器怎么知道我这是一个 JSF 应用，进一步

还会根据 ArcGIS 的 ADF 来生成那些乱七八糟的诸如 WebContext 之类的对象呢？

我们首先来看一下，当一个用户对服务器发送请求的时候，Java Web 服务器怎么判断他要访问的是一个 JSF 应用。让我们打开“/WebContent/WEB-INF/web.xml”，这个“web.xml”文件是 Java Web 工程的部署描述文件，服务器会根据这个文件中的定义来配置这个应用，决定加载哪些 Servlet 以及其它的内容，在这个文件中我们可以分别找到下面两段定义：

```
<servlet>
  <servlet-name>Faces Servlet</servlet-name>
  <servlet-class>javax.faces.webapp.FacesServlet</servlet-class>
  <load-on-startup>1</load-on-startup>
</servlet>

<servlet-mapping>
  <servlet-name>Faces Servlet</servlet-name>
  <url-pattern>*.jsf</url-pattern>
</servlet-mapping>
```

“servlet”元素定义的是Java Web服务器加载的Servlet信息，“servlet-mapping”元素定义的是一个 URL 模式对应的 Servlet，所有符合该 URL 模式的请求都会被映射到该 Servlet 上进行处理。在这里首先我们根据 FacesServlet[1]这个类在服务器启动的时候加载了 JSF 的 Servlet：“Faces Servlet”，然后又定义了一个“*.jsf”的 URL 模式，凡是以“*.jsf”出现的请求都会被映射到“Faces Servlet”进行处理。好了，到这里我们应该可以明白 JSF 是怎么工作的了。

在此基础上，我们需要进一步了解 ADF 的运行。还是在刚才的“web.xml”文件中，从中我们还可以找到以下的这段内容：

```
<listener>
  <listener-class>
```

[1] javax.faces.webapp.FacesServlet

```
        com.esri.adf.web.util.ADFServletContextListener
    </listener-class>
</listener>
```

“listener”元素会根据相应的监听类配置一个监听。在这里模板工程为我们配置了一个 ADFServletContextListener 监听器，这个监听器实现了两个接口：ServletContextListener 和 HttpSessionListener。

ServletContextListener 负责监听 Web 应用（你部署的这个 ADF 工程）的 ServletContext 的变化。当我们的 Java Web 服务器启动的时候，我们的 Web 应用也会跟着启动，这时，Web 应用的 ServletContext 将会被创建并初始化，因此 ADFServletContextListener 的 contextInitialized(ServletContextEvent event)方法会被调用。在这个方法中，ADF 的 WebApplication[1]会被创建并初始化。这个 WebApplication 相当于我们 ADF 的女娲娘娘，以下我们会看到一个又一个的 ADF 对象会从中孕育出来。

WebApplication 中包含了一个由许多 WebSession[2]组成的一个 ArrayList，这个 ArrayList 中包含了所有的用户的 ADF 会话 WebSession；当然，这个 WebSession 与整个 Web 应用的会话 HttpSession[3]并不是独立的，WebSession 会作为一个属性存储在 HttpSession 中。

在这个 WebSession 中其实也是一片广阔的革命天地。在每个 WebSession 中都会有一个由 WebContext[4]组成的 ArrayList，它存放的是这个会话所用到的所有 WebContext 对象。还记得 WebContext 么？在《ADF 的逻辑结构》这小节中，

---

1 com.esri.adf.web.data.WebApplication

2 com.esri.adf.web.data.WebSession

3 javax.servlet.http.HttpSession

4 com.esri.adf.web.data.WebContext

WebContext 是以一个非常大哥的形象出现的。现在发现，这个大哥其实是 WebSession 的小弟之一，WebSession 则是 WebApplication 的小弟之一。

下面再说说 HttpSessionListener，它负责监听 Web 应用当前会话的变化。当有一个新的用户向我们的服务器发送请求的时候，应用会为这个用户新建一个会话，这个时候，ADFServletContextListener 的 sessionCreated(HttpSessionEvent event)方法也会被执行（虽然事实上，在这里 ADF 并不做任何事情，WebSession 并不是在这里创建，这个下面马上就讲）；当一个会话结束时，ADFServletContextListener 的 sessionDestroyed(HttpSessionEvent event)方法会被执行，ADF 将会从 Web 应用的 HttpSession 中取得 WebSession 对象，并将之从 WebApplication 中移除并销毁。

目前为止，只有 WebSession 的出身不太明白了。还是让我们打开“web.xml 文件”，其中有下面的一段：

```
<filter>
  <filter-name>ADFFilter</filter-name>
  <filter-class>com.esri.adf.web.util.ADFFilter</filter-class>
</filter>
```

这里定义了一个过滤器 ADFFilter，每当有用户请求时，请求都会经过 ADFFilter 过滤，这时，ADFFilter 的 doFilter(...)方法将会被执行。在这个方法中，ADF 会试图获得该请求的 HttpSession 中保存的 WebSession 对象，如果这个对象为 null，则新建一个 WebSession 对象并保存(同时保存引用到 HttpSession 和 WebApplication)。

最后，让我们对 ADF 的工作流程稍作梳理。ADF 的工作主要依赖于 Web 应用的 web.xml 中配置的 ADFServletContextListener 监听器和 ADFFilter 过滤器。当 Web 应用启动的时候，ADFServletContextListener 负责创建 WebApplication（老大）；当有用户请求的时候，请求会经过 ADFFilter 过滤并试图获得该请求的 WebSession(老

大的小弟），如果为 null 则创建 WebSession 并保存；在 WebSession 中包含了该会话使用的所有 WebContext（老大的小弟的小弟），有了 WebContext，剩下的事情都好办了。

以上是 ADF 的内部机制，如果理解有困难或者觉得有点麻烦，其实也没必要了解那么清楚，对 ADF 的几个重要对象的关系有所了解就可以了，下面这个简单的图可能会对你的理解有所帮助。

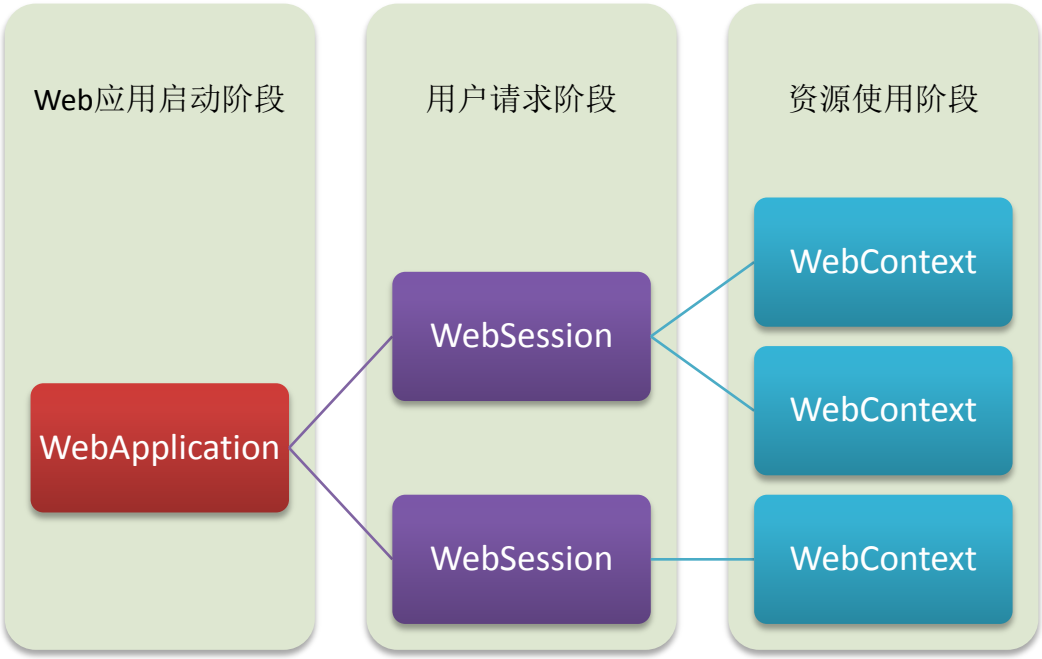


图 4　　ADF 几个重要对象的关系图

第三章 ADF 常用组件及原理分析

理论上讲，如果你愿意的话，完全可以不使用 ADF 的常用组件而完全自己写一个 WebGIS 应用。但是似乎目前还没有人这么做，因为 ADF 的这些组件已经封装了很多基础的 GIS 功能，这些功能没有必要重复地开发，并且开发的工作量巨大。ADF 的这些组件并不仅仅是 JSF 组件 ,它们还包含了非常多的 AJAX 操作 ,一些简单的地图功能，事实上后台的 ADF 做了很多工作。

这一章我们主要去了解 ADF 的常用组件，分析一下其工作机制，最后通过一个功能扩展案例来实践下对 ADF 组件的理解。

一 ADF 页面结构

在介绍 ADF 组件之前，我想还是先花点时间了解一下 ADF 的页面结构比较好，因为首先你要知道 ,使用 ADF 并不是简单地把这些组件拖到一个 JSF 页面上就可以用的。

打开模板工程的地图页面“mapviewer.jsp”，首先在页面顶端我们会找到对 ADF 标签库的引用，JSF 页面会通过这个标签库（前缀为“a”）来定义相应的 ADF 组件。

```
<%@taglib uri="http://www.esri.com/adf/web" prefix="a"%>
```

往下到页面的主体部分，所有的 ADF 组件都放在“f:view”标签内，这很好理解，因为我们的 ADF 组件是基于 JSF 的，因此也必须放在 JSF 的组件树结构中，“f:view”就是 JSF 组件树的根节点。我们可以看到，在“f:view”标签中放的第一个 ADF 组件就是“<a:context value="#{mapContext}" />”，它绑定到服务器上的 WebContext 对象“mapContext”，由它来管理其余所有的 ADF 组件及相关状态，“mapContext”对象的定义我们可以在“/WebContent/WEB-INF/faces-config.xml”中找到：

```
<managed-bean>
  <managed-bean-name>mapContext</managed-bean-name>
   <managed-bean-class>
        com.esri.adf.web.data.WebContext
   </managed-bean-class>
  <managed-bean-scope>request</managed-bean-scope>
</managed-bean>
```

顺便简单说一下这个“faces-config.xml”文件，这是 JSF 默认的配置文件。在一个 JSF 应用启动以后，Java Web 服务器会根据这个配置文件初始化相应的 JSF 对象。对于这个配置文件，我们在后面会多次接触到，届时会有更加详细的解释。

在“f:view”内部还有一个标签“h:form”，这代表的是一个纯 HTML 表单，所有需要向服务器提交数据的 JSF 组件必须放在这个表单下，因此，我们的 ADF 组件（除了“a:context”）也都需要放在“h:form”标签内。

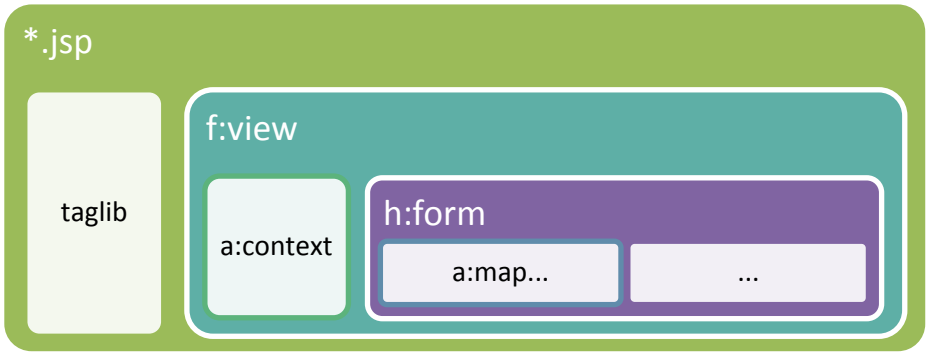


图 5　　ADF 页面主要元素

图 5 表示了上述这些页面元素的关系。一般来说，要让我们的 JSF 页面正确地加载 ADF 和其它 JSF 组件，并正常地工作，页面源代码大概如下所示：

```
<%@taglib uri="http://www.esri.com/adf/web" prefix="a"%>
<%@taglib uri="http://java.sun.com/jsf/core" prefix="f"%>
<%@taglib uri="http://java.sun.com/jsf/html" prefix="h"%>

<html>
 <f:view>
      <a:context value="#{mapContext}" />
      <body>
```

```
            <h:form>
                <a:map value="#{mapContext.webMap}" id="map" />
            </h:form>
        </body>
    </f:view>
</html>
```

二 ADF 中的服务

ADF 的后台都是各种不同的 ArcGIS 服务，一个 ADF 应用所依赖的就是这些服务提供的功能。这个小节让我们看一下服务是怎么配置到 ADF 应用，并和 ADF 对象关联起来的。

比如我现在发布了一个名为“World”的地图服务（Map Service），在 ADF 应用中准备通过 Internet 的方式进行连接。在当前的 JSP 文件上右键菜单，选择【ArcGIS】-【Configure Map Service...】后，如图 6 所示添加地图服务。

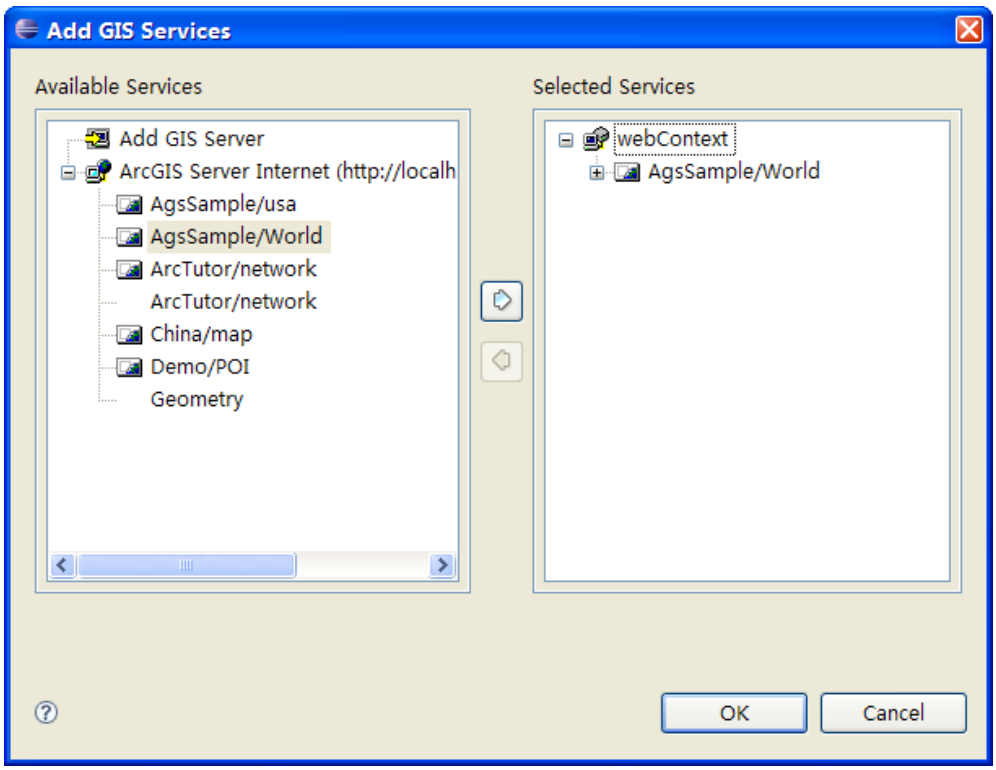


图 6　在当前应用中配置服务

在添加服务以后，让我们打开“faces-config.xml”来看一下 ArcGIS Eclipse Plugin 帮我们生成的配置（在此基础上我对一些名称作了修改）：

```
<managed-bean>
    <managed-bean-name>
        agswsWorld
    </managed-bean-name>
    <managed-bean-class>
        com.esri.adf.web.ags.data.AGSMapResource
    </managed-bean-class>
    <managed-bean-scope>
        none
    </managed-bean-scope>
    <managed-property>
        <property-name>endPointURL</property-name>
        <value>http://localhost:8399/arcgis/services/World/MapServer</value>
    </managed-property>
    <managed-property>
        <property-name>alias</property-name>
        <value>World</value>
    </managed-property>
    <managed-property>
        <property-name>functionalities</property-name>
        <map-entries>
            <map-entry>
                <key>map</key>
                <value>#{agsMap}</value>
            </map-entry>
            <map-entry>
                <key>overview</key>
                <value>#{agsOverview}</value>
            </map-entry>
            <map-entry>
                <key>toc</key>
                <value>#{agsToc}</value>
            </map-entry>
            <map-entry>
                <key>query</key>
                <value>#{agsQuery}</value>
            </map-entry>
        </map-entries>
    </managed-property>
</managed-bean>
```

在 ADF 中，服务是作为资源（Resource）登上舞台的。上面这段配置定义了一个名为 agswsWorld 的 AGSMapResource 对象，事实上，这个 Resource 对象关联的就是我们前面发布的“World”这个地图服务。在这里需要注意的是“functionalities”标签，“functionalities”中定义了当前这个资源可以使用的功能，比如一个地图服务显然可以进行查询，因此在我们的“functionalities”中定义了一个属性“query”，它指向一个名为“agsQuery”的 AGSQueryFunctionality[1]对象（打开“/WebContext/WEB-INF/ functionalities/ags-functionalities.xml”可以发现它的踪迹），它提供了地图查询功能。

这个 agswsWorld 最后会被添加到 WebContext 的 resources 属性中去。通常在“faces-config.xml”中会以如下的方式定义：

```
<managed-bean>
    <managed-bean-name>mapContext</managed-bean-name>
      <property-name>resources</property-name>
      <map-entries>
        <map-entry>
          <key>agswsWorld</key>
          <value>#{agswsWorld}</value>
        </map-entry>
      </map-entries>
    </managed-property>
  </managed-bean>
```

稍微总结一下，ADF 会根据不同的 ArcGIS 服务类型生成对应的 AGSxxxResource 对象进行关联，服务包含各种不同的功能可供使用。这些 Resource 对象根据需要会被添加到 WebContext 的“resources”属性中去进行管理和使用。以上这些关于 Resource 的内容默认都集中在“/WebContext/WEB-INF/“目录下的 “faces-config.xml”和“functionalities/*-functionalities.xml”中进行定义的。

[1] com.esri.adf.web.ags.data.query.AGSQueryFunctionality

三 几个 ADF 常用组件

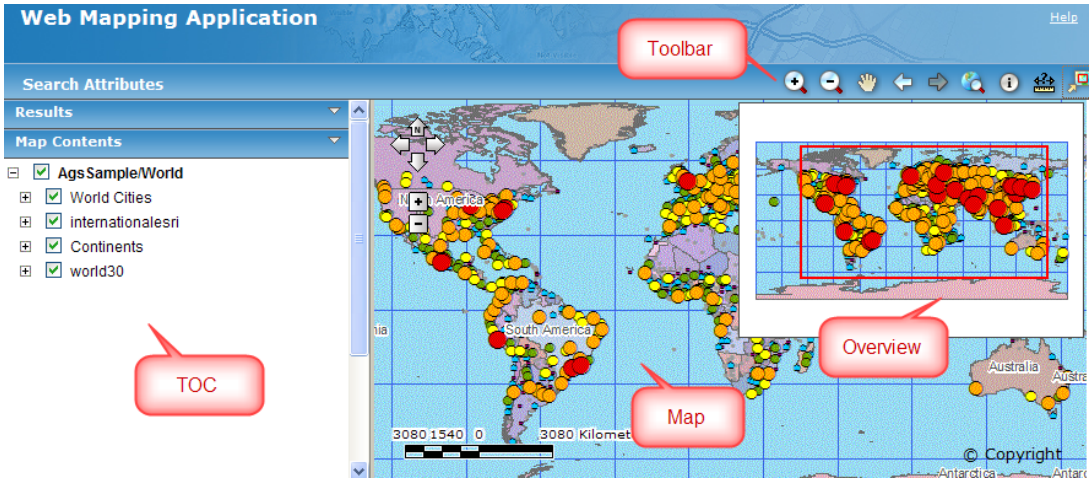


图 7 ADF 常用组件

前面我们已经学会了如何定义一个 ADF 页面，并配置相应的服务供 ADF 应用使用。下面就让我们熟悉一下 ADF 中几个非常常用的组件及其功能。

如图 7 所示，ADF 中几个与地图相关的组件的命名和功能都和 ArcEngine 很类似，主要包括 Map、Toolbar、TOC、Overview 等。我们先大致看一下这几个组件的用法和大概功能。

Map 应该是所有 ADF 应用中都要使用的组件，它是一个地图的容器，内置了对各种地图服务的使用和操作。它在服务器端绑定的是一个 WebMap[1]对象。以下标签在页面中添加了一个 Map 组件：

```
<a:map id="map" value="#{webContext.webMap}" width="800" height="500"
    scaleBar="#{webContext.attributes['scalebar']}" />
```

在这里还出现了 ScaleBar，也就是比例尺。在这里我们稍微看一下如何定义一个 ScaleBar 并对其进行定制：

```
<managed-bean>
    <managed-bean-name>scaleBar</managed-bean-name>
```

[1] com.esri.adf.web.data.WebMap

```
    <managed-bean-class>com.esri.adf.web.data.WebScaleBar</managed-bean-class>
    <managed-bean-scope>none</managed-bean-scope>
    <managed-property>
      <property-name>type</property-name>
      <value>SteppedScaleLine</value>
    </managed-property>
    <managed-property>
      <property-name>position</property-name>
      <value>bottom-right</value>
    </managed-property>
</managed-bean>
```

其中 type 定义了比例尺的样式，positon 定义了比例尺的位置。可以设置的 position 值包括"bottom-left"、"bottom-right"、"top-left"、"top-right"和"none"，这个很好理解。下面的列表则包含了可以设置的 type 值及其效果：

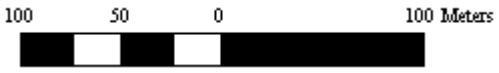
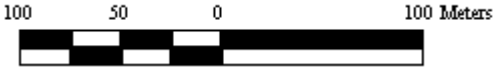
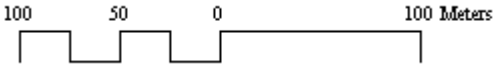
ScaleBar 效果	**type 值**
	Alternating
	DoubleAlternating
	SingleDivision
	SteppedScaleLine

表 1　　ScaleBar 可用样式及其效果

Toolbar 是绑定到 Map 的工具条，在 Toolbar 中可以添加 Tool 和 Command 以实现相应的地图操作 ,而它本身仅仅是一个容器。以下标签在页面中添加了一个 Toolbar 组件，同时绑定到 id 为“map”的 Map 组件，并在其中放置了一个可以设置当前地图操作为“Pan”的 Tool：

```
<a:toolbar id="toolbar" mapId="map" activeTool="pan" >
      <a:tool id="pan" toolTip="浏览"
      defaultImage="images/tasks/maptools/pan.png"
      hoverImage="images/tasks/maptools/panU.png"
      selectedImage="images/tasks/maptools/panD.png"
```

```
        clientAction="EsriMapContinuousPan"
        serverAction="com.esri.adf.web.faces.event.PanToolAction"
        clientPostBack="true"/>
</a:toolbar>
```

TOC 是 Table Of Contents 的缩写，它提供一个面板来显示当前地图的图层信息，并可以进行隐藏/显示图层、缩放到图层等操作。它在服务器端绑定的是一个 WebToc[1] 对象。以下标签在页面中添加了一个 TOC 组件，同时绑定到 id 为“map”的 Map 组件：

```
<a:toc id="toc" mapId="map" value="#{webContext.webToc}" />
```

Overview 提供鹰眼图功能，它在一个全图的地图上用一个矩形框来表示当前地图的视图范围。它在服务器端绑定的是一个 WebOverview[2]对象。以下标签在页面中添加了一个 Overview 组件，同时绑定到 id 为“map”的 Map 组件：

```
<a:overview id="overview" mapId="map" value="#{webContext.webOverview}"/>
```

四 Map 组件详解

以上几个组件的使用非常简单，不需要很多设置就可以搭起一个有模有样的 WebGIS 页面了——当然，这是因为 ADF 在后台为我们做了很多事情。后面我们希望从 Map 组件入手，看看在一个会话中，ADF 是怎么把后台的 ArcGIS 服务呈现到用户的浏览器，并使之能响应用户操作的。

当一个用户会话开始时，让我们首先看一下 Map 组件在服务器端绑定的 WebMap 对象，WebMap 对象的定义我们可以在“/WebContent/WEB-INF/context-attributes.xml”中找到。在前面我们已经知道，

[1] com.esri.adf.web.data.WebToc

[2] com.esri.adf.web.data.WebOverview

一个 JSF 应用在启动的时候会读取“faces-config.xml”文件来配置应用；这里的“context-attributes.xml”虽然不是默认配置文件，但是在这里它也会被读取，因为在“/WebContent/WEB-INF/web.xml”中有如下的配置：

```
<context-param>
  <param-name>javax.faces.CONFIG_FILES</param-name>
  <param-value>
    /WEB-INF/functionalities/aims-functionalities.xml,
    /WEB-INF/functionalities/ags-functionalities.xml,
    /WEB-INF/functionalities/aws-functionalities.xml,
    /WEB-INF/functionalities/wms-functionalities.xml,
    /WEB-INF/functionalities/ejb-functionalities.xml,
    /WEB-INF/functionalities/graphics-functionalities.xml,
    /WEB-INF/ajax-renderers.xml,
    /WEB-INF/context-resources.xml,
    /WEB-INF/context-attributes.xml
  </param-value>
</context-param>
```

前面我们已经提过，“web.xml”是 Java Web 工程的部署描述文件，上述这段“web.xml”中的配置会指定 JSF 应用启动时需要读取的若干配置文件路径，“context-attributes.xml”赫然名列其中。对了，你也可以自己新建一个配置文件，然后在这里进行一下配置，比如我习惯新建一个“context-resources.xml”文件来定义 WebContext 需要的所有服务资源，然后再在“web.xml”中添加如上虚线框中的内容。

在“context-attributes.xml”中首先就是对 WebMap 的定义：

```
<managed-bean>
  <managed-bean-name>map</managed-bean-name>
  <managed-bean-class>com.esri.adf.web.data.WebMap</managed-bean-class>
  <managed-bean-scope>none</managed-bean-scope>
  <managed-property>
    <property-name>imageFormat</property-name>
    <value>PNG24</value>
  </managed-property>
</managed-bean>
```

这段配置文件的含义就是：在应用中，在有需要的时候会实例化一个名为 map 的 WebMap 对象供使用，并且其地图输出格式为 24 位的 PNG 图片。

从图 3 的结构我们可以看到，这个 WebMap 对象将会被放置到 WebContext 的 attributes 中进行使用和管理。比如按照下面这段“faces-config.xml”中的定义，当得到一个 WebContext 对象 mapContext 以后，我们就可以通过 mapContext.attributes["map"]就可以得到 WebMap 对象了：

```
<managed-bean>
  <managed-bean-name>mapContext</managed-bean-name>
  <managed-bean-class>com.esri.adf.web.data.WebContext</managed-bean-class>
  <managed-bean-scope>request</managed-bean-scope>
  <managed-property>
    <property-name>attributes</property-name>
    <map-entries>
      <map-entry>
        <key>map</key>
        <value>#{map}</value>
      </map-entry>
    </map-entries>
  </managed-property>
</managed-bean>
```

好，目前为止我们已经知道了在一个会话中，Map 组件在服务器上绑定的 WebMap 对象如何生成，并且如何引用了。下面让我们看看 Map 组件和 WebMap 怎么配合渲染成浏览器中的一个地图的。

前面在 JSP 页面中我们是通过“a:map”标签来定义一个 Map 组件的，实际在 ADF 中，其对应的 JSF 组件为 MapControl[1]。MapControl 中包含 WebMap、MapRenderer[2]、XslUrl 等属性，WebMap 负责维护地图的功能，MapRenderer 负责将其输出成 XML，

[1] com.esri.adf.web.faces.component.MapControl

[2] 事实上有两个 MapRenderer：com.esri.adf.web.faces.renderkit.xml.MapRenderer 和 com.esri.adf.web.faces.renderkit.xml.ajax.MapRenderer，分别在初次请求和后续 AJAX 请求时使用

最后再通过 XslUrl 指定的 XSL 文件将 XML 解释为 DHTML 输出。

打开“/src/xsl”目录，我们会发现下面有很多“*.xsl”文件，其中的“map.xsl”就是我们的 MapControl 渲染时候默认使用的 XSL 文件。当有用户向服务器发送请求的时候，JSF 会首先创建或者还原 MapControl 这个视图对象，然后再使用其 MapRenderer 按照 XSL 文件渲染成 DHTML 响应输出给浏览器。

你可以运行下你的应用，然后查看地图页面的源代码，从中会发现类似以下的代码，这些就是 Map 组件的 MapRenderer 渲染的。

```
<!--Map Control Renderer Start (map)-->
    ......
    var map = new EsriMap("map", null, "_id1", 400, 300);
    ......
<!--Map Control Renderer End (map)-->
```

在这里出现了 EsriMap 这个 JavaScript 类， 它将会实例化浏览器端的地图对象。在下面这一节里，我们将会对 EsriMap 做详细解释。这里先稍微做一下总结，如图 8，借用帮助里的一张图片，应该可以帮助你了解 ADF 组件的基本原理。

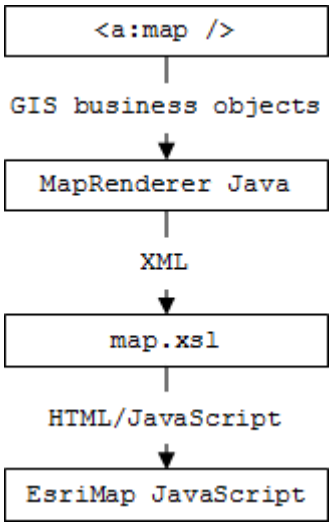


图 8　　Map 组件的生命周期

五 EsriMap 详解

> 提示：本小节所涉及内容主要由 ADF 自己完成，分析只是为了更好地了解原理，为将来有可能做的扩展进行准备。

EsriMap 在“/WebContent/js/esri_map.js”中定义，让我们从 Map 组件输出的 JavaScript 代码开始去了解 ADF 在浏览器中所做的工作。

下面的代码是一个简单地图输出的 JavaScript 代码（为了便于阅读，我对某些代码作了等价的修改），其中“AgsTest”是我的项目名称：

```
<!--Map Control Renderer Start (map)-->

<script type="text/javascript" language="Javascript" src="/AgsTest/js/esri_core.js"/>
<script type="text/javascript" language="Javascript" src="/AgsTest/js/esri_callout.js"/>
<script type="text/javascript" language="Javascript" src="/AgsTest/js/esri_maptip.js"/>
<script type="text/javascript" language="Javascript" src="/AgsTest/js/esri_upload.js"/>
<link type="text/css" rel="stylesheet" href="/AgsTest/css/esri_styles.css">
<input type="hidden" name="__ADFPostBack__" value="true">
<script type="text/javascript" language="Javascript"
src="/AgsTest/js/esri_graphics_vml.js"/>

<script type="text/javascript" language="Javascript">
          var esriInitItems = new Array();
          var userOnload = window.onload;
          window.onload = esriInitApp;
          EsriControls.contextPath = "/AgsTest/";

          function esriInitApp(e)
      {
            var mapInitializers = new Array();
            for (var i=0;i<esriInitItems.length;i++)
        {
              if (esriInitItems[i].toString().indexOf("esriMapInit") == 0)
            {
                eval(esriInitItems[i]);
                mapInitializers.push(i);
              }
            }
```

```
            for (var i=0;i<esriInitItems.length;i++)
        {
            if (mapInitializers.indexOf(i) == -1)
            {
                eval(esriInitItems[i]);
            }
            }
            if (userOnload)
            userOnload(e);
        }
    </script>

<script type="text/javascript" language="Javascript" src="/AgsTest/js/esri_map.js"/>

<div id="EsriMapCell_map" style="; width:400px; height:300px;"></div>

<script type="text/javascript" language="Javascript">
        function esriMapInitmap()
  {
            var map = new EsriMap("map", null, "_id1", 400, 300);
            map.isFuseGraphics = false;
            map.enableMapTips = true;
            var ms = new Array();
            var msNames = new Array();
            var msImages = new Array();
            mapSource1 = new EsriMapSourceDynamic(400, 300);
            msImages.push("/AgsTest/mimedata;jsessionid=8A5018B8B749317BC08B99B651A
3D288?wname=esriWebSession&id=map15177785&uniqueId=21018877-107");
            msNames.push("mapSource1");
            ms.push(mapSource1);

            for (var i=(msNames.length-1);i>=0;i--)
      {
            ms[i] = map.addMapSource(msNames[i], ms[i], i==0);
            }
            map.init(document.getElementById("EsriMapCell_map"));

            for (var i=(msNames.length-1);i>=0;i--)
      {
            if (msImages[i] != "")
            ms[i].addImage(0, 0, ms[i].generateTileId(0, 0), msImages[i]);
            }

        }
```

```
        esriInitItems.push("esriMapInitmap()");
    </script>

<!--Map Control Renderer End (map)-->
```

现在让我们开始看浏览器一步一步执行的 JavaScript 代码。除了引入一些 ADF 的 JavaScript 脚本，首先将会执行的是位于最上面的一些变量声明和位于最下面的 esriInitItems.push("esriMapInitmap()")代码，这里会将 esriMapInitmap()这个函数添加到 esriInitItems 这个初始化数组里面去；然后，在页面加载完成后会触发 window.onload 事件，以致调用 esriInitApp(e)函数；这时，由于 esriMapInitmap()已经被添加到 esriInitItems 中，因此这个函数将被调用。下面让我们仔细看看在 esriMapInitmap()中发生了些什么。

首先第一件事情就是实例化一个 EsriMap 对象，这一步其实有很多内涵，马上就会详细解释；其次，ADF 会针对地图中最重要的数据源进行配置，我们在上面的代码可以看到，由于我们使用的是动态 Map Service，因此 ADF 会新建一个 EsriMapSourceDynamic（继承自 EsriMapSource）对象，并通过 EsriMap 的 addMapSource()方法添加到地图中；然后，addImage()方法会添加这个 EsriMapSource 对象的图片地址，可以看到，这个图片地址是一个相对路径，以下这个地址表明 EsriMapSource 中使用的图片格式是 MIME[1]数据，jsessionid 参数指示了当前会话的 id：

```
/AgsTest/mimedata;jsessionid=5E13F6727CAA0433934A85EFA1AB8D49?wname=esriWebSession&id=map28171097&uniqueId=24691856-0
```

下面回过头去详细看一下在实例化 EsriMap 的时候所发生的事情，这一切相关的操作都封装在了“esri_map.js”这个文件中了 ,我们可以打开这个文件来看一下 EsriMap

[1] 可参考：http://zh.wikipedia.org/wiki/MIME

实例化的详细内容。

首先，EsriMap 是继承自 EsriControl，关于 EsriControl 及其相关的内容，有兴趣可以在 ADF JavaScript 库中继续深入研究。事实上我们对 EsriMap 感兴趣的应该主要是两点：其一，地图图片在 EsriMap 中的放置关系；其二，对地图做操作时候 EsriMap 与 ArcGIS Server 的通讯。

我们先来看下 EsriMap 中地图图片的放置关系。EsriMap 最外层的 div 是 container，默认 id 是“EsriMapCell_map”（在“map.xsl”中定义），它是放置地图的容器，一般不作操作；在 container 中还有几个层次的 div：divObject、controlDiv、imageGrid，它们和 container 的关系如图 9 所示。地图图片会放在 imageGrid 中。

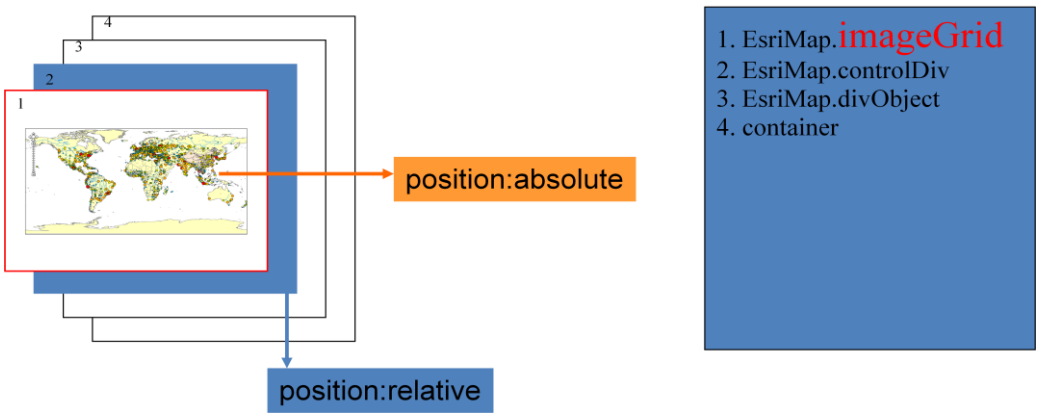


图 9　　EsriMap 中放置图片的 div

下面我们以动态地图为例，看一下当拖动地图时 EsriMap 是如何与 ArcGIS Server 通讯并更新图片的。我们首先找到 EsriMap 中有以下这段代码：

```
EsriControls.addPostBackTagHandler("map", EsriControls.maps[self.id].updateAsync);
```

EsriControls 在“/WebContent/js/esri_core.js”中定义，它负责管理当前页面上所有的 ADF 的 DHTML 组件。以上的代码给 EsriMap 对象添加了一个回调函数 updateAsync，它将会在页面收到 AJAX 响应并且响应中有“map”标签的时候被调用。

当我们拖动地图的时候，客户端的 JavaScript 会将需要的地图范围发给服务器（有兴趣可以参考 EsriMapPan 和 EsriMapContinuousPan 函数），服务器处理完成后会返回如下的响应：

```
<?xml version="1.0" encoding="UTF-8"?>
<response>
 <form>
      <id>_id1</id>
      <eventsource>map</eventsource>
      <com.sun.faces.VIEW>_id142:_id149</com.sun.faces.VIEW>
 </form>
 <map>
      <id>map</id>
      <width>400</width>
      <height>300</height>
      <control-fuse-graphics>false</control-fuse-graphics>
      <enable-info-window>true</enable-info-window>
      <zoom-beyond-cache>false</zoom-beyond-cache>
      <data-sources>
           <data-source>
                <dynamic>
                     <image-url>/AgsTest/mimedata;jsessionid=A03663701DD9557F1C73C
37D3FDE49A0?wname=esriWebSession&amp;id=map6509601&amp;uniqueId=16476003-3
2</image-url>
                </dynamic>
           </data-source>
      </data-sources>
 </map>
 <mapviewer-elements>
          <north-arrow-url>/AgsTest/mimedata;jsessionid=A03663701DD9557F1C73C37D3
FDE49A0?wname=esriWebSession&amp;id=mime31838215&amp;uniqueId=16476003-28<
/north-arrow-url>
 </mapviewer-elements>
</response>
```

这时，updateAsync 函数就会被调用，它会更新页面上 EsriMap 相关的一系列 div 和其它对象的属性值，从而更新地图视图。具体更新的细节，有兴趣可以详细研究下 updateAsync 函数，这里就不赘述了。

从这个 Map 组件的服务器端对象和客户端对象，我们已经基本了解了 Map 的工

作原理。ADF 的其余几个常用组件：TOC、Toolbar、Overview 等也是万变不离其宗，虽各有特点，但大体思路和原理也大概如此，有兴趣的话你还可以做更深入的研究。

六 给地图加上弹出气泡【案例】

在这一小节里，我们将会在地图上加一个气泡，很多时候你需要弹出一些简短的信息到地图上，气泡就是为了满足这样的要求的。

首先让我们回忆下这一章的主要内容，前面讲了 ADF 一些常用的地图组件，然后从 Map 组件入手详细分析了它在服务器端和浏览器端的具体实现。现在我们的气泡功能主要是对浏览器端的 Map 进行功能扩展，那么首先应该想到的就是 EsriMap 这个 JavaScript 类，我们在 EsriMap 中弹出一个 div，再放上我需要的信息不就可以了么？

等等，在页面上弹出一个 div 是很简单，但是用户还会操作地图，比如拖动一下啊什么的，那这个 div 怎么跟地图一起移动呢？再回想下图 9，你应该可以得到答案吧？对了，把这个 div 放到 imageGrid 中去。

下面让我们来具体实现这个功能，先看一下效果吧，如图 10。

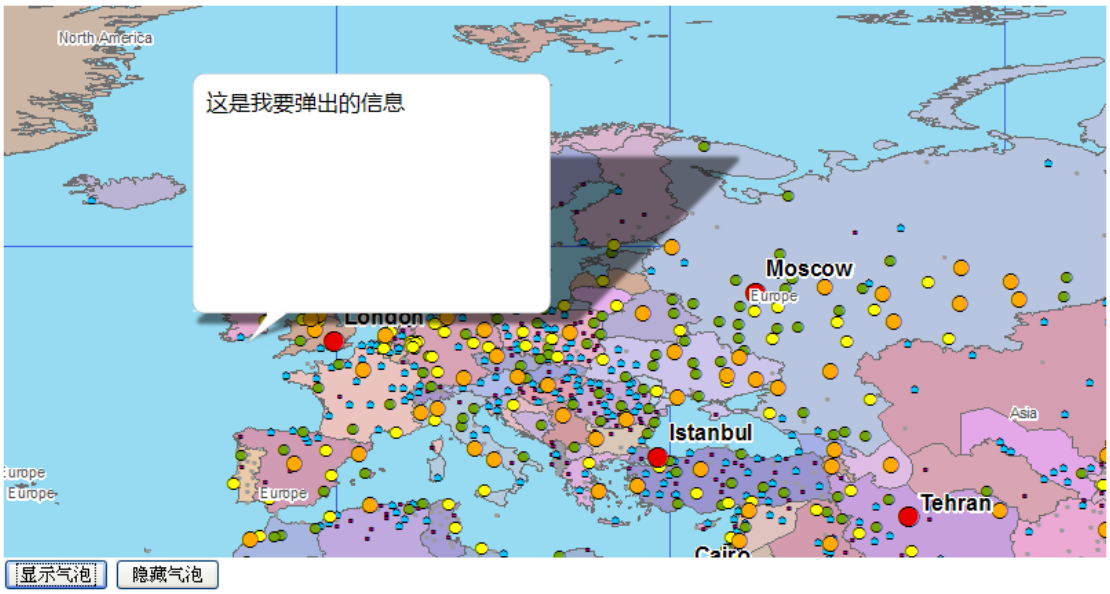


图 10　地图弹出气泡效果

在这个简单的页面上，我们放置了两个按钮。当点击“显示气泡”按钮的时候，地图上会弹出一个提示框，提示框中有我们想要显示给用户的信息；当点击“隐藏气泡”按钮的时候，这个提示框会隐藏掉；另外，当我们拖动地图的时候，这个气泡也会跟着地图一起移动。

在这里主要封装了两个 JavaScript 函数，分别绑定到“显示气泡”和“隐藏气泡”按钮的 onclick 事件上。

我们先来看一下“显示气泡”按钮的事件，它将会触发 showPopUp 函数，这个函数执行的操作如下：

```
function showPopUp()
{
 var popUpId = 'divPopUp';
 var popUp = document.getElementById(popUpId);
 if( popUp==null )
 {
        popUp = document.createElement('div');
        popUp.id = popUpId;
        popUp.innerHTML = "这是我要弹出的信息";
        var map = EsriControls.maps['map'];
```

```
        map.imageGrid.appendChild(popUp);
 }
 var style = 'display:block;position:absolute;z-index:2;';
 style += 'left:100; top:100;';
 style += 'background-image:url(images/wuyf/popup.png);width:400px; height:200px;';
 style += 'background-repeat:no-repeat;';
 style += 'padding:10px;';
 EsriUtils.setElementStyle(popUp, style);
}
```

首先，我们试图通过 getElementById()方法去获得气泡 div 是否已经存在；如果不存在，则新建这个 div 并添加到地图上。我们看到，把气泡 div（id 为 divPopUp）添加到地图上的过程实际上就是把这个 div 添加到 EsriMap 的 imageGrid 中去，这主要是通过 appendChild(popUp)这个方法实现的。

仅仅把这个 div 添加到 imageGrid 以后并不能看到这个它，在下面还有一些其它的设置，否则我们的气泡 div 会被地图遮挡住。这里主要的概念是 CSS 中的 z-index 属性，使用这个属性来控制 imageGrid 中各个子元素的叠置顺序，imageGrid 中地图图片的 z-index 值为 1 ,因此我们需要给气泡 div 设置一个比 1 大的数值才可以。另外，在气泡 div 的样式中我们还必须通过“display:block;”来设置这个 div 可见，再通过“position:absolute;”来设置这个 div 的定位模式，这些都是必须的。

顺便提一句，这里使用的是 EsriUtils 的 setElementStyle()方法来给气泡 div 来设置 CSS 属性；你直接用 obj.style.attributes=value;的方法来设置也是一样的。

再来看一下“隐藏气泡”按钮的事件，这就简单多了：

```
function hidePopUp()
{
 var popUpId = 'divPopUp';
 var popUp = document.getElementById(popUpId);
 if( popUp!=null )
 {
        EsriUtils.setElementStyle(popUp, "display:none;");
```

```
 }
}
```

这里事实上只做了一件事情，就是把气泡 div 的 display 样式设置成“none”，浏览器就不会去渲染这个气泡了。

通过这个小案例，你是不是对 ADF 中的 Map 组件，特别是 JavaScript 脚本库中的 EsriMap 有了更加深入的理解了？

第四章 Command、Tool 和 Button

这一章我们将讨论的主要对象是 Toolbar 的孩子们和亲戚，涉及到两个孩子：Command 和 Tool，还有一个亲戚：Button。它们是基于 ADF 对 WebGIS 功能进行扩展的第一步，它可以让你在地图工具条中加入若干自定义的按钮来执行特定的功能。

在第三章中我们把 Toolbar 列为 ADF 常用组件之一，但是它和 Map、TOC、Overview 等其它组件是有区别的，我们打开“context-attributes.xml”配置文件可以发现，Map、TOC、Overview 等组件都有相应的 JavaBean 定义，但是 Toolbar 没有——因为它不需要在服务器端绑定一个“WebToolbar”对象来记录 Toolbar 的状态信息。

下面让我们分别来看看这些组件的用法吧。

一 Command

Command 在 Toolbar 中表现为一个按钮，当用户点击这个按钮以后，ADF 将会立即执行一些操作。点击 Command 相当于直接给 ADF 一个命令，它并不改变当前地图的状态，比如当前地图正在进行漫游，点击 Command 以后再对地图进行操作时它还是漫游状态。

最典型的 Command 比如“地图全图”功能，当用户点击以后，当前地图会缩放到全图显示。下面的代码定义了这样一个 Command。

```
<a:toolbar id="toolbar" mapId="map" orientation="VERTICAL" >
    <a:command id="zoomFullExtent"
    defaultImage="images/tasks/maptools/fullext.png"
    hoverImage="images/tasks/maptools/fullextU.png"
    selectedImage="images/tasks/maptools/fullextD.png"
    clientPostBack="true" >
```

```
        <f:actionListener type="com.esri.adf.web.faces.event.ZoomFullExtentListener"/>
    </a:command>
</a:toolbar>
```

从这段代码我们可以看到，Command 的最关键的用法在于定义其子元素“f:actionListener”，它的“type”属性指向一个实现了某些功能的监听器类（注意，并不是对象）。这里的“ZoomFullExtentListener”是 ADF 自带的监听器；当然，这个类也可以是任何实现了 ActionListener[1]接口的类，我们在后面的案例会实现我们自己的监听器类。

下面我们再来看两个稍微有点不同的 Command 定义。这两个 Command 的功能是提供两个按钮“后退”、“前进”，当点击“后退”时地图会返回到上一视图范围；当点击“前进”时地图会前进到后一视图范围。下面的代码定义了这样两个 Command。

```
<a:toolbar id="toolbar" mapId="map" orientation="VERTICAL" >
 <a:command id="back"
 defaultImage="images/tasks/maptools/back.png"
 hoverImage="images/tasks/maptools/backU.png"
 selectedImage="images/tasks/maptools/backD.png"
 disabledImage="images/tasks/maptools/backX.png"
 disabled="#{!webContext.attributes['history'].canUndo}"
 action="#{webContext.attributes['history'].doPrevious}"
 clientPostBack="true" />
 <a:command id="forward"
 defaultImage="images/tasks/maptools/forward.png"
 hoverImage="images/tasks/maptools/forwardU.png"
 selectedImage="images/tasks/maptools/forwardD.png"
 disabledImage="images/tasks/maptools/forwardX.png"
 disabled="#{!webContext.attributes['history'].canRedo}"
 action="#{webContext.attributes['history'].doNext}"
 clientPostBack="true" />
</a:toolbar>
```

这里并没有使用监听器，而是使用了 Command 的 action 属性。这个属性将绑定一个服务器端对象的方法，当点击这个 Command 的时候，ADF 会去调用这个服务器

[1] javax.faces.event.ActionListener

端方法。“后退”、“前进”的功能主要使用了 ADF 中的 ExtentHistory[1]对象，它可以维护地图的视图范围历史记录。下面它是在“context-attributes.xml”中的定义：

```
<managed-bean>
    <managed-bean-name>history</managed-bean-name>
    <managed-bean-class>com.esri.adf.web.data.ExtentHistory</managed-bean-class>
    <managed-bean-scope>none</managed-bean-scope>
</managed-bean>
```

当我们点击“后退”按钮的时候，ADF 会调用这个服务器端“history”对象的 doPrevious 方法，因此地图会切换到上一视图；点击“前进”也是类似。另外，这两个 Command 的 disabled 属性分别绑定到“history”对象的 canUndo 和 canRedo 属性，这也很好理解，当地图已经是最新的视图的时候，用户当然应该不可以点击“前进”这个按钮，此时“history”的 canRedo 属性值为 TRUE，因此“前进”这个按钮会呈现不可用的状态。

稍微总结一下，上面主要介绍了 Command 的两种定义用法，一种定义监听器类作为 Command 的子元素 ;另外一种绑定服务器端方法到 Command 的 action 属性。对于自定义 Command 来说，一般使用监听器会比较容易一些。

二 自定义 Command 使地图等比放大【案例】

在这一小节，我们通过自定义 Command 定制一个按钮，这个按钮每点击一次，地图会放大 1 倍。通过这个简单案例，让我们来熟悉一下 Command 的用法。

实现这个功能的主要环节是实现自己的监听器类，以下是这个类的定义：

```
import com.esri.adf.web.data.WebContext;
import com.esri.adf.web.util.WebUtil;
import javax.faces.event.AbortProcessingException;
```

[1] com.esri.adf.web.data.ExtentHistory

```
import javax.faces.event.ActionEvent;
import javax.faces.event.ActionListener;

public class ZoomInListener implements ActionListener{
    public static double ZOOM_FACTOR = 0.5;

    public void processAction(ActionEvent event) throws AbortProcessingException {
        WebContext webContext = WebUtil.getWebContext(event.getComponent());
        webContext.getWebMap().getCurrentExtent().expand(ZOOM_FACTOR);
        webContext.refresh();
    }
}
```

当然，在 JSF 页面中的用法和上面类似，也很简单：

```
<a:command id="zoomIn"
defaultImage="images/tasks/maptools/zoomin.png"
hoverImage="images/tasks/maptools/zoominU.png"
selectedImage="images/tasks/maptools/zoominD.png"
clientPostBack="true" >
    <f:actionListener type="wuyf.ZoomInListener"/>
</a:command>
```

这个ZoomInListener监听器类主要需要实现了ActionListener的processAction方法，当用户点击这个 id 为“zoomIn”的 Command 的时候，这个监听器会被触发，然后其 processAction 方法就会被执行。

在这个 processAction 方法中，我们首先获得 WebContext 对象，通过 WebContext 我们就可以获得 WebMap 对象，进而对它的 currentExtent 属性进行操作（地图范围缩小 1 倍），再刷新 WebContext 对象。这个过程很简单，中间有两个环节这里稍作解释。

一个环节是如何获得 WebContext 对象。processAction 是 JSF 的 ActionListener 接口的一个方法，这个方法的参数是 ActionEvent[1]，因此它的属性与 ADF 基本没有关系，所以，获得 WebContext 就需要用到 ADF 的 WebUtil 工具类，ESRI 的工程师为

[1] javax.faces.event.ActionEvent

了可以在各种环境下获得 WebContext 这个 ADF 中非常重要的环境，制造了 WebUtil 这个工具类及其 getWebContext 方法，通过这个方法我们可以很方便地获得 WebContext 对象。

另外一个环节就是 WebContext 对象的 refresh 方法，我们在对服务器端的 WebMap 对象进行一些操作以后，调用了 WebContext 的 refresh 方法以后，浏览器中的地图就会刷新成新的视图，其相关的 TOC、Overview 等组件也会跟着刷新，这很奇妙吧。这样吧，由于 WebContext 的 refresh 方法在 Command、Tool 包括后面的 Task 中都会被使用到，下面我们就花点时间好好的看一下 WebContext 的 refresh 方法做了什么事情吧。

三 WebContext 的 refresh 方法

> 提示：本小节内容是 ADF 内部实现机制，对 ADF 的使用没有影响，如没有兴趣可以跳过。

当调用 WebContext 的 refresh 方法的时候， ADF 会同步 WebContext 中所有相关属性。比如当我改变了 WebMap 的视图范围 然后再调用 WebContext 的 refresh 方法，那么 ADF 会通知 WebContext 中的 WebOverview 等对象去同步相关信息，比如同步改变视图范围等。以上过程 ADF 在后台是通过 Observer 机制实现的。当 ADF 调用 WebContext 的 refresh 方法时，ADF 会通知所有注册到 WebContext 的 WebContextObserver[1]对象，调用其 update 方法。实现 WebContextObserver 这个接口的大概有如下这些类：AWSTocFunctionality, ExtentHistory, MapToolsTask,

[1] com.esri.adf.web.data.WebObserver

PrintTask, RoutingTask, SearchAttributesTask, WebMap, WebOverview, WebScaleBar。

以上对 WebContext 的刷新都是在服务器端。但是，当一个 Command 被点击以后，服务器端的更新又是怎么被输出到浏览器端的呢？下面就让我们从浏览器端开始，详细地了解一下 Command 的工作原理。

以下是上一节的案例中用a:command标签在JSF页面中定义了一个Command，在下面我们看一下这段标签在浏览器中相应的输出：

```
<a:command id="zoomIn"
defaultImage="images/tasks/maptools/zoomin.png"
hoverImage="images/tasks/maptools/zoominU.png"
selectedImage="images/tasks/maptools/zoominD.png"
clientPostBack="true" >
    <f:actionListener type="wuyf.ZoomInListener"/>
</a:command>
```

以下是上面的标签在浏览器中的输出：

```
var zoomIn = new EsriMapServerAction("zoomIn", "");
zoomIn.clientPostBack = true;
zoomIn.defaultImage = "/AgsDemo/images/tasks/maptools/zoomin.png";
zoomIn.hoverImage = "/AgsDemo/images/tasks/maptools/zoominU.png";
zoomIn.selectedImage = "/AgsDemo/images/tasks/maptools/zoominD.png";
zoomIn.isDisabled = false;
zoomIn.showLoading = true;
toolbar.addToolItem(zoomIn);
```

可见，一个 Command 对应输出的是 EsriMapServerAction 这个 JavaScript 对象，这个对象在“/WebContent/js/esri_map.js”中定义。打开 EsriMapServerAction 的定义你会发现，当点击这个 Command 的时候，ADF 会提交当前页面；当然，由于我们设置了 clientPostBack 属性为 TRUE，因此交互通过 AJAX 方式进行，其回调函数为 EsriControls 的 processPostBack 函数——EsriControls 在“/WebContent/js/esri_core.js”中定义，它包含了所有的 ADF 客户端组件。

提示：以上这段这里说的很简单，其实包含很多的内容，有兴趣的话可以在这几个 JavaScript 脚本中仔细研究相关对象和调用关系。

ADF 在服务器端会有一个 PostBackPhaseListener[1]去监听所有的 AJAX 请求，有兴趣的话你可以解开“/WebContent/WEB-INF/lib/arcgis_webcontrols.jar”这个库文件，在其中的“META-INF”目录下有一个“faces-config.xml”中定义了这个 PhaseListener。

AJAX 请求的参数中会有一个“doPostBack”值，当一个 AJAX 请求发到服务器的时候，PostBackPhaseListener 会监听到这个请求并找到“doPostBack”这个值从而执行后续的操作。当这些操作完成以后，PostBackPhaseListener 通过 AJAXUtil[2]的 writeResponse 方法输出 AJAX 响应到请求的浏览器。

四 Tool

Tool 是另外一种按钮，虽然它在外观上和 Command 可能没什么区别，但是当 Tool 被点击以后，地图状态就进行了改变。比如我当前默认对地图的操作是漫游，通过点击一个 Tool，接下来在地图上的点击可能就变成了放大操作——与地图交互，这就是 Tool 的作用。

以下代码定义了一个“使地图处于漫游状态”的 Tool：

```
<a:toolbar id="toolbar" mapId="map" orientation="VERTICAL" >
    <a:tool id="pan"
    toolTip="浏览"
    defaultImage="images/tasks/maptools/pan.png"
    hoverImage="images/tasks/maptools/panU.png"
```

[1] com.esri.adf.web.faces.event. PostBackPhaseListener

[2] com.esri.adf.web.faces.renderkit.xml.ajax.AJAXUtil

```
        selectedImage="images/tasks/maptools/panD.png"
        clientAction="EsriMapContinuousPan"
        serverAction="com.esri.adf.web.faces.event.PanToolAction"
        clientPostBack="true"/>
</a:toolbar>
```

Tool 定义时两个最重要的属性就是 clientAction 和 serverAction。clientAction 定义的是在浏览器端执行的操作，比如这里的“EsriMapContinuousPan”代表浏览器端执行的是连续漫游操作——当然，比如你需要在地图里画个多边形之类的操作也可以使用其它的定义，可用的操作大概如表 2 所列：

EsriEditingLine 编辑直线	EsriMapLine 画直线
EsriEditingPoint 编辑点	EsriMapMouseWheel 滚轮滚动
EsriEditingPolygon 编辑多边形	EsriMapOval 画椭圆
EsriEditingPolyline 编辑多线	EsriMapPan 漫游
EsriMapCircle 画圆	EsriMapPoint 画点
EsriMapContinuousPan 连续漫游	EsriMapPolygon 画多边形
EsriMapImage 添加图片	EsriMapPolyline 画多线
EsriMapKeyNavigation 键盘导航	EsriMapRectangle 画矩形

表 2　　浏览器端支持的操作列表

当用户在浏览器执行了操作以后，ADF 会把这个操作相关的信息发送到服务器，然后 Tool 的 serverAction 属性中定义的这个类就起作用了。比如上面的这个 Tool，服务器接到请求以后会通过 PanToolAction 的定义对请求过来的参数进行处理（比如向右漫游 100 个像素）经过一系列计算以后它更新服务器端的 WebMap 等对象的状态，所有工作完成以后再调用 WebContext 的 refresh 方法进行刷新，随后输出响应到浏览

器。

serverAction 指向的类主要需要实现 MapToolAction[1]接口，在下面的案例中，我们会实现一个自己的 ToolAction 类。

五 自定义 Tool 实现点击地图居中【案例】

在这一小节中 ,我们通过自定义一个 Tool 来激活这样的地图功能 :每次点击地图，都以这个点击点为中心使地图居中。

如下，我们新建了这样一个类 CenterAtToolAction：

```
import com.esri.adf.web.data.WebContext;
import com.esri.adf.web.data.geometry.WebPoint;
import com.esri.adf.web.faces.event.MapEvent;
import com.esri.adf.web.faces.event.MapToolAction;

public class CenterAtToolAction implements MapToolAction{
    public void execute(MapEvent event) throws Exception    {
        WebContext webContext = event.getWebContext();
        WebPoint webPoint =
(WebPoint)event.getWebGeometry().toMapGeometry(webContext.getWebMap());
        webContext.getWebMap().centerAt(webPoint, 0.5);
        webContext.refresh();
    }
}
```

这个类的关键是实现了 MapToolAction 的 execute 方法，当一个 Tool 被激活，并在地图上进行了操作以后，一个请求就会被发送到服务器，对应的 MapToolAction 的 execute 方法就会被执行。

这里，我们通过 MapEvent 的获得 WebContext 和 WebGeometry2对象。比如我们在浏览器中点击了一下，那么 WebGeometry 就是一个点对象，当然，这个点坐

[1] com.esri.adf.web.faces.event.MapToolAction

[2] com.esri.adf.web.data.geometry.WebGeometry

标是基于屏幕坐标的，我们需要调用它的 toMapGeometry 方法转化成地理坐标。

当这些准备工作完成以后，我们就可以使用 WebMap 的 centerAt 方法使地图居中，随后刷新 WebContext 输出响应到浏览器。

如图 11 是点击居中前后的地图。

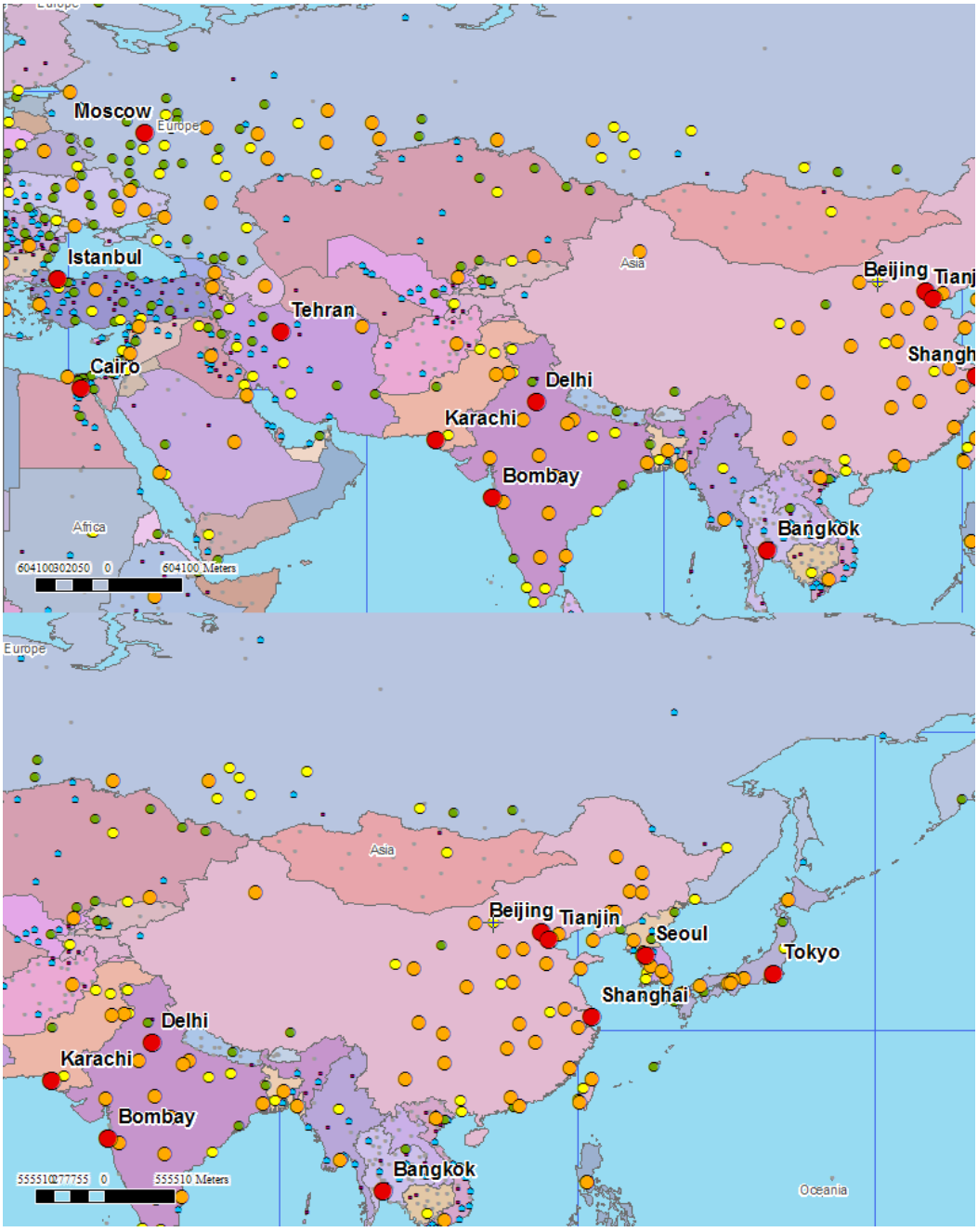


图 11　点击居中前后的地图

六 Button

Button 不属于 Toolbar 的子元素，而是和 Toolbar 有类似的地位；但同时，它的功能又集合了 Command 和 Tool。Button 的出现主要是为了方便用户，当用户不希望出现一个工具条来摆放与地图相关的功能，那么 Button 可以在页面的任意地方摆放一个与地图相关的按钮。

在 ADF 工程里，让我们打开 ADF 自带的模板页面“edit.jsp”，在里面有很多 Button。以下是定义了一个在客户端绘制一个矩形，服务器端执行“选择要素”操作的 Button：

```
<a:button id="select"
mapId="map1"
clientAction="EsriMapRectangle" serverAction="#{mapContext.attributes.mapEditor.select}"
defaultImage="./images/tasks/editing/selectfeature.gif"
hoverImage="./images/tasks/editing/selectfeatureU.gif"
selectedImage="./images/tasks/editing/selectfeatureD.gif"
/>
```

我们可以看到，Button 和 Command 以及 Tool 都很相像，有 clientAction 也有 serverAction，所不同的是，Button 需要指定一个 mapId 属性，因为 Command 和 Tool 都是存放在 Toolbar 中，Toolbar 会有个统一的 mapId 来表示和哪个 Map 绑定；而 Button 则由于是独立的按钮，需要自己指定与之绑定的 Map。

clientAction 属性是可选的，如果你不填 clientAction，那么这个 Button 的行为就类似于 Command；如果你设置了 clientAction，那么这个 Button 的行为就类似与 Tool。另外，这里的 serverAction 需要指向一个服务器端对象的方法，如果没有 clientAction，那么这个 serverAction 指向的方法应该是一个无参数的方法；如果有 clientAction，那么这个 serverAction 指向的方法应该是一个带 MapEvent[1]参数的方

[1] com.esri.adf.web.faces.event.MapEvent

法。

七 自定义 Button 实现前面的 Command 和 Tool【案例】

在这一章前面的小节中，我们实现了两个案例分别演示 Command 和 Tool 的功能；在这个案例里，让我们用 Button 来完成相同的工作。

首先，我们需要在一个我们自己定义的 ManagedBean[1]来实现这些功能。注意，在这个 ManagedBean 中我们会使用 WebContext 对象（否则怎么调用地图功能），因此，我们这个类定义需要实现 WebContextInitialize 接口。当然，在 IDE 中有更简单的方法，比如在 Eclipse 中选择菜单【File】-【New】-【Context Attribute】就可以新建一个 ManagedBean 并添加到 WebContext 对象的 attributes 中去。

[1] 前面出现过许多次的“服务器端对象”其实就是指 ManagedBean，被 JSF 管理并调用的服务器端 JavaBean 就是 ManagedBean，一般在配置文件中通过 managed-bean 标签定义

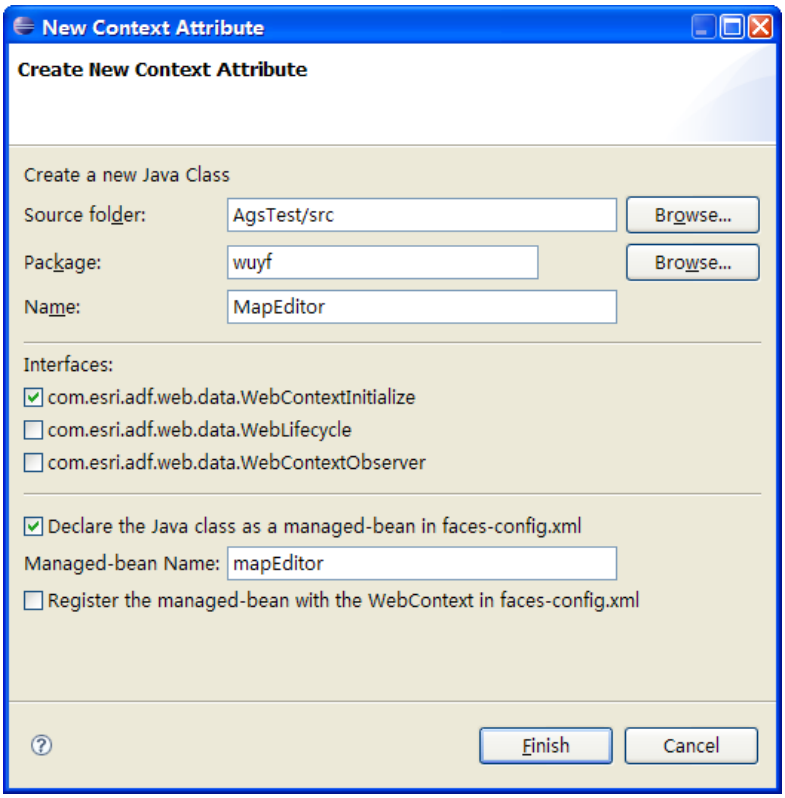


图 12　为 WebContext 添加一个 Attribute

好了，现在我们已经有了自己的 ManagedBean 类“MapEditor”；接下来我们先实现地图等比放大的功能。在这个类中 我们添加一个不带任何参数的方法 zoomIn，在 zoomIn 实现地图放大一倍的功能。源代码如下：

```
package wuyf;

import com.esri.adf.web.data.WebContext;
import com.esri.adf.web.data.WebContextInitialize;

public class MapEditor implements WebContextInitialize {
 private WebContext webContext;

 public void destroy() {}

 public void init(WebContext context) {
       this.webContext = context;
 }

 public void zoomIn(){
```

```
            webContext.getWebMap().getCurrentExtent().expand(0.5);
            webContext.refresh();
    }
}
```

在页面中用如下的标签进行定义：

```
<a:button mapId="map"
toolTip="放大按钮"
defaultImage="images/tasks/editing/point.gif"
hoverImage="images/tasks/editing/pointU.gif"
selectedImage="images/tasks/editing/pointD.gif"
serverAction="#{webContext.attributes['mapEditor'].zoomIn}"
/>
```

这个时候，你在页面上点击这个 Button 的时候，地图也会放大一倍了。

下面我们接着去实现点击地图居中，和上面类似，只不过实现这个功能的方法需要

带一个 MapEvent 参数：

```
public void centerAt(MapEvent event){
    WebPoint webPoint =
(WebPoint)event.getWebGeometry().toMapGeometry(webContext.getWebMap());
    webContext.getWebMap().centerAt(webPoint, 1.0);
    webContext.refresh();
}
```

在页面中定义的标签则如下：

```
<a:button mapId="map"
toolTip="点击地图居中按钮"
defaultImage="images/tasks/editing/point.gif"
hoverImage="images/tasks/editing/pointU.gif"
selectedImage="images/tasks/editing/pointD.gif"
clientAction="EsriMapPoint"
serverAction="#{webContext.attributes['mapEditor'].centerAt}"
/>
```

好了，试着点击这个按钮，再在地图上点击，是不是地图以点击处居中了？

第五章 Task

这一章我们主要研究一下 Task 同志的成份和出身。ADF 中本身包含一些现成的 Task，不过相比对开发人员来说没有很大的意义，这一章我们的重点还是在如何自定义 Task。

Task 是实现业务逻辑的重要部分。简单来说，Task 可以让输入一些参数（在页面上以文本框、组合框等形式出现）、通过 Command 和 Tool 的组合，帮助你完成特定的“任务”并返回结果，这就是所谓的 Task。

一 Task 的请求

虽说这一节的标题是 Task 的请求，但在这里我们对 ADF 的 Task 怎么向服务器做请求并不作深入的探讨，这里我们主要是去了解一个 Task 在服务器端执行的时候需要客户端提供的信息有哪些。



图 13 一个 Task 的请求面板

如图 13 是一个简单的 Task 请求面板，在这个面板中会有一些 Parameter（一般以文本框、组合框等形式出现）和一些 Command 或 Tool（一般以按钮形式出现）。下面我们先来定义一个 Task，其中包含一个 Parameter、一个 Command 和一个 Tool。

在 IDE 中新建一个 Task 最简单的方法就是使用菜单【File】-【New】-【Task】，然后给你的 Task 起个名字，如图 14。

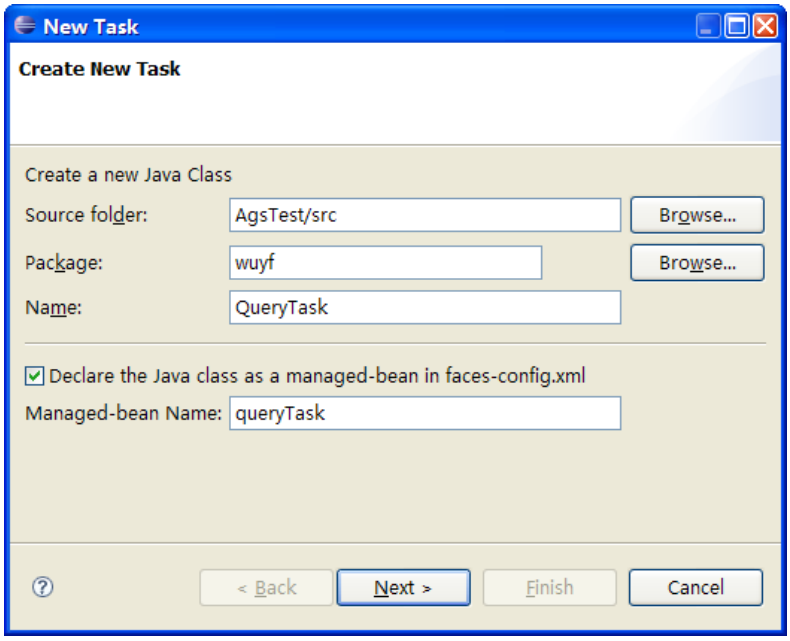


图 14　新建一个 Task

这个时候，IDE 会弹出另外一个对话框供你对这个 Task 进行一些设置，比如 Task 的名称、里面包含的 Parameter、Command（在这里叫 Action）、Tool 等。这里就根据你的需要进行填写了，我们先各添加一个，如图 15 所示。

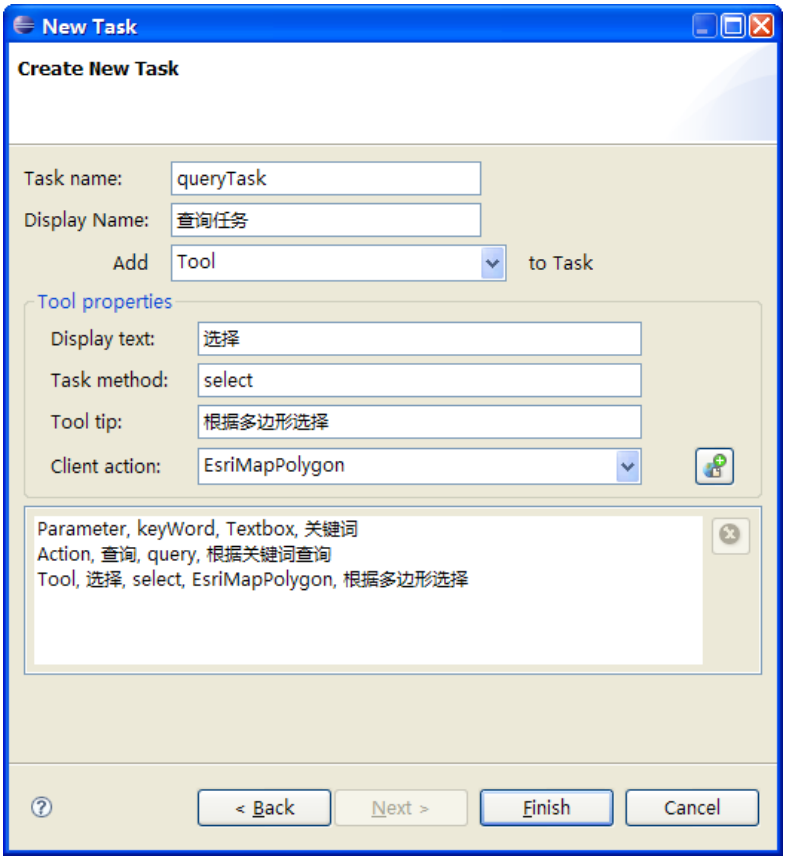


图 15　给 Task 添加一些元素

这时你会发现，在 Java 源文件目录下多了 2 个文件，一个是“QueryTask.java”，另一个是“QueryTaskTaskInfo.java”，同时在 faces-config.xml 文件中也多了一个名为 queryTask 的 ManagedBean。

打开“QueryTask.java”文件，我们可以找到和刚才定义的那些内容相对应的代码，下面的表简述了这样的一种对应关系：

	类型	名称
Parameter	String	keyWord
Command	Void 方法	query(TaskEvent event)
Tool	Void 方法	select(MapEvent event)

表 3　Task 对应的 Java 源文件中的定义

在“QueryTask.java”中我们还可以找到 QueryTaskTaskInfo 对象，关于 QueryTaskTaskInfo 这个类的定义及其作用我们在后面再详细说。下面让我们先把这个 Task 添加到页面上去：

```
<a:task id="queryTask"
taskInfo="#{queryTask.taskInfo}"
mapId="map"
clientPostBack="true"
windowingSupport="true"
value="#{webContext.attributes['queryTask']}" />
```

这里的 taskInfo 属性就是这个 Task 的 TaskInfo 对象，我们可以先不用管，唯一一个以前没有出现过的属性就是 windowingSupport，这个属性定义的是这个 Task 面板是否可以在页面上浮动并拖拽。让我们运行下这个页面看一下效果：

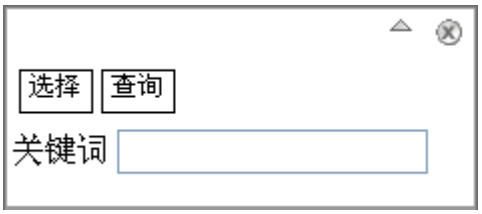


图 16　上面定义的 Task 面板的效果

稍微总结一下，Task 中主要包含 3 个功能元素：Parameter、Command、Tool；Task 对应的 Java 类有 2 个，其中在实现功能的 Task 类（比如上面的 QueryTask 类）中，Parameter 对应类中的属性字段、Command 对应类中的带 TaskEvent[1]参数的 void 方法、Tool 对应类中的带 MapEvent[2]参数的 void 方法。这样，前台的 Task 请求就和后台具体的功能联系起来了。

[1] com.esri.adf.web.faces.event.TaskEvent

[2] com.esri.adf.web.faces.event.MapEvent

二 Task 的执行

Task 的执行主要在对应的功能类（比如上面的 QueryTask 类）中实现，上面已经详述了 Task 中的 Parameter、Command、Tool 分别对应的功能类中的元素，下面就让我们看看怎么去使用它们。

在上面定义的那个 QueryTask 中，我们设计点击“查询”按钮就能根据文本框中输入的关键词去查找相应的要素。下面是实现这样功能的一段代码：

```
    public void query(TaskEvent event) {
        WebContext webContext = event.getWebContext();

        WebQuery webQuery = (WebQuery) webContext.getAttribute("query");
        TextCriteria textCriteria = new TextCriteria();
        textCriteria.setMaxRecordCount(100);
        textCriteria.setSearchText(this.keyWord);

        List<QueryResult> listQueryResult = webQuery.query(textCriteria,
webQuery.getQueryLayers());

        webContext.getWebGraphics().clearGraphics();
        for (int i = 0; i < listQueryResult.size(); i++) {
            QueryResult queryResult = listQueryResult.get(i);
            queryResult.highlight();
        }

        webContext.refresh();
    }
```

我们可以看到，从 TaskEvent 中可以直接获得我们的 WebContext 对象；从 WebContext 开始可以执行一系列的操作，其中用到了 keyWord 这个属性，keyWord 对应的就是用户在 Task 面板的文本框中输入的文字；最后查询到结果在地图上通过 Graphic 高亮出来，通过 WebContext 的 refresh 方法输出到浏览器。

这里涉及到了 Parameter 和 Command 的使用；至于 Tool，唯一的区别就是

WebContext 是从 MapEvent 中获得，因此这里就不详细写了。另外，这段代码中还涉及到如何进行查询、如何使用 Graphic 等，这些内容我们在后面会详细讲。

三 Task 的渲染

Task 的功能类中还有一个属性，IDE 默认生成的名字是 taskInfo，它对应这个另外一个 Java 类（如上面 QueryTaskTaskInfo 类）的对象。这个类主要控制 Task 的渲染，我们可以回想一下，如果光在 Task 的功能类中定义了 Command 是一个 void 类型的 query 方法，Task 面板怎么会生成一个标签为“查询”的按钮出来的呢？答案就在 taskInfo 中。

打开上面生成的“QueryTaskTaskInfo.java”文件，我们可以看到以下的内容：

```
public class QueryTaskTaskInfo extends SimpleTaskInfo {
 public TaskDescriptor getTaskDescriptor()  {
        TaskDescriptor descriptor = new TaskDescriptor(QueryTask.class, "queryTask", "查询任务");
        return descriptor;
 }

 public TaskParamDescriptorModel[] getParamDescriptors(){
        TaskParamDescriptor[] descriptors = new TaskParamDescriptor[1];
        descriptors[0] = new TaskParamDescriptor(QueryTask.class, "keyWord", "关键词", "getKeyWord", "setKeyWord");
        return descriptors;
 }

 public TaskActionDescriptorModel[] getActionDescriptors(){
        TaskActionDescriptor[] descriptors = new TaskActionDescriptor[1];
        descriptors[0] = new TaskActionDescriptor(QueryTask.class, "query", "查询");
        return descriptors;
 }

 public TaskToolDescriptorModel[] getToolDescriptors(){
        TaskToolDescriptor[] descriptors = new TaskToolDescriptor[1];
        descriptors[0] = new TaskToolDescriptor(QueryTask.class, "select", "选择",
```

```
"EsriMapPolygon");
        return descriptors;
 }
}
```

简单的说，TaskInfo 类中包含 4 个主要的属性：TaskDescriptor 对象、TaskParamDescriptor[]数组、TaskActionDescriptor[]数组、TaskToolDescriptor[]数组。很好理解，TaskDescriptor 对应整个 Task 的外观，比如 Task 的标题等；TaskParamDescriptor[]描述所有 Parameter 的外观；TaskActionDescriptor[]描述所有 Command 的外观；TaskToolDescriptor[]描述所有 Tool 的外观。一个 Task 中肯定包含不定数目的 Parameter、Command 或 Tool，因此他们的描述需要数组来存放。

让我们通过 TaskToolDescriptor 来更详细地看一下，上面的代码设置“select”这个 TaskTool 表现为一个“选择”按钮，并在点击时通知 Map 执行“EsriMapPolygon”操作。现在我想让它渲染成一个自定义的图片按钮，并随鼠标移上或点击切换不同的图片，用下面的代码可以实现这样的功能：

```
descriptors[0].setRendererType(TaskToolDescriptor.IMAGE_RENDERER_TYPE);
descriptors[0].setDefaultImage("images/tasks/maptools/polygon.gif");
descriptors[0].setHoverImage("images/tasks/maptools/polygonU.gif");
descriptors[0].setSelectedImage("images/tasks/maptools/polygonD.gif");
descriptors[0].setDisabledImage("images/tasks/maptools/polygonX.gif");
```

现在我还想调整一下这些按钮、文本框的布局，怎么办？TaskInfo 还可以指定 Task 各个子元素的布局，让我们在刚刚的 QueryTaskTaskInfo 类中再添加以下的代码：

```
public TaskLayout[] getTaskLayout(){
        TabularLayout[] descriptors = new TabularLayout[1];
 descriptors[0] = new TabularLayout();
        descriptors[0].addComponent(getParamDescriptors()[0], new TabularPosition(0, 0, 0,
0));
        descriptors[0].addComponent(getActionDescriptors()[0], new TabularPosition(1, 0, 0,
0));
        descriptors[0].addComponent(getToolDescriptors()[0], new TabularPosition(2, 0, 0, 0));
 return descriptors;
```

```
}
```

经过上面的两次修改，我们会发现 Task 面板的 Tool 变成了使用图片渲染，并且布局方式根据我们的定义进行了变化，效果如图 17 所示。



图 17　更改 Tool 渲染方式和布局后的 Task 面板

顺便回头再看看页面中 Task 的定义，其中有 taskInfo 属性，它对应的就是这个 Task 的 TaskInfo 对象：

```
taskInfo="#{queryTask.taskInfo}"
```

最后稍微总结一下吧，Task 的外观主要通过 TaskInfo 来定义，ADF 会根据 Task 所对应的 TaskInfo，通过 XSL 样式文件渲染成 DHTML 代码，并输出到浏览器。到现在为止，一个 Task 面板包含什么内容、对应后台什么对象、怎么控制它们的表现，你应该都清楚了吧？

四 定制 Task 布局的高级话题

Task 窗口中有很多工具是组合使用的，比如如果我有一个自定义的 CustomTask，它的有缩放和点击居中两个功能，其中“缩放比率”参数和“缩放”按钮是一组的，“选择点居中显示”工具是一组的。有时可能用户会希望每次只显示 Task 中相关的一组工具和按钮，当然，这种情况可以用多个 Task 来实现，这里则介绍另一种通过 Task 的布局来控制显示的方法。步骤如下：

首先，在 CustomTask 中增加一个参数，作为下拉列表，供用户选择布局方案：

```
private String layoutType;
 private Map<String,String> layoutOptions;

 public String getLayoutType() {
       return layoutType;
}

 public void setLayoutType(String layoutType) {
       this.layoutType = layoutType;
}

 public Map<String,String> getLayoutOptions(){
       if(layoutOptions == null){
             layoutOptions = new LinkedHashMap<String,String>();
             layoutOptions.put("action", "布局 1");layoutOptions.put("tool", "布局 2");
       }
       return layoutOptions;
}
```

然后，在 CustomTask 中增加一个方法（在 Task 窗口中增加一个按钮），用户可以通过这个方法调用 CustomTaskInfo.changeLayout 方法，这个方法目前还没有，它将会在后面的步骤中创建：

```
 public void changeLayout(){
       if("tool".equals(this.layoutType)){
             taskInfo.changeLayout("tool");
       }else if("action".equals(this.layoutType)){
             taskInfo.changeLayout("action");
       }
}
```

然后，我们继续在 CustomTaskInfo 中增加一个参数和按钮：

```
actions[1] = new TaskActionDescriptor(CustomTask.class,"changeLayout","布局切换");
params[1] = new TaskParamDescriptor(CustomTask.class,"layoutType","布局
","getLayoutOptions");
params[1].setRendererType(TaskParamDescriptor.SELECT_RENDERER_TYPE);
```

接着，我们在 CustomTaskInfo 中设置一个默认的布局，修改构造函数中与布局相关的初始化代码：

```
layouts[0] = new TabularLayout();
layouts[0].addComponent(params[1], new TabularPosition(0,0,1,1));
```

```
layouts[0].addComponent(actions[1], new TabularPosition(0,1,1,1));
layouts[0].addComponent(params[0], new TabularPosition(1,0,1,1));
layouts[0].addComponent(actions[0], new TabularPosition(1,1,1,1));
```

最后，在 CustomTaskInfo 中增加一个函数，通过它的输入参数来调整布局的样式：

```
/**
 * 修改 task 的布局，当参数值为 tool 时，显示点居中按钮；当参数值为其他时，
 * 显示缩放比率参数和缩放按钮
 * @param type
 */
public void changeLayout(String type){
    if("tool".equals(type)){
        layouts[0] = new TabularLayout();
        layouts[0].addComponent(params[1], new TabularPosition(0,0,1,1));
        layouts[0].addComponent(actions[1], new TabularPosition(0,1,1,1));
        layouts[0].addComponent(tools[0], new TabularPosition(1,0,1,1));
    }else{
        layouts[0] = new TabularLayout();
        layouts[0].addComponent(params[1], new TabularPosition(0,0,1,1));
        layouts[0].addComponent(actions[1], new TabularPosition(0,1,1,1));
        layouts[0].addComponent(params[0], new TabularPosition(1,0,1,1));
        layouts[0].addComponent(actions[0], new TabularPosition(1,1,1,1));
    }
}
```

以上的代码运行效果如图 18 和图 19 所示：

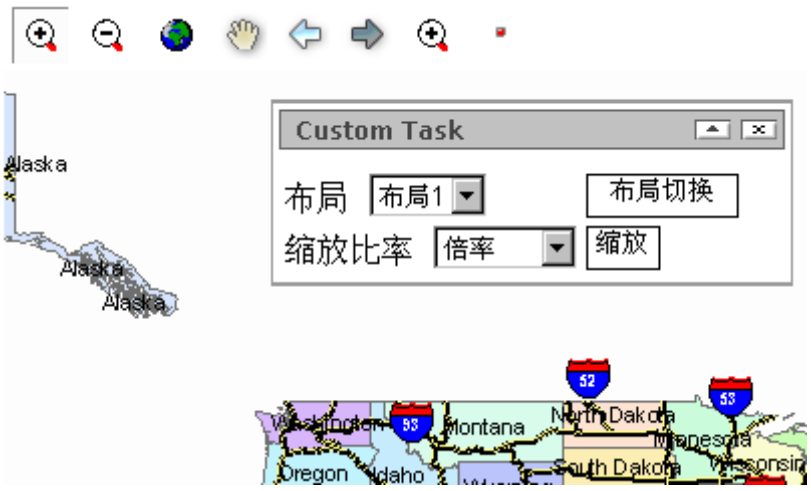


图 18　CustomTask 的默认布局

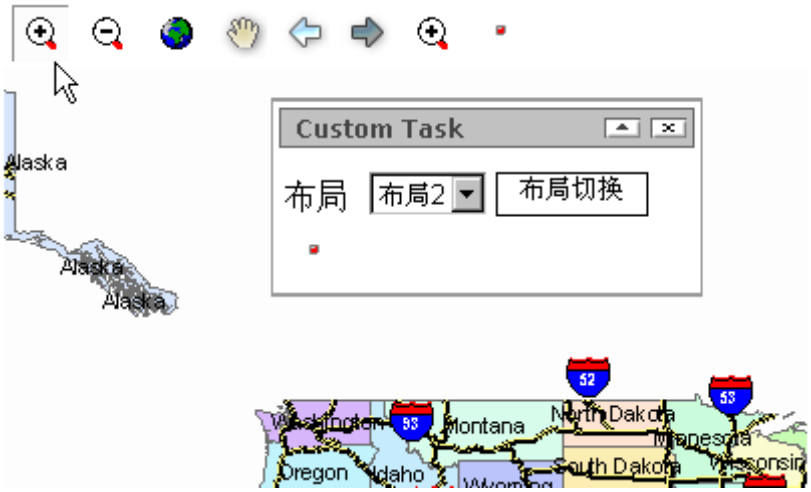


图 19　切换到 CutomTask 的第二种布局

第六章 查询及结果

对空间信息进行查询是 GIS 中很基本且很常用的功能，在这一章中，我们主要看在 ADF 中如何进行查询、查询出结果以后怎样返回客户端还有如何使用各种不同的符号来表示查询的结果。

一 文本查询

在上一章 QueryTask 的 query 方法中我们已经用到了文本查询的功能，你可以回忆一下，在这个方法中我们根据 Task 面板中文本框的内容，对图层进行了查询，用到的就是 keyWord 这个字符串变量。下面我们来详细说一下文本查询的过程。

在 ADF 中负责进行文本查询的是 WebQuery 对象，需要用到查询功能的 ADF 应用都需要在配置文件中加上了这个对象：

```
<managed-bean>
   <managed-bean-name>query</managed-bean-name>
   <managed-bean-class>com.esri.adf.web.data.query.WebQuery</managed-bean-class>
   <managed-bean-scope>none</managed-bean-scope>
</managed-bean>
```

当我们需要使用查询功能的时候，首先第一步就是获得这个 WebQuery 对象，当然，通常都是先获得 WebContext 对象，再查找它的 attribute 来获得 WebQuery 对象：

```
WebQuery webQuery = (WebQuery) webContext.getAttribute("query");
```

通过这个 WebQuery 对象我们就可以开始进行查询了，WebQuery 最重要的方法就是 query 方法，让我们先看一下 query 方法的原型：

```
public List<QueryResult> query(QueryCriteria criteria,
                                List<WebLayerInfo> queryLayers)
```

这个 query 方法主要有两个参数：criteria 参数表示查询条件，这是一个实现了

QueryCriteria[1]接口的对象，实现这个接口的类有三种：IdentifyCriteria, PredefinedQueryCriteria, TextCriteria，分别对应不同的查询类型，在这里我们将要用到 TextCriteria 这个类 ;query 方法的另外一个参数 queryLayers 表示需要进行查询的图层，一个服务中可能有多个图层，这个参数可以限制查询的范围。

上面说到进行查询需要一个实现了 QueryCriteria 接口的对象作为查询条件，这里我们需要查询文本，所以需要使用 TextCriteria，这是一个专为文本查询设计的类。下面的代码定义了一个 TextCriteria 对象，通过这个对象定义了一些查询条件和限制，最后通过 WebQuery 对象的 query 方法来得到查询结果：

```
WebQuery webQuery = (WebQuery) webContext.getAttribute("query");
TextCriteria textCriteria = new TextCriteria();
textCriteria.setMaxRecordCount(100);
textCriteria.setSearchText(this.keyWord);

List<QueryResult> listQueryResult = webQuery.query(textCriteria,
webQuery.getQueryLayers());
```

这是一个很简单的查询，事实上 TextCriteria 还可以设置很多限制和条件，比如限制查询字段、是否模糊查询、设置返回字段、设置 LayerDefinition[2]（包括字段别名、符号渲染方式）等等。最后，查询后会得到一个 QueryResult 的集合，关于这个 QueryResult 的特点和用法，我们在后面会有一个小节专门说明。

二 几何查询

在上一章的 QueryTask 中也有对几何对象的查询，你可以回忆一下那个 Task 面板中“选择”那个 Tool。这个 Tool 会在 Map 中绘制一个多边形，然后 ADF 根据这个多

[1] com.esri.adf.web.data.query.QueryCriteria

[2] com.esri.adf.web.data.query.LayerDefinition

边形查询位于多边形内的所有要素。

几何对象的查询主要使用 IdentifyCriteria 对象，下面的代码实现了这样一个根据多边形查询要素的功能：

```
WebContext webContext = event.getWebContext();
WebGeometry webGeometry = (WebPolygon)
event.getWebGeometry().toMapGeometry(webContext.getWebMap());

WebQuery webQuery = (WebQuery) webContext.getAttribute("query");
List<WebLayerInfo> layers = webQuery.getQueryLayers();
List<WebLayerInfo> queryLayers = new ArrayList<WebLayerInfo>();
for (WebLayerInfo layer : layers){
    if (layer.getName().equals("World Cities")){
        queryLayers.add(layer);
        break;
    }
}

IdentifyCriteria identifyCriteria = new IdentifyCriteria();
identifyCriteria.setWebGeometry(webGeometry);

List<QueryResult> listQueryResult = webQuery.query(identifyCriteria, queryLayers);
```

在这个查询过程中，首先从 MapEvent 事件中获得从客户端传来的 WebGeometry 对象（这里是用户在客户端的 Map 中画的多边形），这个对象是基于客户端坐标的，因此需要转化成地图坐标。由于我们只想查“World Cities”这个图层上的要素，因此下面又构造了一个 queryLayers，将我们想要查询的图层挑选了出来。在此之后，我们就可以使用 IdentifyCriteria 对象，将 ADF 转化成地图坐标的 WebGeometry 对象作为它的参数，进行查询得到结果。

三 Where 条件查询

三个实现了 QueryCriteria 接口的类我们还剩 PredefinedQueryCriteria 没有介绍，

这个类是用于进行 where 查询的。在进行查询的时候，通常需要设置一个比较复杂的过滤条件，这里就需要 PredefinedQueryCriteria。

下面我们尝试把前面使用 TextCriteria 查询的方法用 PredefinedQueryCriteria 来实现：

```
WebQuery webQuery = (WebQuery) webContext.getAttribute("query");

PredefinedQueryCriteria criteria = new PredefinedQueryCriteria();
criteria.setWhereClause("CITY_NAME like '%"+this.keyWord+"%'");

List<QueryResult> listQueryResult = webQuery.query(criteria, webQuery.getQueryLayers());
```

PredefinedQueryCriteria 的用法非常简单 ,无非是把你想要设置的 where 条件放到它的 whereClause 属性中去，这里就不赘述了。

四 QueryResult

上面几个小节让我们知道了不同的查询功能应该怎么去实现，每个查询中都返回了 QueryResult 对象的集合。这个小节我们就来详细说说 QueryResult 对象。

QueryResult 对象是一个查询结果，当然，这是一个服务器端对象，然而在服务器上 QueryResult 并不仅仅是一个记录，而是一个可以和 Map 互动，可以高亮显示、移除、缩放到的一个对象。

比如当我们做完查询以后，想在地图上显示所有的结果要素，那么，调用 QueryResult 的 highlight 方法就可以做到：

```
webContext.getWebGraphics().clearGraphics();
for (int i = 0; i < listQueryResult.size(); i++){
    QueryResult queryResult = listQueryResult.get(i);
    queryResult.highlight();
}
webContext.refresh();
```

上面的代码遍历了查询结果中所有的 QueryResult 对象，调用每个对象的 highlight 方法使其在地图上高亮显示，让我们看一下执行的结果：

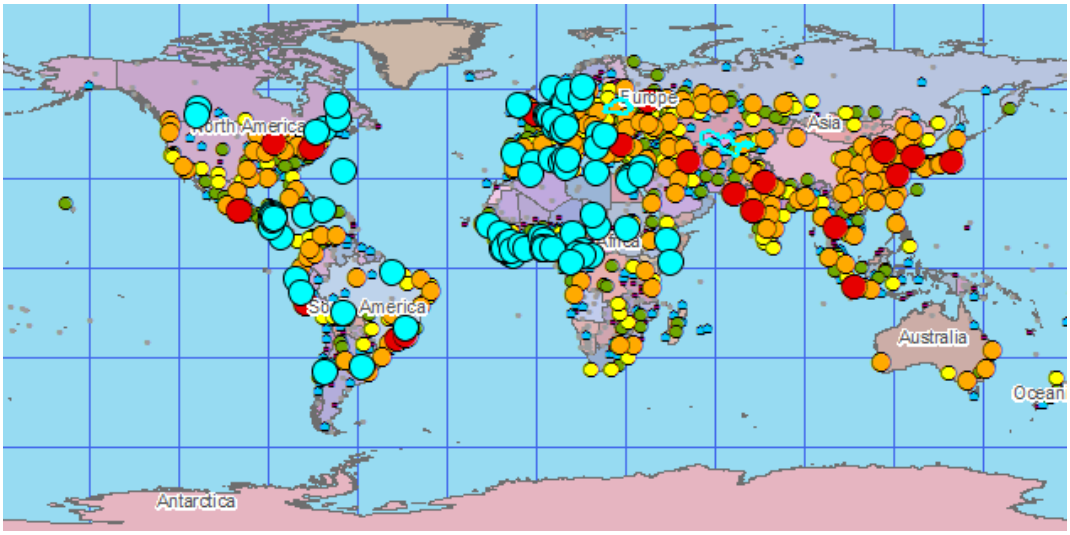


图 20　高亮查询结果

高亮的这些元素事实上是将这些查询结果以 GraphicElement[1]的形式添加到 Map 中的。ADF 的 GraphicElement 通常用于表现临时的、动态生成的业务数据，下面让我们会马上去看 GraphicElement 的来龙去脉。

五 GraphicElement

要说 GraphicElement 就得先说 WebGraphics[2]，欲用 GraphicElement，必先添加 WebGraphics，在你的配置文件中你应该可以找到如下的定义：

```
<managed-bean>
    <managed-bean-name>graphics</managed-bean-name>
    <managed-bean-class>com.esri.adf.web.data.WebGraphics</managed-bean-class>
    <managed-bean-scope>none</managed-bean-scope>
</managed-bean>
```

这个 WebGraphics 对象的作用就是在服务器端存放所有你需要动态表示的

[1] com.esri.adf.web.data.GraphicElement

[2] com.esri.adf.web.data.WebGraphics

GraphicElement。当然，现在我们已经很熟悉应该怎么去调用 WebGraphics 对象了：首先肯定要得到 WebContext 对象，然后在它的 attribute 中找到 WebGraphics 对象。

在 WebGraphics 对象中添加 GraphicElement 以后，这些 GraphicElement 就可以自动在 Map 中渲染出来，它们和后台服务没有关系，WebGraphics 中的对象会在后台服务的上层被绘制出来的。下面就让我们改写一下上一小节中的代码，在显示查询结果的时候不使用 QueryResult 的 highlight 方法，而是手动把每个 GraphicElement 添加到 WebGraphics 中去。

```
webGraphics.clearGraphics();
for (int i = 0; i < listQueryResult.size(); i++){
    QueryResult queryResult = listQueryResult.get(i);
    //queryResult.highlight();

    GraphicElement element = new GraphicElement();
    element.setGeometry(queryResult.getHighlightGeometry());
    element.setSymbol(symbol);
    webGraphics.addGraphics(element);
}
```

这里我们还通过 setSymbol 方法给这些 GraphicElement 设置了一个符号，这个符号的定义在上面的代码中被省略了，在下面一小节我们就重点看一下都有什么符号可以被 GraphicElement 使用吧。

六 GraphicElement 的符号化

GraphicElement 的 symbol 属性是一个 WebSymbol[1]对象，这个对象定义了使用什么符号来表示 GraphicElement 对象。下面让我们大概看一下有些什么 WebSymbol 可以为我使用：

[1] com.esri.adf.web.data.symbol.WebSymbol

WebSimpleMarkerSymbol	简单点符号
WebSimpleLineSymbol	简单线符号
WebSimplePolygonSymbol	简单多边形符号
WebPictureMarkerSymbol	图片点符号
WebTrueTypeMarkerSymbol	TrueType 点符号

比如一个简单点符号可以这样定义：

```
WebSimpleMarkerSymbol symbol = new WebSimpleMarkerSymbol();
symbol.setMarkerType(WebSimpleMarkerSymbol.STAR);
symbol.setAntialiasing(true);
symbol.setTransparency(0.7);
symbol.setColor("255,255,0");
symbol.setOutlineColor("255,0,0");
```

这样的代码定义的是一个星形点符号，在图 21 中你可以看到这个符号的效果：

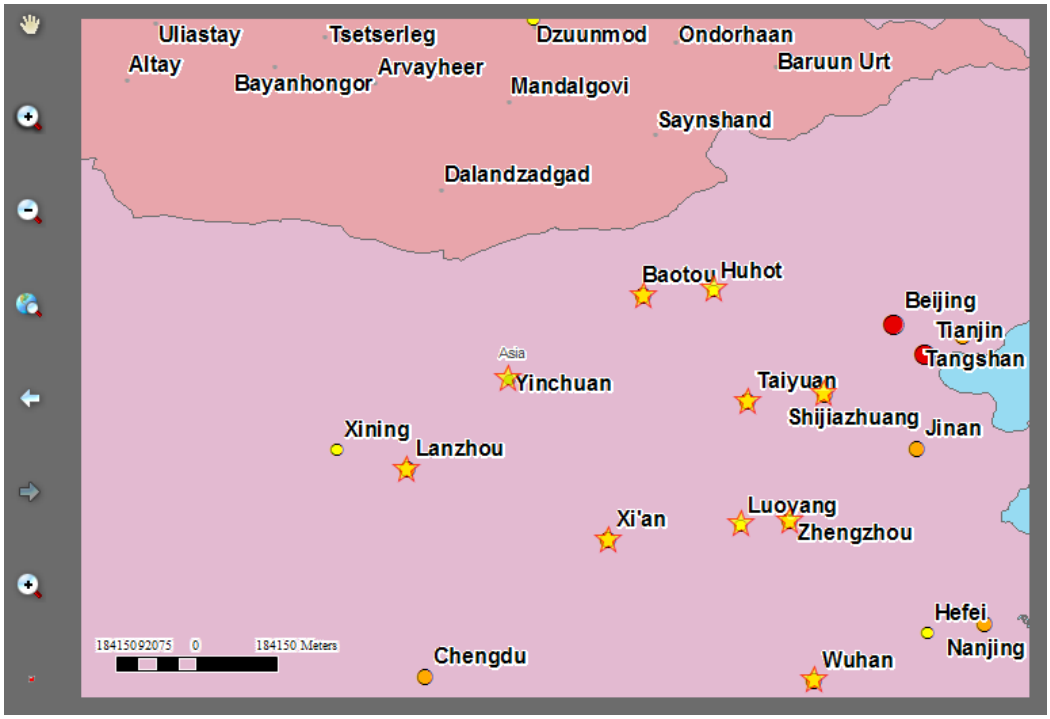


图 21　简单点符号的效果

当简单的点、线、面让你觉得枯燥，那么图片标注或许就是一个可以吸引眼球的好

主意：

```
WebPictureMarkerSymbol symbol = new WebPictureMarkerSymbol();
symbol.setFileName("images/esri-logo.gif");
```

这个图片是在 WebContent 下 images 子目录中的一个 GIF 图片，下面是这个图片用作符号的效果：



图 22　图片点符号的效果

还有就是 TrueType 的点符号 ,你应该知道 ArcGIS 带了很多 TrueType 字体格式的符号，下面我们将一个 TrueType 文字设置为符号：

```
WebTrueTypeMarkerSymbol symbol=new WebTrueTypeMarkerSymbol();
symbol.setFontName("ESRI Business");
symbol.setCharacterIndex(0);
symbol.setFontColor("0,255,0");
symbol.setFontSize(50);
symbol.setFontStyle(WebTrueTypeMarkerSymbol.UNDERLINE);
```

下面就是这个 TrueType 点符号的效果：

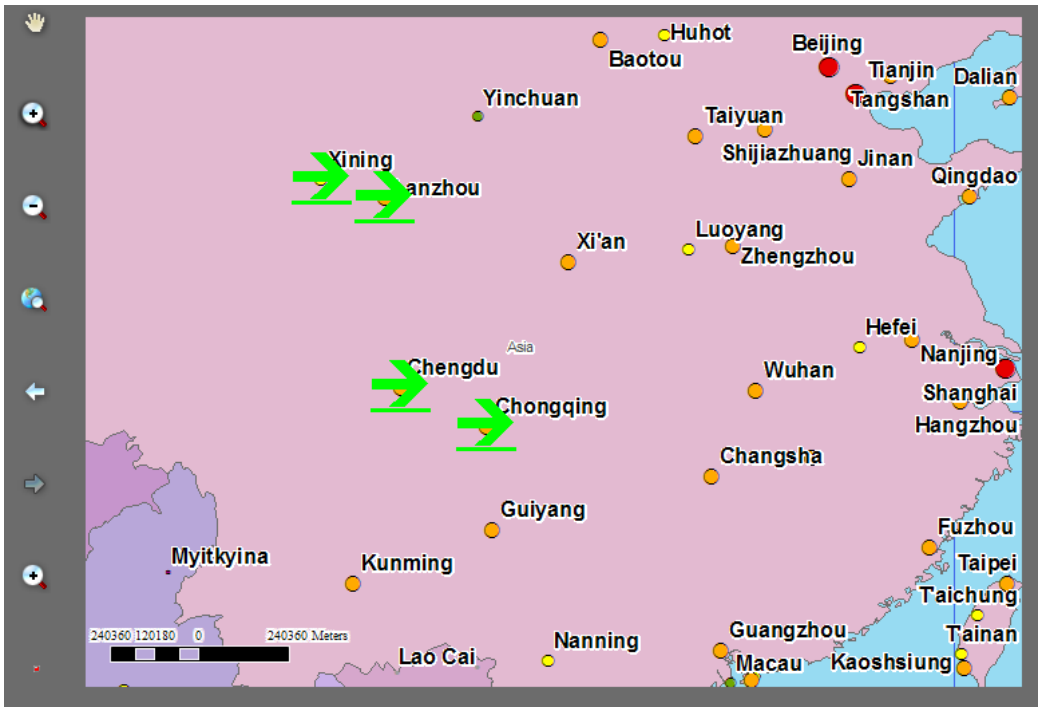


图 23　TrueType 点符号的效果

七 GraphicRenderer

上一小节我们知道怎样给 GraphicElement 指定 WebSymbol 来进行符号化，这里我们讲讲如何对 WebGraphics 进行设置，让它可以按照一定的规则自动将前面查询到的结果进行不同的渲染。

WebGraphis 有个 addRenderer 方法，它可以添加 GraphicRenderer[1]对象，这个 GraphicRenderer 对象是几何对象的渲染器，它有一个 renderer 属性用以存放真正的符号渲染接口——WebRenderer，因此，它可以遵循某些规则对几何对象的集合进行渲染。

有这几个类实现了 WebRenderer 接口：WebSimpleRenderer 、WebClassBreaksRenderer、WebUniqueValueRenderer。看到这些名称你是不是感

[1] com.esri.adf.web.data.GraphicRenderer

觉很熟悉？下面我们就通过 WebUniqueValueRenderer 来实现对查询结果中所有的 QueryResult 进行按值渲染。

```
ArrayList<WebGeometry> geometries = new ArrayList<WebGeometry>();
ArrayList<String> values = new ArrayList<String>();
for (int i = 0; i < listQueryResult.size(); i++){
    QueryResult queryResult = listQueryResult.get(i);
    geometries.add(queryResult.getHighlightGeometry());
    values.add(queryResult.getDetails().get("POP_RANK").toString());
}

WebUniqueValueRenderer wuvr = new WebUniqueValueRenderer();
wuvr.setField1("POP_RANK");
wuvr.setDefaultSymbol(symbol);

ArrayList<WebUniqueValueInfo> infos = new ArrayList<WebUniqueValueInfo>(2);
WebSimpleMarkerSymbol s1 = new WebSimpleMarkerSymbol();
s1.setColor("0,0,255");
s1.setWidth(25);
s1.setMarkerType(WebSimpleMarkerSymbol.CIRCLE);
WebUniqueValueInfo uvi = new WebUniqueValueInfo("", "", "1", s1);
infos.add(uvi);
WebSimpleMarkerSymbol s2 = new WebSimpleMarkerSymbol();
s2.setColor("0,255,0");
s2.setWidth(25);
s2.setMarkerType(WebSimpleMarkerSymbol.CIRCLE);
uvi = new WebUniqueValueInfo("", "", "2", s2);
infos.add(uvi);
wuvr.setUniqueValueInfos(infos);

GraphicRenderer graphicRenderer = new GraphicRenderer();
graphicRenderer.setRenderer(wuvr);
graphicRenderer.setGeometries(geometries);
graphicRenderer.setValues(values);

webGraphics.addRenderer(graphicRenderer);

webContext.refresh();
```

从上面的代码我们可以看到，我们首先需要做的准备工作就是遍历 QueryResult 的集合，生成几何对象的集合 geometries 和需要分值渲染的值的集合 values；然后，

我们需要一个 WebUniqueValueRenderer 对象来定义分值渲染的规则，比如某值需要渲染成某符号等等；当这些准备工作完成以后，GraphicRenderer 就登场了，前面的准备工作都会被放到 GraphicRenderer 中去，再把这个 GraphicRenderer 对象添加到 WebGraphics 后，刷新 WebContext 的同时 ADF 就会根据前面定义的规则，将这些查询结果分别渲染成不同的符号并添加到地图上。

下面是这个分值渲染的效果：

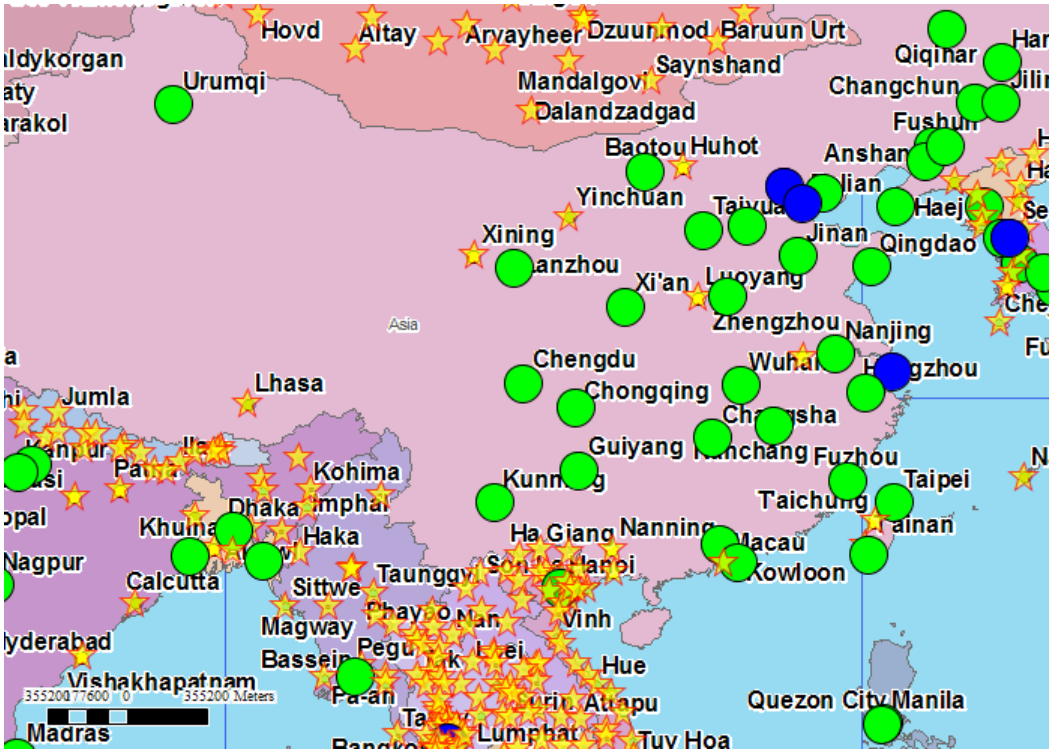


图 24　GraphicRenderer 进行分值渲染的效果

在这里，让我们顺便来看一个分类渲染的效果：

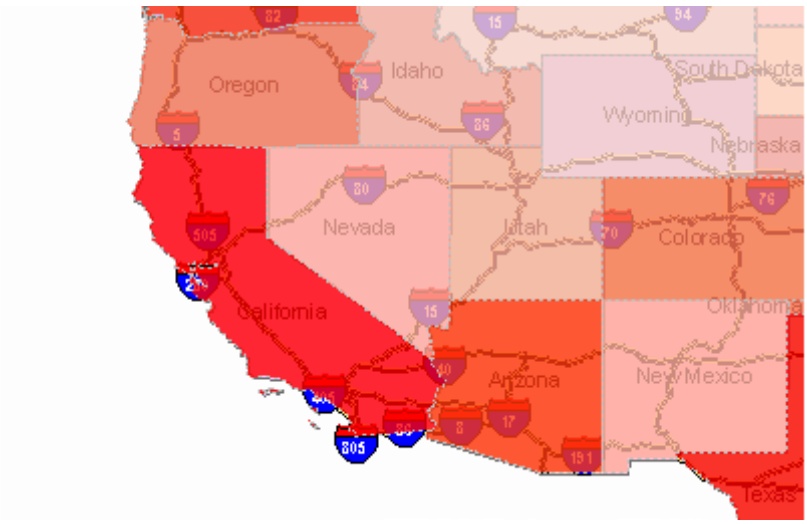


图 25　GraphicRenderer 进行分类渲染的效果

第七章 编辑工具

ADF 有很完善的编辑工具，你只需要做一点点定制就可以给你的应用加上对空间信息的编辑功能。正是因为 ADF 的编辑功能做地比较完备，因此本章的内容也会比较简单一点，主要介绍编辑工具 Editing Task 的配置和效果。

一 编辑的条件

使用 ADF 提供的 Editing Task 的前提是在当前的应用中，存在需要编辑的地图服务，这个地图服务必须使用 SDE 数据源。如果你原来的数据源有问题，你也可以很方便地在 JSP 页面上右键，然后选择【ArcGIS】-【Configure Map Service】菜单来改变你的数据源，如图 26 所示：

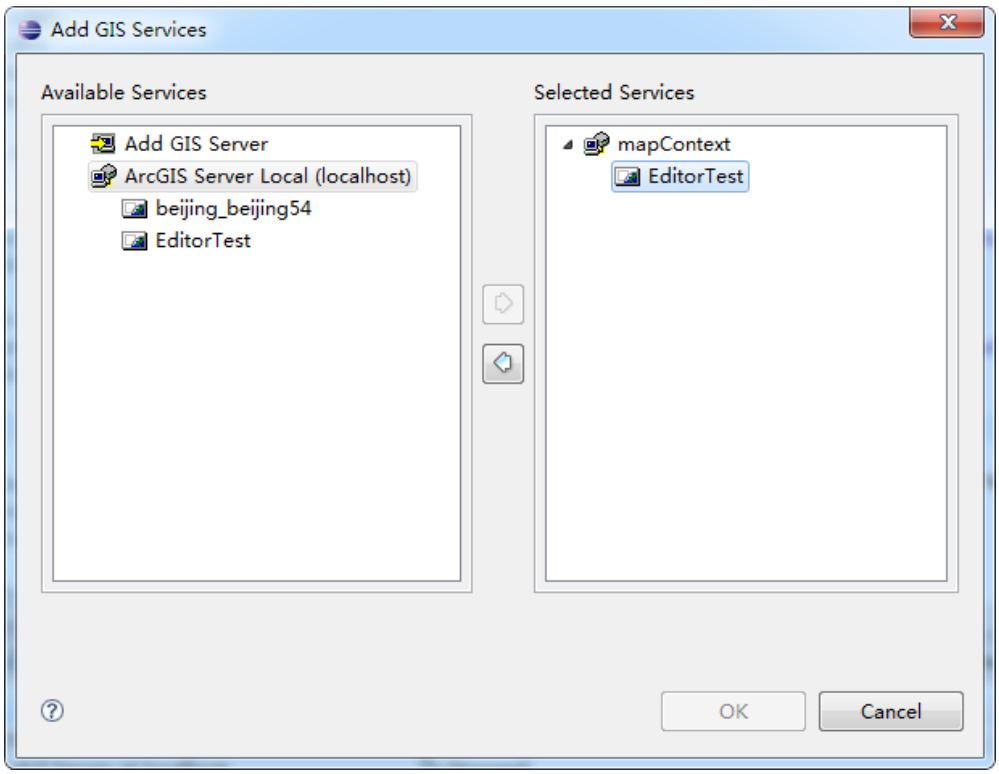


图 26 通过 IDE 工具配置应用中的服务

二 添加编辑工具

接下来，我们要做的工作自然是添加编辑工具。同样，我们通过 IDE 工具可以比较方便地配置编辑工具，方法就是在 JSP 页面上点击右键，在菜单中选择【ArcGIS】-【Add Task】-【Editing】，马上会出现一个添加 Editing Task 的对话框，如图 27。

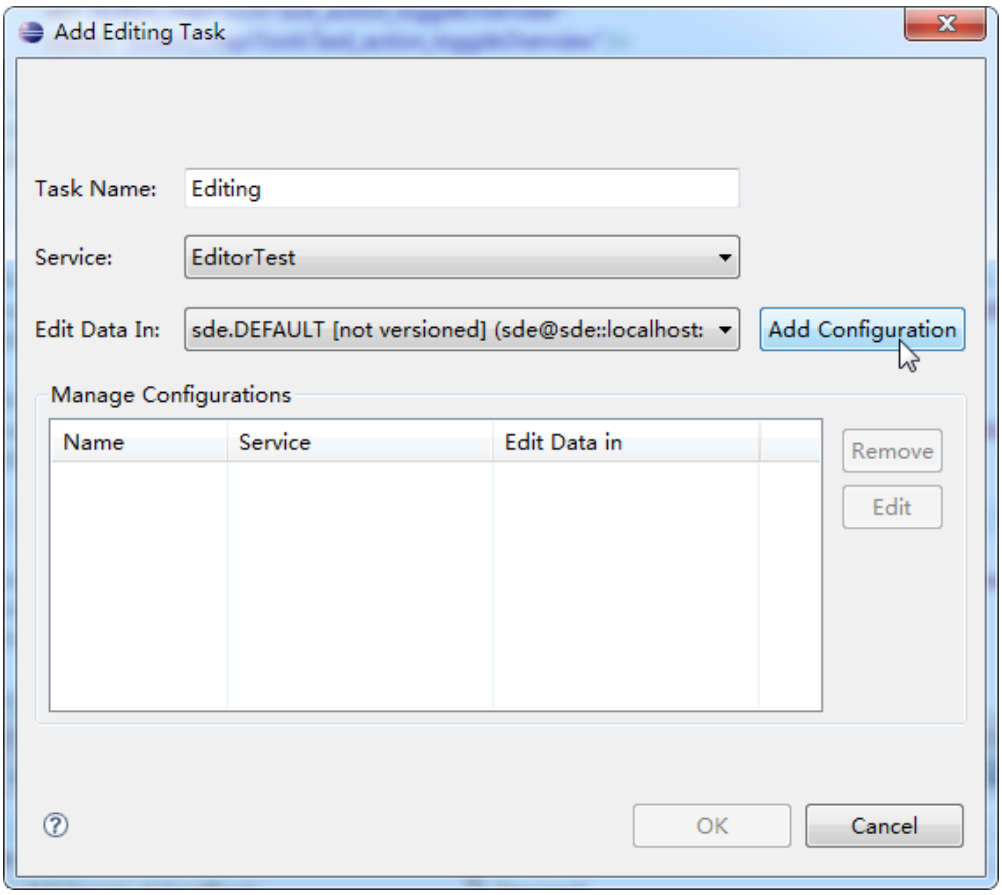


图 27　通过 IDE 工具添加 Editing Task

在这里，我们选择了在非版本的 SDE 数据源中进行编辑，这时，我们还需要点击“Add Configuration”按钮进行进一步的配置：

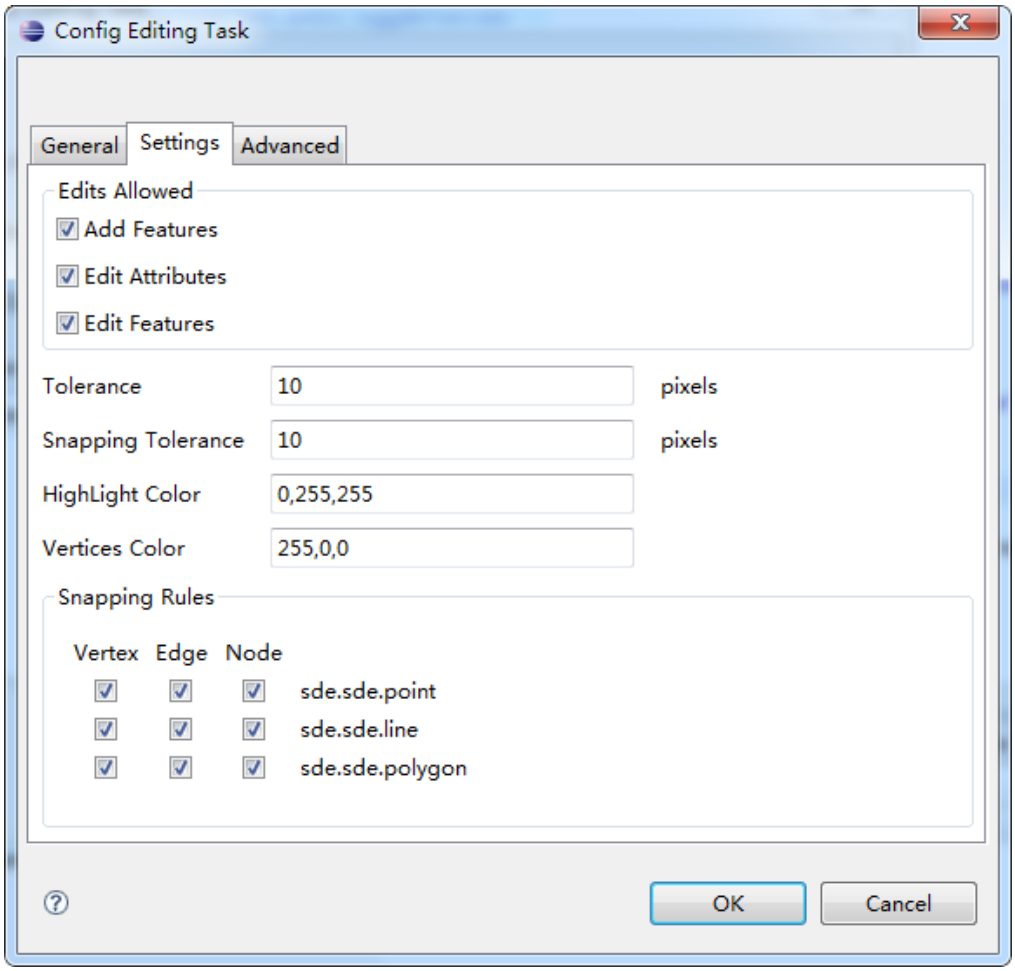


图 28　配置 Editing Task

在这里有很多选项可以编辑，比如哪些 Feature Class 你需要编辑；编辑是否包含添加、属性更改；编辑的捕捉容差等等。总而言之，这个配置页面可以帮助你从多方面去配置 Editing Task。

最后，回到“Add Editing Task”窗体，可以看到刚才新添加的配置，这时我们可以点击“OK”按钮完成添加。

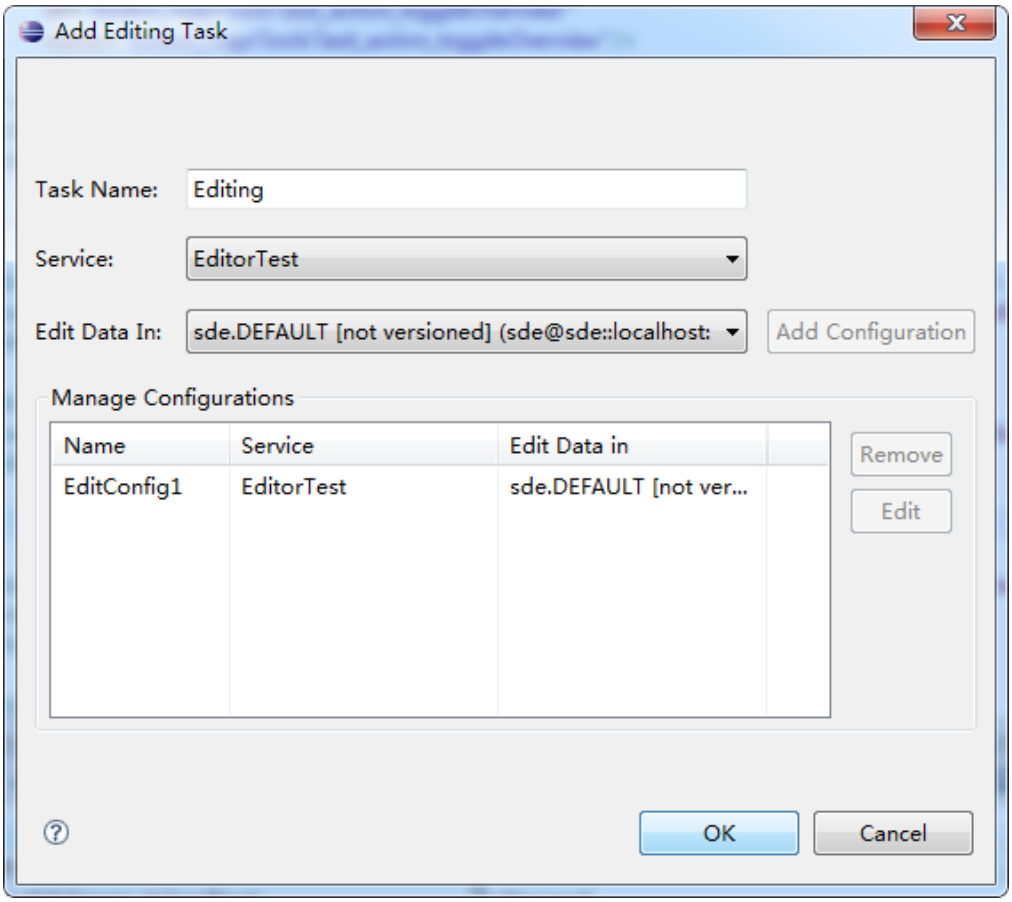


图 29　完成 Editing Task 的添加

三 使用编辑工具

下面让我们看一下 ADF 的 Editing Task 的大概界面和功能：

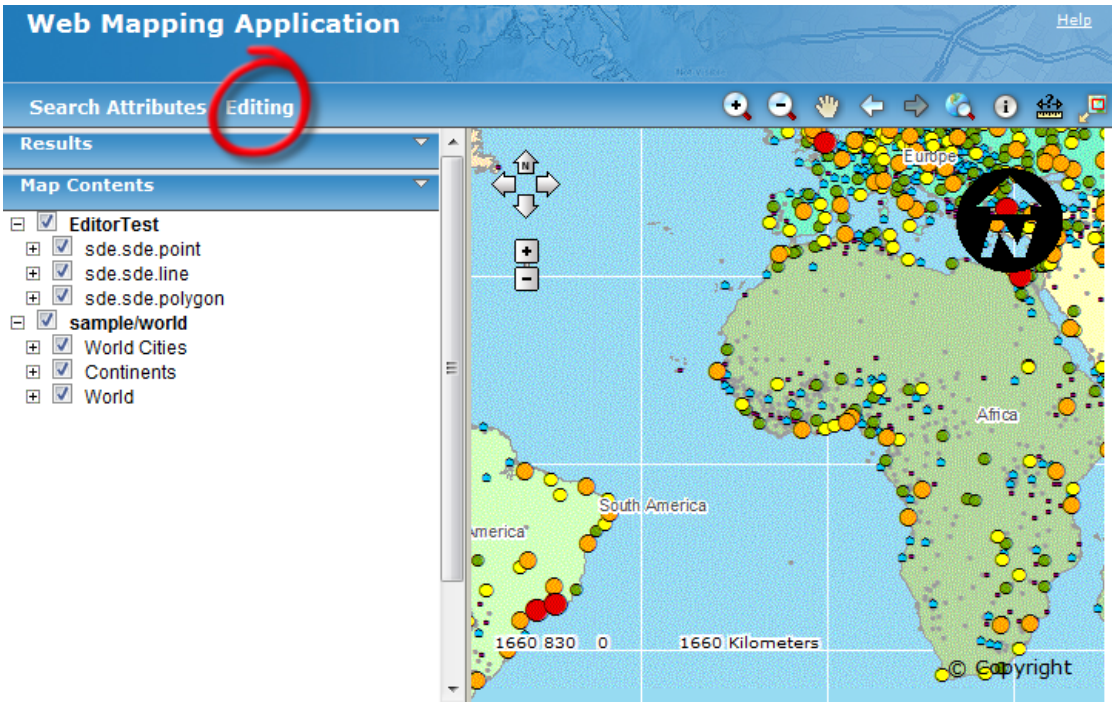


图 30　通过 Editing 菜单调用 Editing Task

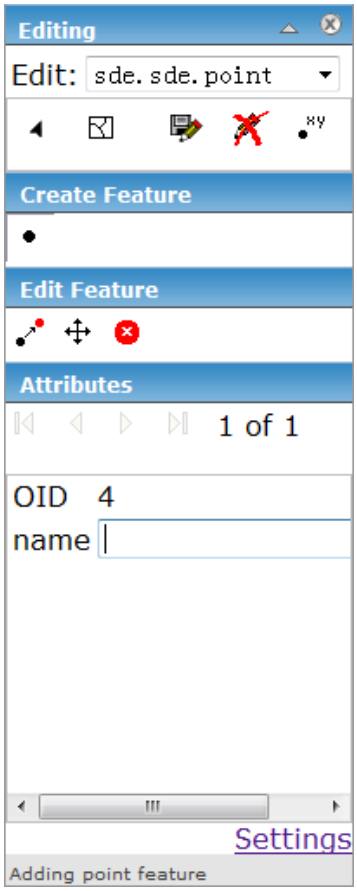


图 31　使用 Editing Task 编辑点层

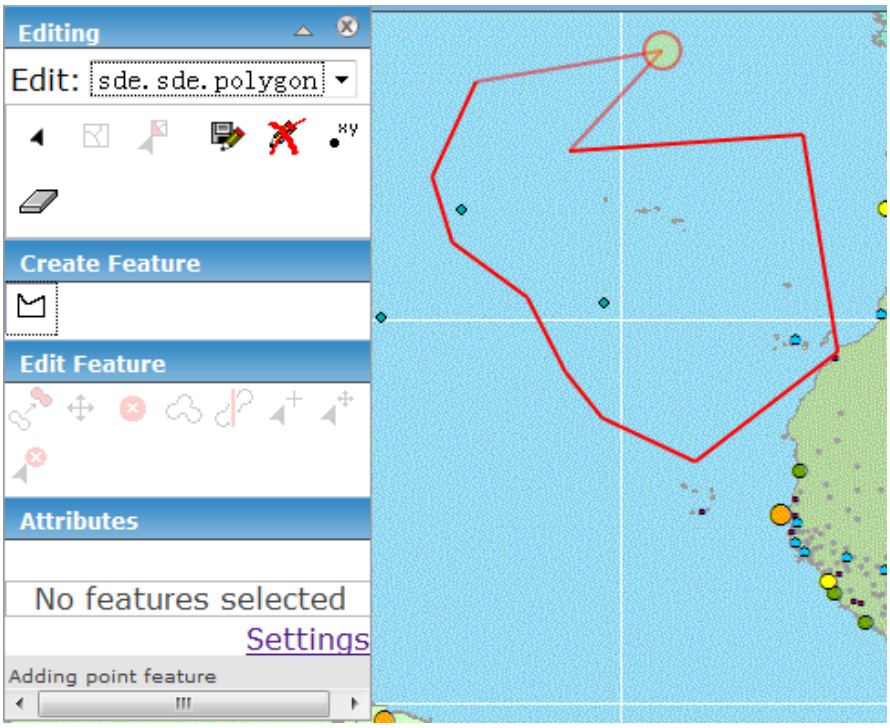


图 32　使用 Editing Task 编辑面层

更多精彩内容，请跳过本处广告，自行添加一个 Editing Task 看看。

第八章 自定义的请求和响应

前面用户的请求都是通过 Task、Toolbar 来执行的，在这一章里，我们将不使用这些 ADF 组件，来看看没有了它们的帮助，你应该怎么自己发送 AJAX 请求，或者在原先输出的 AJAX 响应中添加上一些额外的信息。

四 发送 AJAX 请求

首先让我们看一下怎么不依靠任何 ADF 组件的帮助来发送一个 AJAX 请求 ,比如，在页面上有一个纯 HTML 的表单按钮，我们希望点击这个按钮的时候可以发送一个 AJAX 请求到服务器，应该怎么做？

那就让我们假设这样一个场景：用户点击一个 HTML 的“放大 5 倍”按钮，地图就可以放大 5 倍。要实现这样的功能 ,当然首先是需要添加一个按钮 ,这个按钮的 onclick 事件对应到一个 JavaScript 函数：

```
<input type="button" value="放大 5 倍" onclick="mapZoom();" />

function mapZoom()
{
    var formId = map.formId;
    var url = EsriUtils.getServerUrl(formId);
    var params = "mapZoom=true&" + EsriUtils.buildRequestParams(formId);
    EsriUtils.sendAjaxRequest(url, params, true, EsriControls.processPostBack);
}
```

发送 AJAX 请求有两个条件，一是知道需要发送的目标 URL、还有一个就是发送的参数。我们可以从上面看到 ,AJAX 请求的 URL 可以通过 EsriUtils 这个工具类来获得，ADF 的 JavaScript 库已经做好了封装；对于 AJAX 的参数，则首先需要使用 EsriUtils 的另外一个方法 buildRequestParams 来构建基础的请求参数，在这些参数中包含了

ADF 组件的相关信息，这是 ADF 的 AJAX 请求必须的；在这个参数的基础上再加上额外的信息，比如这里需要地图作一个放大操作，那么我可以添加一个请求参数 mapZoom，后面你会看到我们在服务器端会使用到这个 mapZoom 的值。

五 监听 AJAX 请求并响应

当从页面使用 JavaScript 发送了一个 AJAX 请求到服务器，服务器就需要对这个请求进行处理，因此这一小节的内容主要就是在服务器端。首先，让我们来了解一些 JSF 的内容——有关 PhaseListener[1]。

所谓的 Phase 就是 JSF 请求的阶段，一个 JSF 请求的生命周期中存在不同的阶段，当用户从页面向服务器发送一个请求的时候，JSF 请求的生命周期就开始了。以下就是著名的 JSF 请求生命周期中 6 个阶段：

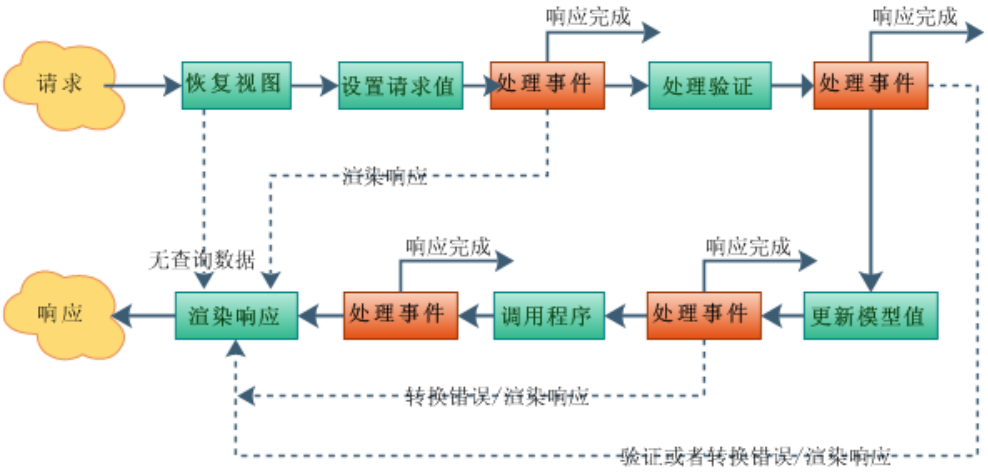


图 33　JSF 请求的生命周期

PhaseListener 是对 JSF 请求的阶段进行监听的接口，当某个类实现了这个接口并添加到 JSF 应用中以后，每当服务器接收到 JSF 请求，并处理这个请求到某个阶段的时

[1] javax.faces.event.PhaseListener

候，这个监听类就会被触发。

下面是实现的一个监听自定义 AJAX 请求的监听类：

```
public class MapUpdatePhaseListener implements PhaseListener{
    public void afterPhase(PhaseEvent event){
        FacesContext facesContext = event.getFacesContext();
        Map params = facesContext.getExternalContext().getRequestParameterMap();
        WebContext webContext = WebUtil.getWebContext(facesContext.getViewRoot());
        WebMap webMap = webContext.getWebMap();

        if("true".equals(params.get("mapZoom"))){
            UIComponent form = facesContext.getViewRoot().findComponent("formMain");
            Vector<String> eventSources = new Vector<String>();
            try{
                AJAXResponseRenderer renderer = new
                        AJAXResponseRenderer(facesContext, eventSources, form);
                WebExtent extent = webMap.getCurrentExtent();
                extent.expand(0.2);
                webContext.refresh();
                Document doc = renderer.renderResponse(facesContext);
                AJAXUtil.writeResponse(facesContext, doc);
            }catch (Exception e) {
                 e.printStackTrace();
            }finally{
                 facesContext.responseComplete();
            }
        }
    }

    public void beforePhase(PhaseEvent event) { }

    public PhaseId getPhaseId() {
        return PhaseId.APPLY_REQUEST_VALUES;
    }
}
```

通过实现 getPhaseId 方法，我们指定了这个类监听的是 JSF 请求的应用请求值阶段；在这个阶段中，我们截获请求，并判断请求的参数。上一小节我们在请求的参数中加上了 mapZoom 这个属性，这里我们就要判断请求中有没有这个属性，如果这个属性值为 true，那么就表示当前请求正是需要服务器处理“地图放大 5 倍”这个功能的自

定义请求。

在对这个请求的处理中，我们可以通过对 WebContext 对象的操作实现地图放大 5 倍的功能，但是在这里你会发现，实现这个功能以后并不像以前那样直接调用 WebContext 的 refresh 方法就可以展示给浏览器了，这是为什么呢？

关于 WebContext 的 refresh 方法为何可以将结果输出到浏览器，你可以回顾一下 44 页的《WebContext 的 refresh 方法》。由于这里我们的请求并没有通过 ADF 组件来发送，因此 PostBackPhaseListener 不会被触发（而且其中也没有我们需要服务器实现的业务逻辑），这里的 MapUpdatePhaseListener 就需要自己输出 AJAX 响应。

具体如何构造 AJAX 响应，并通过 AJAXUtil 的 writeResponse 方法输出到浏览器，看一下上面的代码就清楚了。这里构造 AJAX 响应的过程显得比较复杂，这是因为我们在这里实现了对地图进行了更新（而不是通过 Tool 或 Command），所以也需要我们自己去通知客户端当前服务器端 Map 等组件的状态以便其刷新（如果你不需要通知客户端关于 ADF 组件的相关信息，比如只想通过 AJAX 获得某些信息，那就会比较简单，在下一小节中有这样的例子）。顺便提醒一下，自己构造 AJAX 响应的时候一定要注意实例化 AJAXResponseRenderer 对象、刷新 WebContext、输出 AJAX 响应这几个环节的先后顺序。

还有一点，需要使用这个 PhaseListener，需要把这个类注册到 faces-config.xml 文件中去，如下所示：

```
<lifecycle>
    <phase-listener>wuyf.MapUpdatePhaseListener</phase-listener>
</lifecycle>
```

当然，还有一种比较简便的办法就是在新建这个 PhaseListener 类的时候使用

eclipse 的菜单来定义，这比手动去改代码和配置文件要方便多了：

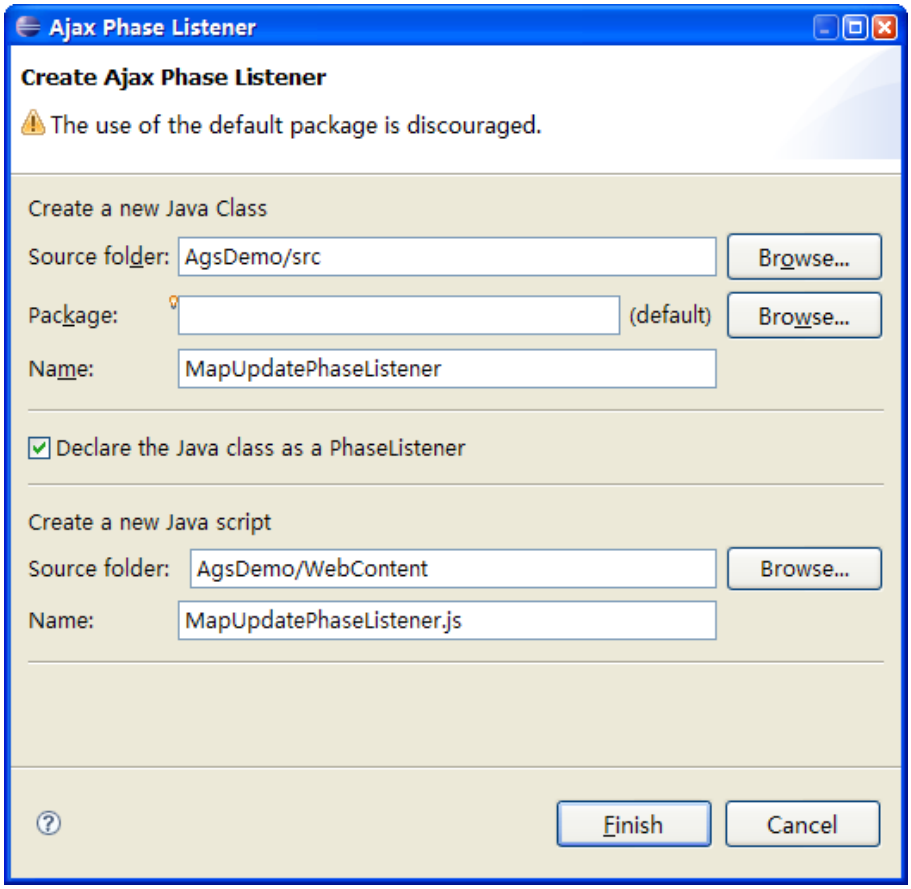


图 34　使用 eclipse 的代码模板来新建 PhaseListener

最后，在这个小节结束之前，我想再说明一下上面示例的那个例子。这个例子假设的功能是“地图放大 5 倍”，因此它需要更新服务器上的 ADF 组件的状态，所以你可以看到有对 WebContext 的刷新操作。相对的，另外还有一种场景有可能不需要更新服务器上的 ADF 组件状态，比如获取地图比例等等，那这个时候构造自定义的响应将会有所不同。我们下一个小节就是一个这样的例子，希望你可以关注一下这个例子和下一小节例子之间的区别，重点注意如何构造响应的 XML 对象的不同。

六 自定义 AJAX 请求获得地图范围【案例】

比如你现在想随时知道当前地图的边界范围，也就是说每次地图范围更新后，希望

服务器可以输出一个包含边界范围的 AJAX 响应到浏览器，那么首先你需要在用户对地图进行操作的时候发送一个 AJAX 请求。这个请求你可以在每次地图被更新的时候去调用，比如一开始就通过 addUpdateListener 给 Map 添加一个监听：

```
<body onload="onBodyLoad();">
</body>

function onBodyLoad()
{
    map = EsriControls.maps['map'];
    map.addUpdateListener("onMapUpdate", onMapUpdate);
}

function onMapUpdate()
{
    var formId = map.formId;
    var url = EsriUtils.getServerUrl(formId);
    var params = "getMapInfo=true&" + EsriUtils.buildRequestParams(formId);
    EsriUtils.sendAjaxRequest(url, params, true, processMapInfo);
}
```

这个时候，服务器就接收到了用户发出的参数中带有 getMapInfo 属性的 AJAX 请求，我们将通过这个属性来判断是不是用户的自定义请求，需要服务器额外返回当前的地图范围给浏览器。

下面是服务器端的处理，主要是实现一个 PhaseListener 对请求进行监听并过滤：

```
public class MapUpdatePhaseListener implements PhaseListener{
    public void afterPhase(PhaseEvent event) {
        FacesContext facesContext = event.getFacesContext();
        Map params = facesContext.getExternalContext().getRequestParameterMap();
        WebContext webContext = WebUtil.getWebContext(facesContext.getViewRoot());
        WebMap webMap = webContext.getWebMap();

        if ("true".equals(params.get("getMapInfo"))){
            WebExtent extent = webMap.getCurrentExtent();
            double xmax = extent.getMaxX();
            double ymax = extent.getMaxY();
            double xmin = extent.getMinX();
            double ymin = extent.getMinY();
```

```
                Document doc = XMLUtil.newDocument();
                Element mapInfo = XMLUtil.createElement(doc, "mapInfo", "", null);
                XMLUtil.createElement(doc, "xmax", String.valueOf(xmax), mapInfo);
                XMLUtil.createElement(doc, "xmin", String.valueOf(xmin), mapInfo);
                XMLUtil.createElement(doc, "ymax", String.valueOf(ymax), mapInfo);
                XMLUtil.createElement(doc, "ymin", String.valueOf(ymin), mapInfo);
                try{
                    AJAXUtil.writeResponse(facesContext, doc);
                }catch (IOException e) {
                    e.printStackTrace();
                }finally {
                    facesContext.responseComplete();
                }
            }
        }

        public void beforePhase(PhaseEvent event) { }

        public PhaseId getPhaseId() {
            return PhaseId.APPLY_REQUEST_VALUES;
        }
}
```

从这个 PhaseListener 中 ,我们过滤出了请求中含有 getMapInfo 属性的 AJAX 请求，如果这个属性值为 true，那么表示需要输出包含地图范围的 AJAX 响应，此时，通过构造一个 XML 的 Document，并加上需要输出的 xmin、xmax、ymin、ymax 等值后，通过 AJAXUtil 的 writeResponse 方法来输出到客户端。

下面让我们回到客户端 ,看看发送 AJAX 请求时设置的回调函数 processMapInfo：

```
function processMapInfo(xh)
{
    if (xh != null && xh.readyState == 4 && xh.status == 200)
    {
        var xml = EsriUtils.getXmlDocument(xh);
        var xmax = EsriUtils.getXmlText(xml.getElementsByTagName("xmax").item(0));
        var xmin = EsriUtils.getXmlText(xml.getElementsByTagName("xmin").item(0));
        var ymax = EsriUtils.getXmlText(xml.getElementsByTagName("ymax").item(0));
        var ymin = EsriUtils.getXmlText(xml.getElementsByTagName("ymin").item(0));

        document.getElementById("xmax").innerHTML = xmax;
```

```
            document.getElementById("xmin").innerHTML = xmin;
            document.getElementById("ymax").innerHTML = ymax;
            document.getElementById("ymin").innerHTML = ymin;
        }
}
```

这个函数的参数 xh 就是 AJAX 请求（XMLHttpRequest），通过它我们可以获得 AJAX 响应的具体内容。当获得响应的内容以后，你就可以继续进行任意的处理了，比如上面的代码就是通过几个 div 把地图范围的坐标给显示出来。

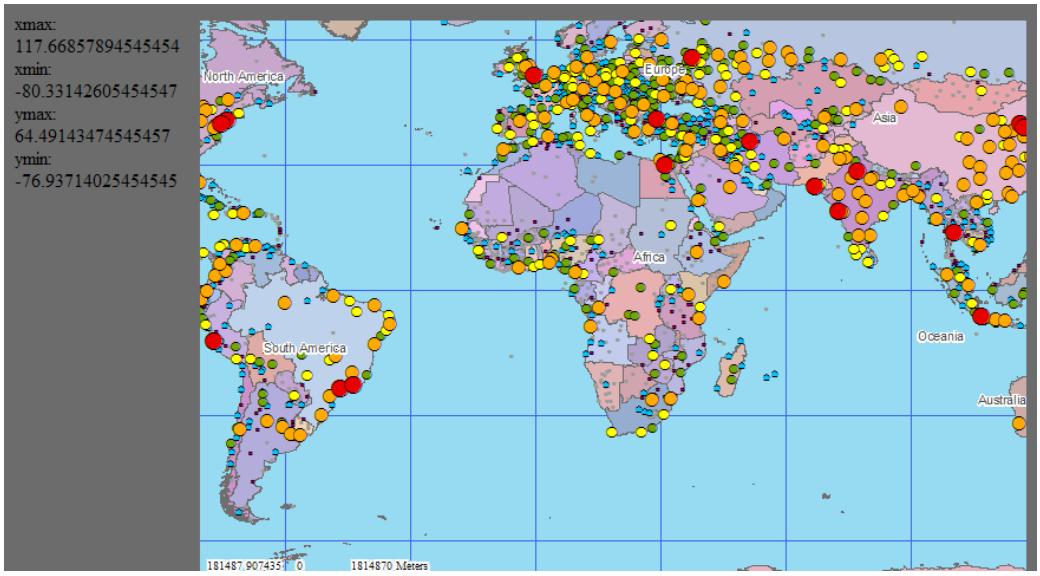


图 35　自定义 AJAX 请求获得地图范围

七 通过 AJAXRenderer 进行交互

前面涉及的都是自己在客户端发 AJAX 请求，在服务器端通过 PhaseListener 捕捉请求参数，然后进行请求的分发和处理并生成 XML 响应，最后在客户端接受响应进行解析。这一节将要看到的是 Task 处理的结果如何以 XML 响应的形式返回给客户端。与前面的例子不同之处在于，用 Task 的按钮提交的请求，客户端的 AJAX 不是自己构建的，服务器端也不是用 PhaseListener 进行业务处理的。

我这里创建了一个 GraphicTask，其中有个工具是矩形框查询并把查询结果进行高亮，

这是一个很简单的例子，这里我们要对它进行扩展，不仅要把这些要素高亮显示，而且要在响应流中添加一些自定义的内容。实现步骤如下：

首先打开 GraphicTask 的定义，在进行选择操作的 selectByRect(MapEvent event) 方法最后增加以下代码，用以将查询结果数暂时保存在 WebSession 中：

```
int queryResultNumber = 0;
if(rs != null){
 queryResultNumber = rs.size();
}
wctx.getWebSession().setAttribute("queryResultNumber", queryResultNumber);
```

然后再增加一个 Java 类：CustomRenderer，这个类实现了 AJAXRenderer[1]接口：

```
import javax.faces.component.UIComponent;
import javax.faces.context.FacesContext;
import org.w3c.dom.Element;
import com.esri.adf.web.faces.component.MapControl;
import com.esri.adf.web.faces.renderkit.xml.ajax.AJAXRenderer;

public class CustomRenderer implements AJAXRenderer {
 public Class getControlClass() {
       return MapControl.class;
 }

 public Object getOriginalState(UIComponent component) {
       return null;
 }

 public void renderAjaxResponse(FacesContext fctx, UIComponent component,
             Object state, boolean isEventSource, Element parent) {
 }
}
```

在 CustomRenderer 的 renderAjaxResponse 方法中添加以下代码，用以从 WebSession 中读取 GraphicTask 执行的结果，然后写到响应流中：

```
WebContext wctx = WebUtil.getWebContext(component);
Object obj = wctx.getWebSession().getAttribute("queryResultNumber");
```

[1] com.esri.adf.web.faces.renderkit.xml.ajax.AJAXRenderer

```
if(obj != null){
 Element myElement = XMLUtil.createElement("queryResultNumber","",parent);
 Element number = XMLUtil.createElement("number", "", myElement);
 number.setAttribute("count", obj.toString());

 wctx.getWebSession().setAttribute("queryResultNumber",null);
}
```

最后，打开 WEB-INF/ajax-renderers.xml，为 CustomRenderer 添加一个 managed-bean 的声明，并注册到 AJAXRenderers：

```
<managed-bean>
    <display-name>AJAXRenderers</display-name>
    <managed-bean-name>AJAXRenderers</managed-bean-name>
    <managed-bean-class>java.util.ArrayList</managed-bean-class>
    <managed-bean-scope>request</managed-bean-scope>
    <list-entries>
      ..................
      <value>#{customRenderer}</value>
    </list-entries>
</managed-bean>

<managed-bean>
      <managed-bean-name>customRenderer</managed-bean-name>
      <managed-bean-class>CustomRenderer</managed-bean-class>
      <managed-bean-scope>none</managed-bean-scope>
</managed-bean>
```

下面的工作集中在客户端，首先，我们需要在页面的 onload 函数中增加一个处理 XML 响应的回调函数：

```
function init(){
 closeTaskWindows();
 EsriControls.addPostBackTagHandler("queryResultNumber",queryResultNumberHandler);
}
```

这里的 queryResultNumberHandler 用以处理服务器端返回 XML 代码段，从中提取查询结果数并弹出一个对话框提示：

```
function queryResultNumberHandler(xml){
 var numberTag = xml.getElementsByTagName("number").item(0);
 alert("矩形框查询的结果一共有<" + numberTag.getAttribute("count") + ">个");
```

```
}
```

运行以后的结果如图：

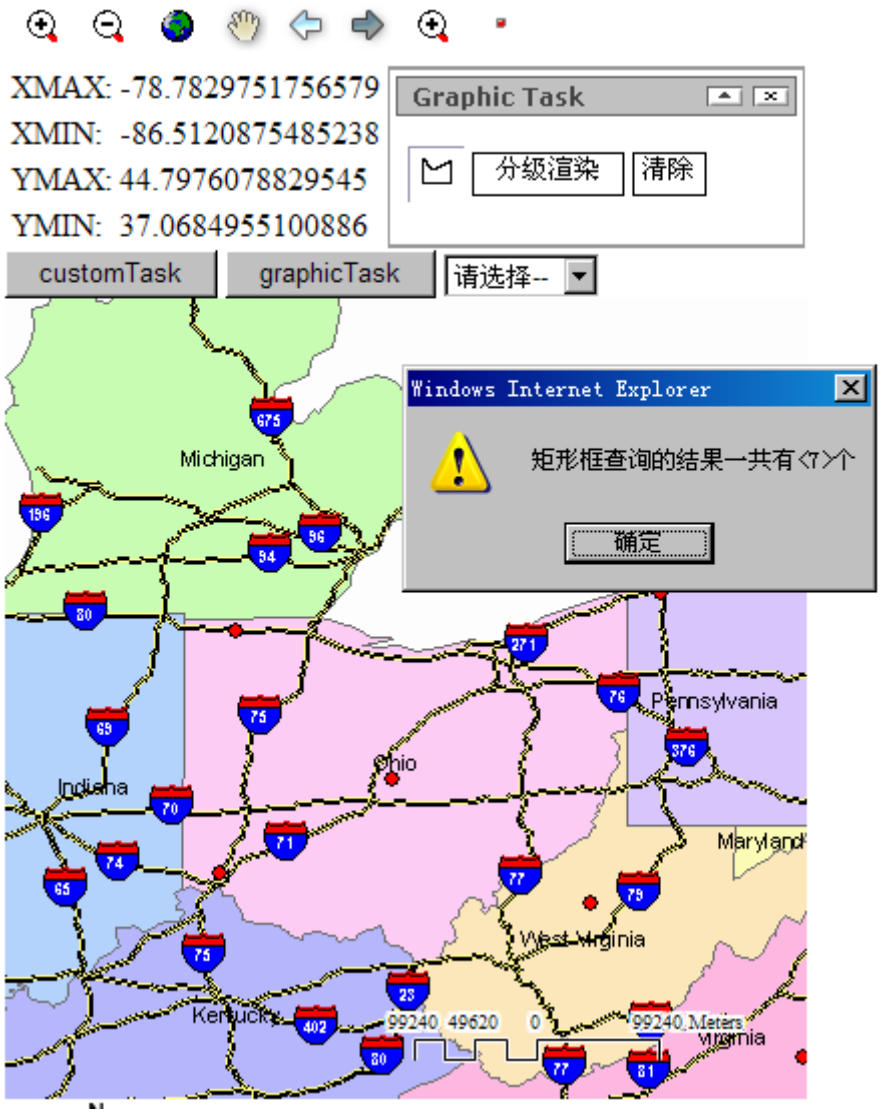


图 36　通过 AJAXRenderer 把查询结果返回给浏览器

第九章 自定义 Functionality

Functionality 在 ADF 中的全称是 GISFunctionality，从 ADF 的逻辑结构中我们知道，每个 GISResource 会对应多个 GISFunctionality，由这些 GISFunctionality 来提供实际的功能。这一章我们主要来看如何定义自己的 Functionality 来扩展资源的功能。

一 GISFunctionality 详解

既然 Functionality 与 Resource 关系紧密，那么就让我们从 Resource 入手，来看 GISFunctionality 是如何工作的。

下面是我们在“faces-config.xml”中配置的一个地图服务：

```
<managed-bean>
 <managed-bean-name>agswsWorld</managed-bean-name>
 <managed-bean-class>com.esri.adf.web.ags.data.AGSMapResource</managed-bean-class
>
 <managed-bean-scope>none</managed-bean-scope>
 <managed-property>
      <property-name>endPointURL</property-name>
      <value>http://localhost:8399/arcgis/services/AgsSample/World/MapServer</value>
 </managed-property>
 <managed-property>
      <property-name>alias</property-name>
      <value>AgsSample/World</value>
 </managed-property>
 <managed-property>
      <property-name>functionalities</property-name>
      <map-entries>
           <map-entry>
                <key>map</key>
                <value>#{agsMap}</value>
           </map-entry>
           <map-entry>
                <key>query</key>
```

```
            <value>#{agsQuery}</value>
        </map-entry>
        <map-entry>
            <key>tile</key>
            <value>#{agsTile}</value>
        </map-entry>
        <map-entry>
            <key>overview</key>
            <value>#{agsOverview}</value>
        </map-entry>
        <map-entry>
            <key>toc</key>
            <value>#{agsToc}</value>
        </map-entry>
    </map-entries>
</managed-property>
</managed-bean>
```

很显然，这是一个通过 Internet 方式去访问的本机上的一个名为 World 的地图服务资源，类型是 AGSMapResource。从上面可以看到，这个 Managed Bean 有一个名为 functionalities 的属性 其中存放了很多比如名为 agsMap 等的 Managed Bean，这些都是 ADF 自带的 Functianality：比如 agsMap，你打开“/WebContent/WEB-INF/functionalities/ags-functionalities.xml”就可以看到，这是一个 AGSMapFunctionality 对象。

AGSMapFunctionality 实现了 GISFunctionality 接口，下面让我们看一下 GISFunctionality 接口的定义：

```
package com.esri.adf.web.data;

public abstract interface GISFunctionality{
  public abstract void initFunctionality(GISResource paramGISResource);
  public abstract void destroyFunctionality();
  public abstract GISResource getResource();
}
```

在 GISResource 被初始化的时候，ADF 会遍历其所有的 GISFunctionality 对象并

调用其 initFunctionality 方法，这样这个 Functionality 就可以被使用了。从这之后，你不但可以从 GISResource 获得 GISFunctionality ,也可以从 GISFuctionality 获得其 GISResource。当然，这些都是后台 ADF 自动完成的工作，如果不感兴趣你也完全可以不用去关心这些细节。

二 自定义一个地图放大 5 倍的 Functionality【案例】

在这个小节 ,我们去实现一个 GISFunctionality 去封装前面实现过的使地图放大 5 倍的功能。

首先，当然是新建一个 Functionality 类并添加到 ADF 应用中去。前面我们经常通过直接编辑代码的方式去定义类和配置文件，这主要是为了帮助你理解 ADF 应用的配置和各个元素的相互关系。现在估计你也有一定了解了，我们也就尽量省事些，多使用 ArcGIS 的 eclipse 插件去完成这些工作吧。

让我们打开 eclipse 的菜单【ArcGIS】-【Class Templates】-【Functionality】，在新建的窗口上填上我们希望实现的 Functionality 的包名、类名、实例化 Managed Bean 的名称、注册到的 Resource 等，完成以后 ADF 会自动新建一个实现 GISFunctionality 接口的类，并把这个类实例化后注册到 ADF 应用的配置文件中去。

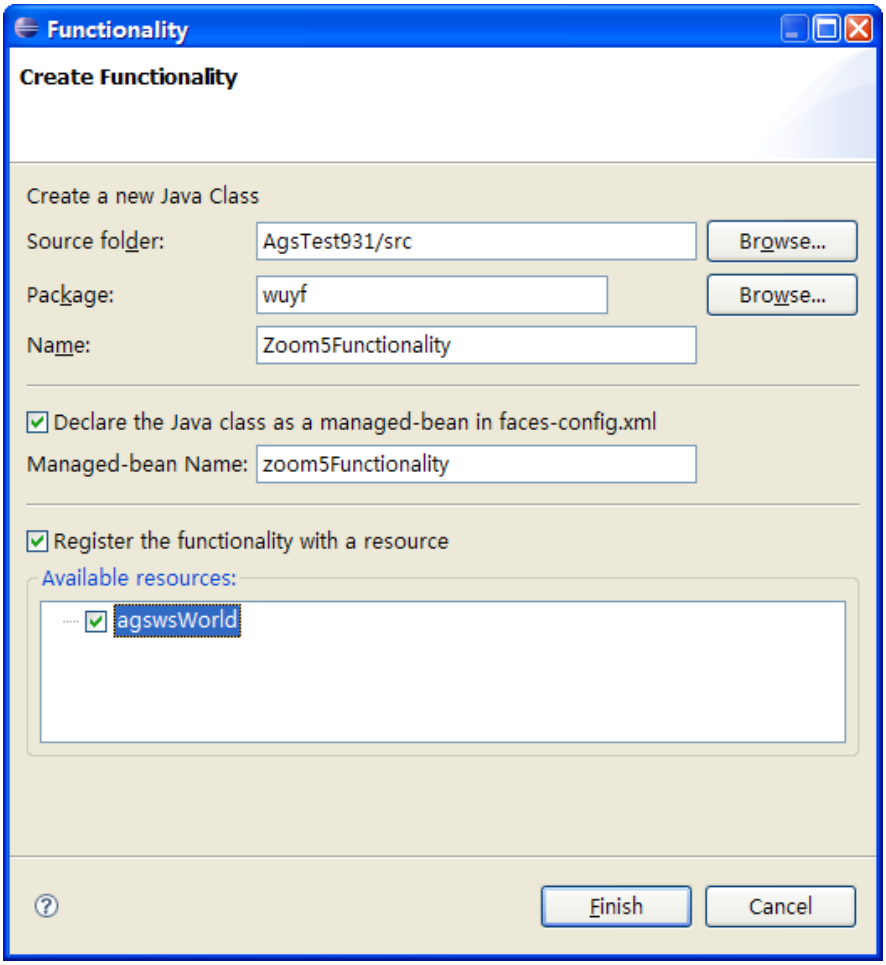


图 37　通过 ArcGIS 的 IDE 插件添加自定义 Functionality

下面我们需要对新生成的这个 Functionality 类进行编码以实现自己的功能：

```
public class Zoom5Functionality implements GISFunctionality{
 private GISResource resource;

 public void zoomIn5(){
       WebContext webContext = resource.getWebContext();
       webContext.getWebMap().getCurrentExtent().expand(0.2);
       webContext.refresh();
 }

 public void zoomOut5(){
       WebContext webContext = resource.getWebContext();
       webContext.getWebMap().getCurrentExtent().expand(5);
       webContext.refresh();
 }

 public void initFunctionality(GISResource resource){this.resource = resource; }
 public void destroyFunctionality(){}
```

```
 public GISResource getResource(){return this.resource;}
}
```

这样，这个 Zoom5Functionality 就具有了两个功能：zoomIn5 和 zoomOut5，分别对应了放大地图 5 倍和缩小地图 5 倍。接下来就让我们在页面上放一个 Command 去调用 Zoom5Functionality 的 zoomIn5 功能：

```
<a:command id="zoomIn"
            defaultImage="images/tasks/maptools/zoomin.png"
            hoverImage="images/tasks/maptools/zoominU.png"
            selectedImage="images/tasks/maptools/zoominD.png"
            clientPostBack="true" >
    <f:actionListener type="wuyf.ZoomInListener"/>
</a:command>
```

当然，服务器上还需要一个监听器去处理这个 Command 的请求，并实际去调用 Zoom5Functionality 的方法：

```
public class ZoomInListener implements ActionListener{
    public void processAction(ActionEvent event) throws AbortProcessingException {
        WebContext webContext = WebUtil.getWebContext(event.getComponent());
        GISResource resource = webContext.getResourceById("agswsWorld");
        Zoom5Functionality functionality =
            (Zoom5Functionality)resource.getFunctionality("zoom5Functionality");
        functionality.zoomIn5();
    }
}
```

或许你会觉得这样做有点多此一举，简单的功能干嘛要做一个 Functionality 来封装一下呢——当然了，这主要是因为这里演示的这个功能实在是太简单了。如果你有复杂的逻辑需要封装，而且还想灵活地配置某些资源需要这个功能，某些资源则不需要，那么 Functionality 就可以帮你大忙了。

这里还需要特别提一下的就是 GISFunctionality 的 initFunctionality 方法，这个方法在 Functionality 被初始化的时候调用，这个时候其实还可以对 GISResource 作一些操作，比如根据权限验证来选择需要显示的图层等等。

第十章 调用 ArcObjects

到这里我们对 ADF 已经讲了不少内容了，让我们先稍微回忆一下。我们从 ADF 的结构入手，首先介绍了 Tool、Command 和服务器的交互、在此基础上介绍了可以帮助实现复杂业务逻辑的 Task；然后，进行了更细节的功能讲解，比如怎么进行查询、怎么显示动态结果、怎么不通过 ADF 组件来发送 AJAX 请求并获得响应、怎么自定义 Functionality 封装功能等内容。

前面这些内容仅仅围绕着 ADF 封装的 GIS 功能，或许你会开始怀念 AO 强大的功能，所以这一章就让我们来了解一下怎样可以用 AO 来进行更加底层的 GIS 功能实现。

一 Internet 和 Local 方式连接服务的区别

首先让我们去了解一下在 ADF 应用中，连接服务的两种方式：Internet 和 Local 的区别。

Internet 方式通过一个 URL 来连接服务，它通过 HTTP 协议去访问 ArcGIS Server 的 Web Service，这些 Web Service 会通过 DCOM 访问 Server Object 从而实现 GIS 的功能，这些都是 ArcGIS Server 封装好的，在使用的时候只需要知道服务的 URL 就可以了。由于这种方式是通过 HTTP 来访问的，因此这些连接是无状态的。

Local 方式通过主机、用户名、密码的认证来连接位于本地网络中的服务，它直接通过 DCOM 协议访问 Server Object 而不需要经过其它中间环节，因此它可以直接使用 ArcObjects API 来操作服务器的 GIS 资源。

要在 ADF 应用中调用 AO，前提条件就是必须使用 Local 方式去连接服务。

二 IServerContext 详解

在 ADF 中使用 Local 方式去连接服务产生的资源是 AGSLocalMapResource[1]，它有一个方法是 getServerContext，返回的是一个实现了 IServerContext[2]接口的对象。这个接口在 ADF 调用 AO 的过程中很重要，下面我们来详细谈一下这个接口。

当我们的 ADF 应用使用 Local 的方式去连接服务，那么在发生请求并连接到 ArcGIS Server 的时候，在服务器上就会产生一个服务器环境对象（server context），这个对象可以被理解为这个 Local 连接的服务器环境，ADF 会在这个连接中处理所有的请求，因此处理请求时产生和使用的所有 AO 对象都被存放在这个服务器环境中。IServerContext 就是访问这个服务器环境的接口。

IServerContext 最重要的方法就是 createObject，它的作用就是通过 CLSID 来在服务器环境中新建一个 AO 对象。或许你有这样的疑问，为什么不能直接 new 一个对象呢？如果你 new 了一个对象，这个对象是在 ADF 应用中的；而 AO 对象实际是在 ArcGIS Server 的服务器环境中，在 ADF 中使用的只是这个对象的代理，因此不能 new 出这样的对象，而应该通过 IServerContext 来创建、获得 AO 对象。

下面的代码在 ADF 中新建了一个 Point 类型的 AO 对象：

```
Object point = serverContext.createObject(com.esri.arcgis.geometry.Point.getClsid());
```

另外，在 ADF 中还有一个 AGSLocalUtil 工具类可以将 AGSLocalMapResource 中获得的 SOAP 对象转化为 AO 对象，在下一小节中我们会看到这个工具类的用法。

---

[1] com.esri.adf.web.ags.data.AGSLocalMapResource

[2] com.esri.arcgis.server.IServerContext

三 调用 AO 实现缓冲功能【案例】

在这个小节中，让我们通过调用 AO 实现一个缓冲区分析的功能。

首先，我们需要配置一个地图服务资源，还是用前面使用过的世界地图服务吧，但是这次采用的是 Local 方式去连接，你可以直接在打开的 JSP 页面上右键选择【ArcGIS】-【Configure Map Service...】菜单打开服务配置窗口：

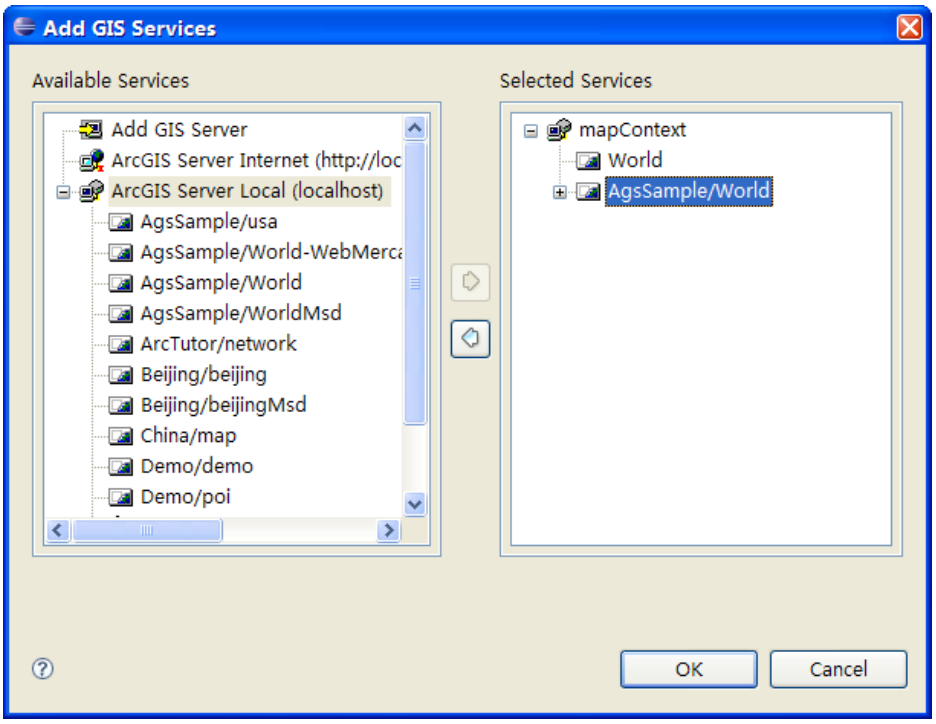


图 38　使用 IDE 插件配置 Local 方式连接的服务

下面我们在一个 Tool 中通过这个 AGSLocalMapResource 来调用 AO：

```
public class BufferPointToolAction implements MapToolAction{
    public void execute(MapEvent event) throws Exception {
        WebContext webContext = event.getWebContext();
        AGSLocalMapResource agsLocalMapResource =
(AGSLocalMapResource)webContext.getResourceById("ags1");
        IServerContext serverContext = agsLocalMapResource.getServerContext();

        WebPoint webPoint =
(WebPoint)event.getWebGeometry().toMapGeometry(webContext.getWebMap());
```

```
            Point point = (Point)AGSUtil.toAGSGeometry(webPoint);
            Object objPoint = AGSLocalUtil.createArcObjectFromStub(point, serverContext);
            ITopologicalOperator topologicalOperator = (ITopologicalOperator)objPoint;
            IPolycurve objPolygon = (IPolycurve)topologicalOperator.buffer(10);
            objPolygon.densify(-1, -1);
            PolygonN polygonN =
(PolygonN)AGSLocalUtil.createStubFromArcObject(objPolygon, PolygonN.class,
serverContext);
            WebPolygon webPolygon = (WebPolygon)AGSUtil.fromAGSGeometry(polygonN);

            WebSimplePolygonSymbol symbol = new WebSimplePolygonSymbol();
            symbol.setAntialiasing(true);
            symbol.setFillTransparency(0.7);
            symbol.setColor("0,255,0");
            symbol.setFillColor("255,0,0");

            WebGraphics webGraphics = webContext.getWebGraphics();
            GraphicElement element = new GraphicElement();
            element.setGeometry(webPolygon);
            element.setSymbol(symbol);
            webGraphics.addGraphics(element);

            webContext.refresh();
        }
}
```

当用户在浏览器中的地图上点击的时候 我们可以获得一个 WebPoint 类型的 ADF 对象，首先需要把这个对象转化为 AO 对象，这里必须通过两步走的方法：第一步把 ADF 对象转化为 SOAP 对象，第二步把 SOAP 对象转化为 AO 对象。具体可以参考上面的代码，主要使用了 AGSUtil 和 AGSLocalUtil 工具类的两个方法。

当我们得到 AO 的点以后，就可以使用 AO 的方法对要素进行处理了，关于 AO 的处理这里就省略了，关键在于处理完成以后，你还需要把 AO 对象的结果转化为 ADF 对象，同样，也需要两步方法，具体见上面的代码。

最后，让我们看一下这个通过 AO 实现的点击缓冲功能的效果：

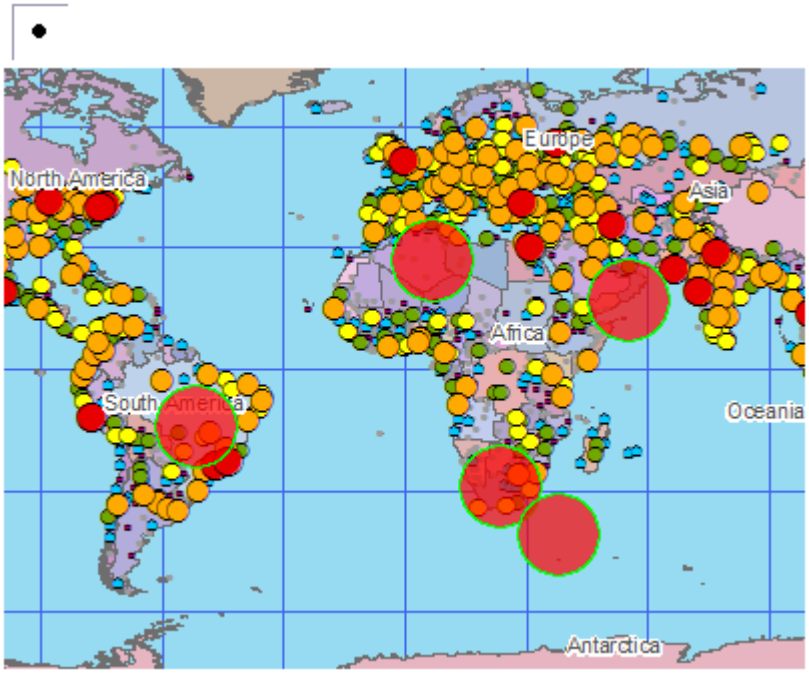


图 39　调用 AO 实现的缓冲效果

第十一章 有关 Web Service

在 IDE 的菜单中 ,你会看到有个很长的【ArcGIS Web Services Project】,别高兴，这不是给你自定义 Web Service 的，运行完向导你会发现生成的是一个 Java 桌面程序——有着一如既往丑陋的界面和很不友好的 GIS 功能。原来我一直当它是废物 ,不过前段时间它还真派上用场了：我们需要展示 ArcGIS Server 对 Web Service 的支持，要求必须抓取到浏览器与服务器交互的可读的 SOAP 消息；但是要知道，考虑到效率 ArcGIS Server 内部都是用二进制的 SOAP 进行交互 ,要抓到可读的 SOAP 消息 ,ArcGIS 的 Web Service 包必须被部署到浏览器端——最后，还真靠了这个丑陋的“ArcGIS Web Services Project”，做了很多修改封装成了 Applet 才达到了这个要求。

扯远了，说了这么多其实只为了说明这样一点：“ArcGIS Web Services Project”是一个使用 ArcGIS 的 Web Service 类库实现一些 GIS 功能的 Java 桌面程序，演示意义大于应用意义。下面我们首先来讨论一下如何使用 SOAP SDK 来访问 ArcGIS Server 原生的 Web Service；然后再谈谈怎么自定义 Web Service，以便其它业务系统调用或出于其它目的的 SOAP 访问。

一 通过 SOAP SDK 访问 ArcGIS Web Service

这里我们通过一个简单的功能来演示如何使用 SOAP SDK 来访问 ArcGIS Server，调用一个 Map Service 的功能。不管你是在 Java 桌面、Applet、ADF 还是自定义的 servlet 中，如果你需要直接使用 SOAP 和 ArcGIS 交互，那么下面的这段内容对你可能会有帮助。

这个简单的功能是“计算当前范围内的要素个数 “，这个功能是如此的简单以至于

我觉得看一下代码就可以明白了：

```
MapServerBindingStub mapserver = new MapServerBindingStub(url);

SpatialFilter filter = new SpatialFilter();
filter.setFilterGeometry(mapDescription.getMapArea().getExtent());
filter.setSpatialRel(EsriSpatialRelEnum.esriSpatialRelIntersects);
filter.setGeometryFieldName("SHAPE");
filter.setSubFields("*");
filter.setSearchOrder(EsriSearchOrder.esriSearchOrderSpatial);
filter.setWhereClause("");
filter.setSpatialReferenceFieldName("");
filter.setSpatialRelDescription("");

 int count = mapserver.queryFeatureCount("Layers", 0, (QueryFilter) filter);
 JOptionPane.showMessageDialog(this, "Feature count = " + count);
```

上面涉及到的各个 ArcGIS 的功能类都在 com.esri.arcgisws 包下，从命名上就可以看出这是专门为 Web Service 准备的。代码里用到的 url 是地图服务的地址，比如“http://host:8399/arcgis/services/name/MapServer“。在整个环节中你可能会对 MapServerBindingStub 的 queryFeatureCount 方法有一点兴趣 ,其实你打开 ArcGIS 帮助里的 SOAP SDK，找到 Map Services 下面的 QueryFeatureCount 方法就可以看到，QueryFeatureCount 需要 3 个参数，分别对应 DataFrame 名称、图层序号和 QueryFilter，正与上面的代码相符。换句话说，com.esri.arcgisws 包里的功能类就是 SOAP SDK 的 Java 实现。

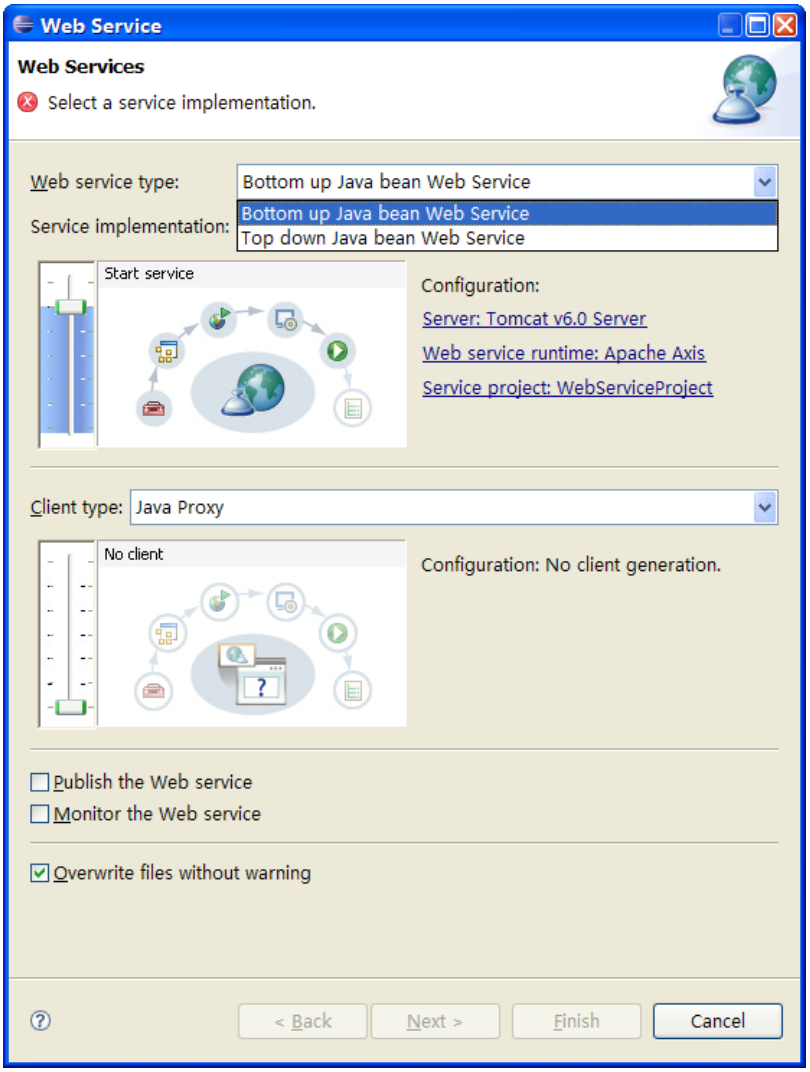
可是，不管是 ADF 还是 REST SDK 都提供了比 SOAP SDK 好用多的调用 ArcGIS Server 功能的方法，所以想来你会和我一样，对上面的这些内容兴趣寥寥——我觉得 ESRI 对 SOAP SDK 也是兴趣寥寥，以至于在 SOAP SDK 的文档里几乎都没有 Java 的示例代码。

好了，让我们抛开 ArcGIS 提供的 Web Service，来看看怎么自定义 Web Service

吧。我个人觉得自定义的 Web Service 是更加有用的，而且，功能也可以做得更强大。

二 两种方式自定义 Web Service

在 IDE 里创建一个 Web Service 工程是很容易的，只要找到【File】-【New】-【Web Service】菜单就可以了，点击这个菜单将会弹出下面的对话框：



这里的 Web service type 有两种选择：一种是 Bottom up、另外一种是 Top down。这两种方式分别对应了自下而上和自上而下两种新建 Web Service 的模式。如果你已经有了实现的功能类，现在想要把它封装成 Web Service，那么你应当选择 Bottom up 模式；如果你首先定义了 Web Service 的描述文件（wsdl），现在想要实现这个 wsdl

的功能，那么你应当选择 Top down 模式。

三 实现 Web Service 的功能

不管通过 Bottom up 还是 Top down 方式，当你把 Web Service 的框架搭起来以后，剩下的工作就是如何去实现 Web Service 的功能了。下面让我们通过讲解一个网络分析 Web Service 的实现，来了解自定义 Web Service 的运作。

这个网络分析的 Web Service 需要的功能是预先定义好的，其中有比如上下游追溯、连通性分析等等，这里只挑连通性分析这个小功能进行说明。

首先，我们通过给定的 wsdl 新建了一个 Web Service 工程，然后找到实现类，在这个类中已经根据 wsdl 自动生成了对应各个功能的方法，这里就连通性分析功能进行讲解，对应的方法是 connectivity，下面是这个方法的实现：

```
public class TestGISAnalysisSoapBindingImpl implements TestGISAnalysisPortType      {
    private GeoNetworkAnalysis geoNetAnalysis;

    public TestGISAnalysisSoapBindingImpl()          {
        geoNetAnalysis = new GeoNetworkAnalysis();
    }

    public java.lang.String connectivity(String sourceDeviceType
            , String sourceDeviceID
            , String targetDeviceType
            , String targetDeviceID,
            String type) throws java.rmi.RemoteException{
        String result = "";

        try{
            result = geoNetAnalysis.findPath(Integer.parseInt(sourceDeviceType)
            , Integer.parseInt(sourceDeviceID)
            , Integer.parseInt(targetDeviceType)
            , Integer.parseInt(targetDeviceID)
            , type);
        }catch (Exception ex){
```

```
                ex.printStackTrace();
            }

            return result;
        }
}
```

怎么这么简单？当然，因为这里对功能做了封装，真正的分析功能在 geoNetAnalysis 这个对象中，在这个对象中进行的都是直接对 AO 的操作，我们通过一些代码片段来看一下这个功能的实现：

```
public class GeoNetworkAnalysis{
    public GeoNetworkAnalysis() {
        try {
            EngineInitializer.initializeEngine();
            new
AoInitialize().initialize(esriLicenseProductCode.esriLicenseProductCodeArcInfo);

            ResourceBundle bundle = ResourceBundle.getBundle("geometrynetwork");

            traceFlowSolver = new TraceFlowSolver();
            INetSolver netSolver = (INetSolver) traceFlowSolver;
            geometricNetwork = openNetwork(bundle.getString("workspacePath"),
                    bundle.getString("datasetName"),bundle.getString("networkName"));
            pNetElements = new INetElementsProxy(geometricNetwork.getNetwork());

            netSolver.setSourceNetworkByRef(geometricNetwork.getNetwork());
            ...

        } catch (Exception e) {
            e.printStackTrace();
        }
    }

    /**
     * 连通性分析
     *
     * @param sourceLyrID
     *                起点的图层 ID
     * @param sourceFeatID
     *                起点的要素 ID
     * @param targetLyrID
```

```
     *                  终点的图层 ID
     * @param targetFeatID
     *                  终点的要素 ID
     * @param type
     * @return
     */
      public String findPath(int sourceLyrID
            , int sourceFeatID
            , int targetLyrID
            , int targetFeatID
            , String type) throws Exception{
            this.putTraceOrigins(getOriginPoints(sourceLyrID
                  ,sourceFeatID
                  ,targetLyrID
                  ,targetFeatID));

                  IEnumNetEID[]   junctionEIDS = new IEnumNetEID[1];
                  IEnumNetEID[] edgeEIDs = new IEnumNetEID[1];
                  Object[][] costArray = new Object[1][1];

                  traceFlowSolver.findPath(esriFlowMethod.esriFMConnected,
                              esriShortestPathObjFn.esriSPObjFnMinSum
                              , junctionEIDS
                              , edgeEIDs
                              , 1
                              , costArray);
                  ...

                  Set<IFeature> junctions = this.getSolverResultFeature(junctionEIDS[0]);
                  Set<IFeature> edges = this.getSolverResultFeature(edgeEIDs[0]);
                  return FeatureUtil.networkResultToJSON(junctions, edges);
      }
}
```

注意，上面只是代码片段，整个功能的实现代码太长了，和我们要了解的 Web Service 其实又没有什么关系，所以，就看个大概吧。

做完这些，在很多场合都可以调用这个自定义的 Web Service 了，比如上面这个功能我们是在 Flex 中调用的：

```
<mx:Script>
      <![CDATA[
```

```
        private function analysisConnectivity():void
        {
                    opt = wsAnalysis.getOperation("connectivity");
                    args = new Object();
                    args.sourceDeviceType = g1.attributes.layerId;
                    args.sourceDeviceID = g1.attributes.OBJECTID;
                    args.targetDeviceType = g2.attributes.layerId;
                    args.targetDeviceID = g2.attributes.OBJECTID;
                    args.type = "connectivity";
                    opt.arguments = args;
                    opt.send();
        }
        ]]>
 </mx:Script>

 <mx:WebService id="wsAnalysis"
 wsdl="http://localhost:8080/WebServiceProject/wsdl/TestGISAnalysisHttpSoapEndpoint.ws
dl"
        showBusyCursor="true"
        concurrency="last"
        result="onWsAnalysisResult(event)"
        fault="onWsAnalysisFault(event)"/>
```

设想一下，是不是不同的业务系统之间交互也可以做了呢？

第十二章 扩展 ArcGIS Server

这一章其实和ADF已经没有太大关系了，在ArcGIS的帮助里也是在ArcGIS Server开发章节里。考虑到使用 ADF 开发的一般对功能要求比较多一点，所以对于怎么扩展 ArcGIS Server 可能比较关心，反正 ADF 差不多也讲完了，这一章就算是一个结束吧。

从9.3.1开始，ArcGIS Server提供了Java开发者更多的可扩展性。通过扩展ArcGIS Server 可以使某些自定义的功能更加完善和强大、使你的某些工作可以更加模块化并和 ArcGIS Server 很好地结合在一起。下面让我们来看看 ArcGIS Server 可以通过什么样的途径进行扩展。

一 Utility Objects

Utility Objects 主要是为了提升 ArcGIS Server 在 Local 方式连接下某些操作的性能而准备的。在此之前，让我们首先去了解一些 ADF 调用服务器端 AO 对象的过程和特点。

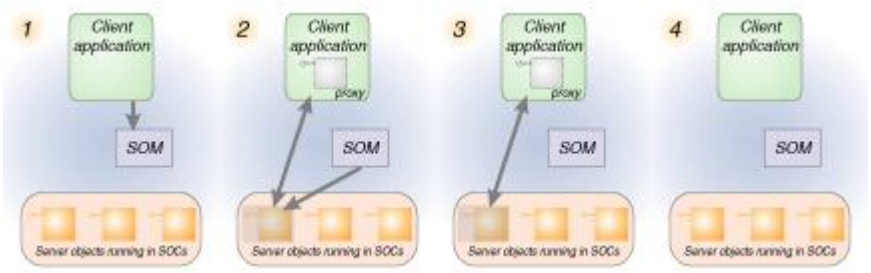


图 40 ADF 调用服务器端对象的过程

如图 40 所示，当用户发生请求的时候，ADF 首先会和 SOM 交互（图中 1）；SOM 将根据服务器情况分配一个 SO 用以处理这个用户的请求，同时返回这个 SO 的代理（proxy）给 ADF（图中 2）；然后，ADF 就会通过这个代理和服务器的 SO 进行交互

（图中 3，注意，这是 ADF 应用和 GIS 服务器之间的交互，是跨进程的）；当请求完成后，ADF 释放相应的资源供其它用户使用（图中 4）。

跨进程的调用会消耗比较多的系统资源，响应速度较慢；当 ADF 应用和 GIS 服务器分开部署的时候，还有相应的网络开销。因此，当你的 ADF 应用中有大量细粒度操作，频繁地对服务器端 AO 对象进行调用的时候，性能会非常低下。Utility Objects 就是为了解决这个问题而诞生的。

Utility Objects 的原理就是把上述分别属于 ADF 和 GIS 服务器两个进程中的操作合并到一起，都放到 ArcGIS Server 中执行，这样可以极大地减小进程间通讯的开销。在实现上 Utility Objects 也并不复杂，你只需要把原来直接在 ADF 中实现的功能，封装到一个实现了自定义接口的类中，然后在 ADF 中调用这个自定义的功能类就可以了。下面让我们看一下这个大体的流程，在 ArcGIS Server 帮助中有这样一个例子：把计算一个 FeatureLayer 中所有多边形面积总和的功能封装成一个 Utility Object。

首先，我们需要定义一个接口 ICalculateArea，并使用 CalcAreaUtil 类来实现这个接口，注意，Utility Object 的接口和类都需要使用“@ArcGISExtension”注记来声明：

```
@ArcGISExtension
public interface ICalculateArea{
    public double calculateArea(IFeatureLayer featureLayer)
        throws IOException, AutomationException;
}

@ArcGISExtension
public class CalcAreaUtil implements ICalculateArea{
    public double calculateArea(IFeatureLayer featureLayer){
        double totalArea = 0;
        IFeatureClass featureClass = featureLayer.getFeatureClass();
        int numFeatures = featureClass.featureCount(null);
```

```
            IFeatureCursor featureCursor = featureLayer.search(null, true);

            IFeature feature = null;
            Polygon polygon = null;
            for (int i = 0; i < numFeatures; i++){
                feature = featureCursor.nextFeature();
                polygon = (Polygon)feature.getShape();
                totalArea += polygon.getArea();
            }
            return totalArea;
        }
}
```

有两种方法可以部署这个 Utility Object，当然，作为一个懒人我比较倾向于选择相对简单、绿色的方法：将上面实现的这个类编译成 jar 包，再将这个 jar 包复制到服务器上的 %ARCGISHOME%/java/lib/ext 目录下。另外一种复杂一些，而且让我觉得有点脏兮兮的方法，有兴趣的可以到帮助文档中寻觅一下。

好，下面我们就可以使用这个 Utility Object 了：

```
IFeatureLayer featureLayer = mapServer.getLayer(" ", 2);
ICalculateArea calArea = (ICalculateArea)
    com.esri.arcgis.server.ServerContext.createObject(CalcAreaUtil.class);
double totalArea = calArea.calculateArea(featureLayer);
```

在这段代码中我们看到，调用“计算一个 FeatureLayer 中多边形总面积”的功能全部由一个 GIS 服务器端的 CalcAreaUtil 类实现，ADF 中只需要使用 IServerContext 来实例化这个类，再调用它相应的方法即可。

二 使用 Utility Objects 提高查询性能【案例】

这是我们使用 Utility Objects 解决某个用户的一个实际问题。这个用户通过调用 AO 进行空间查询，然后在此基础上进行分页，分页的时候遍历该页所有的 IFeature，提取属性信息，构造成自己的结果返回——结果，遇到了严重的性能问题，在上万条记

录中查询一般两三分钟才可以出结果，这在一个国家级项目中是难以忍受的。

经过添加日志分析，发现 AO 查询的速度非常快，问题就是出在遍历 IFeature 这个操作上，是不停地进行跨进程调用产生的。就在这个时候 9.3.1 正式发布了，Utility Objects 有如救星般地降临了！

由于在上一小节关于怎么定义 Utility Object 已经讲得比较清楚，这里就不在啰嗦怎么封装接口和实现类了。下面是封装起来的功能代码，主要就是遍历分页要素并提取有用的信息，当然，原先这部分代码是在 Web 层执行的。

```
IFeature feature = featureCursor.nextFeature();
int featureCount = 0;
while (feature != null){
        if (featureCount >= first + pageSize){
                break;
        }
        if (featureCount >= first){
                result.add(ServiceUtil.feature2Einfo(feature));
        }
        featureCount++;
        feature = featureCursor.nextFeature();
}
```

经过这样的处理，性能有了极大的提升，一个简单的测试可以说明情况：这里做了一个半径 1000 公里的空间查询，返回结果总数为 30437 条。

分页起始记录	分页记录数	优化前(s)	优化后(s)
0	500	179.406	5.203

如果你有细粒度的操作，又遇到了性能问题，现在是不是很心动？

三 Server Object Extensions

Server Object Extensions（以下简称 SOE）是 Server Object 的扩展，比如

MapServer、GPServer 等等。和 Utility Objects 不同，SOE 只在 Server Object 被创建的时候被初始化，它的生命周期和 Server Object 是相同的。相比 Utility Objects，SOE 有以下的一些特点：

- 开发人员不需要手动实例化 SOE，SOE 是在 Server Object 被创建的同时被实例化的
- ArcGIS Server 的管理程序可以显示 SOE 的参数配置
- SOE 的运行信息将被记录到 ArcGIS Server 的日志中去
- SOE 的功能可以通过 SOAP 访问

更直观一点，让我们看一下下面这张截图，这是一个添加了一个自定义 SOE 的 Map Service 管理界面：

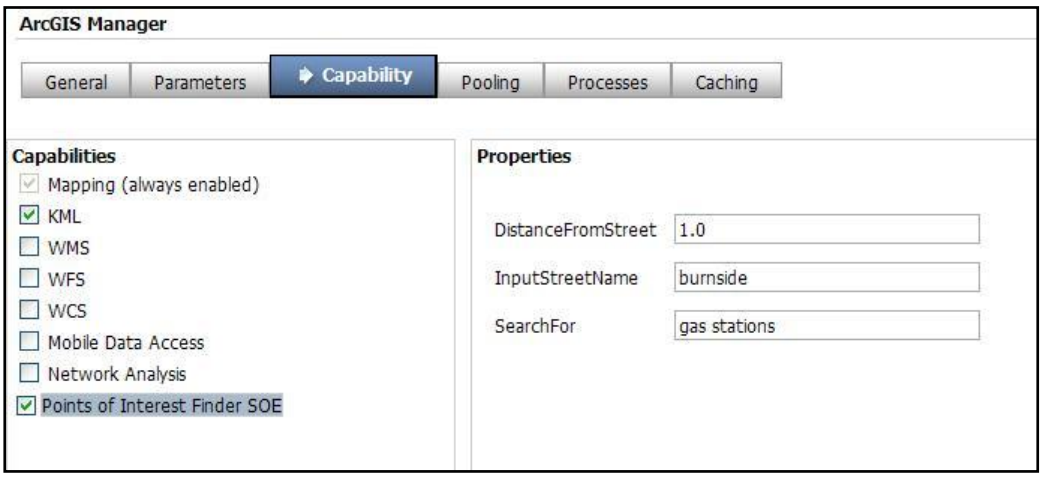


图 41　添加了 SOE 的 Map Service

是不是可以把 SOE 理解成一个服务的自定义 Capability？似乎 ESRI 就是这样理解的。那下面我们就通过自定义一个新的 Capability 来看看 SOE 的用法。

四 创建一个自定义的 Server Object Extension【案例】

首先，我们使用 IDE 插件来帮助我们新建这个 SOE，打开菜单的【File】-【New】

-【Server Object Extension】打开新建窗口，如图 42；然后在如图 43 的属性页中做些自定义的编辑，ArcGIS 插件会自动新建一些类和配置文件。

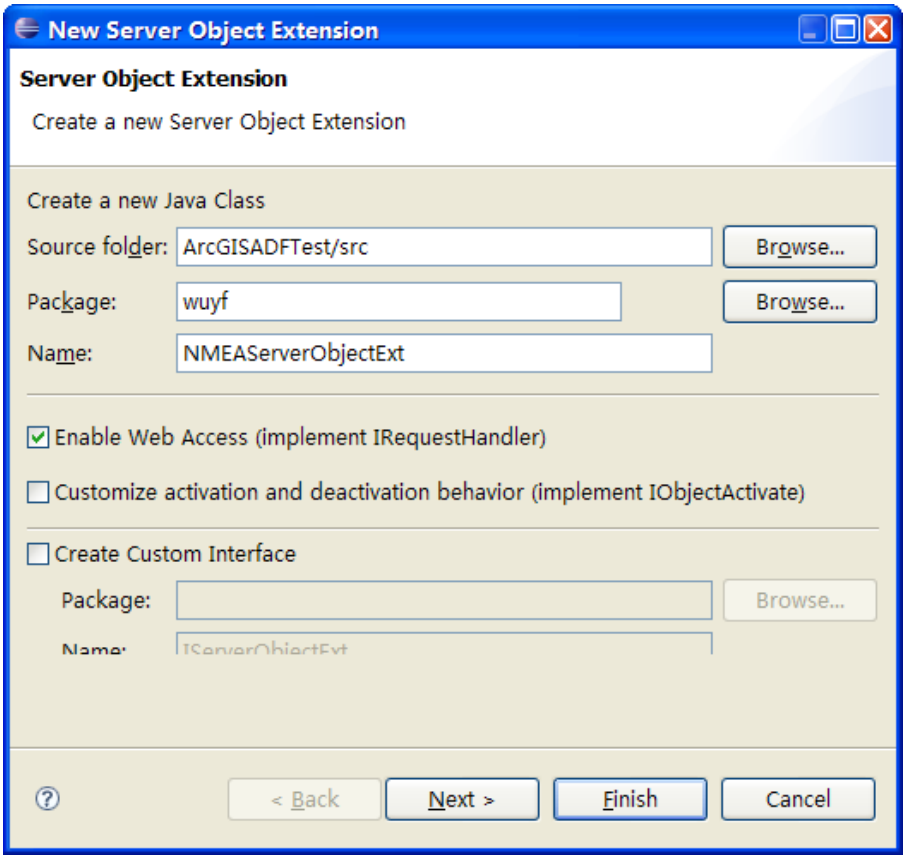


图 42　新建 SOE 窗口

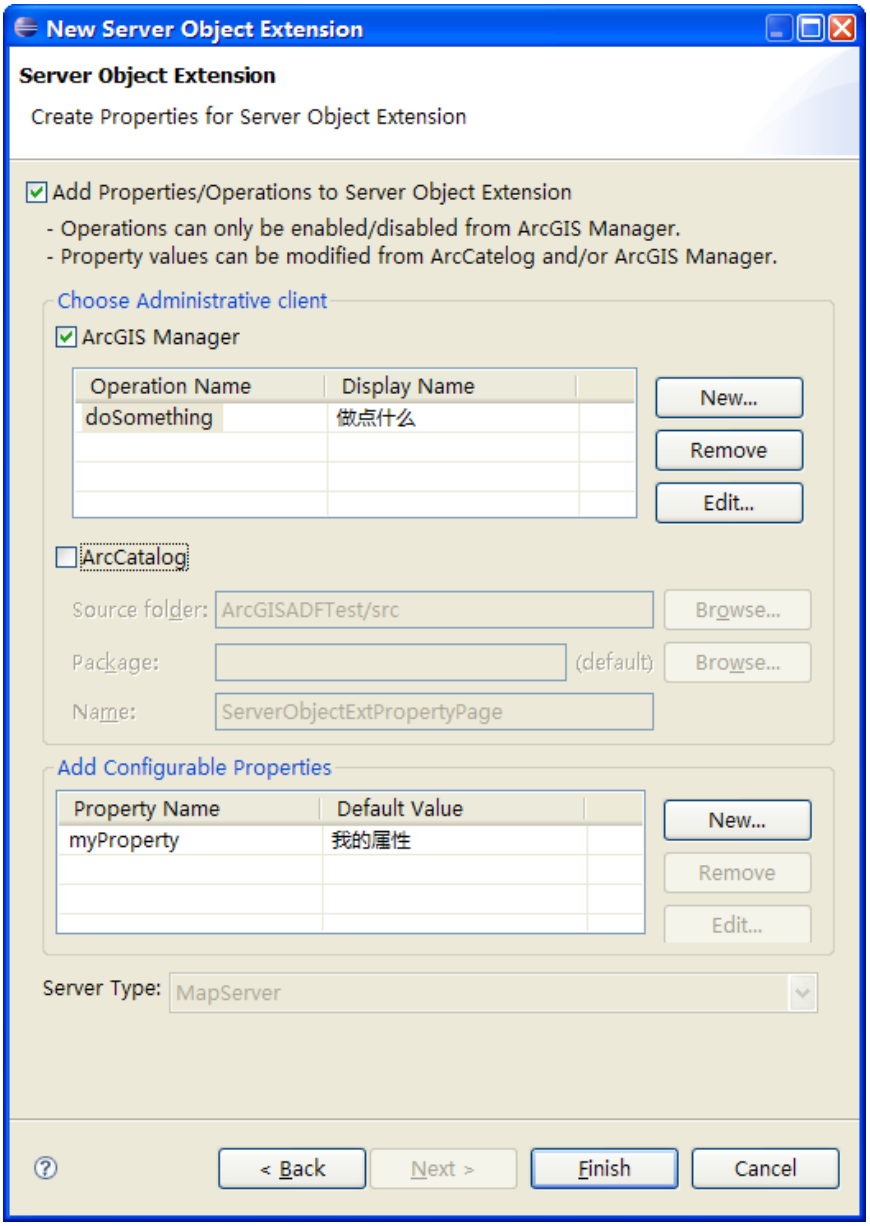


图 43　编辑一些 SOE 的属性

好，现在虽然我们没有添加什么实质的功能代码，但是 SOE 的框架已经起来了，让我们尝试部署这个 SOE 到 ArcGIS Server 中去。

首先，右键工程，选择【Export】，找到 Java 文件夹下的 JAR 文件选项，把这个 SOE 导出成 jar 包；然后，复制这个 jar 文件到 %ARCGISHOME%/java/lib/ext 目录下 ;最后 需要用一个工具添加这个 SOE 到 Map Service 这个工具叫 SOMManager，你可以在 %ARCGISHOME%/java/tools 目录下打开控制台，然后执行以下命令：

```
soemanager -a [server name] [domain name] [username] [password] [fully qualified SOE name] [SOE display name]
```

比如上面在 wuyf 命名空间下新建了一个 NMEAServerObjectExt 类，因此大概的命令就是如下这个样子的：

```
soemanager -a localhost localhost arcgismanager passwd wuyf.NMEAServerObjectExt NMEA
```

好了，下面在添加一个新的 Map Service 的时候就可以看到一个新的 Capability 了：

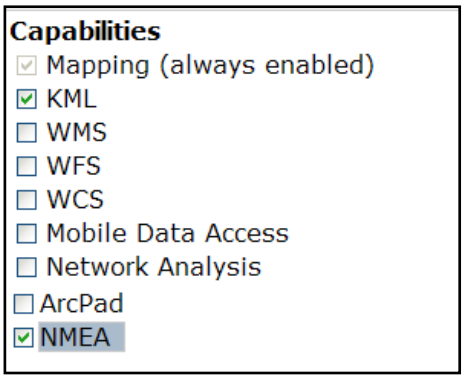


图 44　自定义的 NMEA Capability

结语

从开始计划写这篇教程，到初步完稿已经过了好几个月了。从开始写教程起，我每次都把我写的内容同步更新到我的 blog 上，原来我使用的是 Google Blogger，可恼的是中途 Google Blogger 又被伟大的墙给屏蔽了，很多以前的文章和图片都看不到了，因此我又把这些文章复制到我现在的 blog(http://warrenwyf.spaces.live.com/)上，不过可惜的是原来文章中链接的图片始终看不到了，文章的版式也变得乱七八糟。

在写这个教程的过程中，进度也是时快时慢，有一段时间一两个月基本没有更新，这是因为有个大项目让我们几个同事每天加班，连周末都没有。也希望一直等待的朋友可以谅解。

还一直有人问我这个教程有没有 PDF 的版本，我总是说等我都写完整理一下再发布。现在终于可以发布这个 PDF 了，我也非常高兴。

祝大家学习工作愉快，搞 GIS 的同时也不要忘记享受美好的生活。

附录

一 配置 ArcGIS Server Java 开发环境（Eclipse 3.3 为例）

ArcGIS Server 的 Java 开发首先需要安装 Java Web 服务器， ArcGIS Server 9.3 支持的服务器种类有：

Apache Tomcat

IBM WebSphere

JBoss

我这里采用的是 Tomcat 6.0.16。

如果我们使用 eclipse，需要 eclipse 基本平台和 WTP 2.0，建议直接下载针对 J2EE 开发的 All In One 套件包（3.3，其他版本请直接查看 eclipse 网站）：

http://www.eclipse.org/downloads/download.php?file=/technology/epp/downloads/release/europa/winter/eclipse-jee-europa-winter-win32.zip

我们可以选择在本地安装 ArcGIS 插件：

在 eclipse 主菜单中选择 Help->Software Updates->Find and Install，选择“Search for new features to install”，点击右侧的“New Local Site”，添加本地插件升级路径：“%ARCGISHOME%/java/tools/eclipse_plugin /arcgis_update_site/server/”，确定。另外，与此类似，添加“%ARCGISHOME%/java/tools /eclipse_plugin/arcgis_update_site/doc”。

在上述操作完成后，下一步 eclipse 将自动扫描上述路径，发现可用插件时会给出提示，后续操作根据提示下一步即可。

我们也可以从 Internet 安装 ArcGIS 插件：

相比从本地安装 ArcGIS 插件，从 Internet 安装更能在以后保持软件的升级更新。安装过程和从本地类似，只不过在添加升级站点的时候，选择“New Remote Site”，添加升级地址：“http://downloads.esri.com/EDN/java/plugins/eclipse/v93/”。

Ok，当 eclipse 提示你重启时，插件已经安装完成，重启 eclipse 后可以看到新的 Overview 屏幕，上面包含了新增的 ArcGIS 的内容。

在这个基础上，让我们新建一个项目进行开发环境的测试：首先，在 eclipse 主菜单选择 New->Project...，点击“ESRI Templates”下的“ArcGIS Web Project”；然后，在新建 ArcGIS Web Project 窗口对项目进行必要的配置，还没有配置 Target Runtime 时，首先点击右侧的“New...”按钮新建一个已经安装的 J2EE Web 服务器；点击下一步，在此界面应该可以添加 GIS Server 并选择需要添加到工程中的图层；最后运行项目，出现地图应用界面，表示项目部署运行成功。

二 配置 ArcGIS Server Java 开发环境（NetBeans 6 为例）

同样，我们首先需要配置 Java Web 服务器，然后我们需要下载并安装 NetBeans，ArcGIS Server 9.3 支持的 NetBeans 版本为 6.x，我们可以从 NetBeans 网站免费下载：http://www.netbeans.org/， NetBeans 下载包的类型建议选择 Web&J2EE（第一列）。

安装完 NetBeans 后，在 NetBeans 主菜单中选择 工具->插件，切换到“设置”选项卡，点击 “添加”按钮，添加插件升级路径：

“http://downloads.esri.com/edn/java/plugins/netbeans/v93 /updates.xml”，ok，接下来按照提示操作就可以了。

安装成功的话，在“插件”选项卡可以看到 ESRI ArcGIS Java ADF web controls NetBeans Module 的内容。

下面，我们新建一个模板工程对开发环境进行测试。首先，在 NetBeans 主工具条点击“新建项目”，选择“Web 应用程序”；然后，按照 NetBeans 提示进行操作，在选择框架这一步注意选中“ESRI Web Java ADF 9.3”，并设置数据源；最后运行项目，出现地图应用界面，表示项目部署运行成功。