文章
姚 鑫 · 七月 31, 2022 阅读大约需 8 分钟

第十三章 手动创建 REST 服务（一）

第十三章 手动创建 REST 服务（一）

本附录描述了如何通过继承 %CSP.REST 类来手动创建 REST 服务；此过程创建了一个手动编码的 REST 服务，它不能与所有 API 管理工具一起使用。

手动创建 REST 服务的基础知识

要手动定义 REST 服务，请执行以下操作：

- 创建一个 REST 服务类 — %CSP.REST 的子类。在子类中：
  - 定义一个 URL 映射，该映射指定为 REST URL 和 HTTP 方法执行的 IRIS 方法。
  - 可以选择指定 UseSession 参数。此参数控制每个 REST 调用是在其自己的 Web 会话下执行还是与其他 REST 调用共享一个会话。
  - （可选）覆盖错误处理方法。

如果想将实现代码与调度代码分开，可以在单独的类中定义实现 REST 服务的方法，并从 URL 映射中调用这些方法。

- 定义一个使用 REST 服务类作为其调度类的 Web 应用程序。

要定义 Web 应用程序及其安全性，请转至 Web 应用程序页面（单击System Administration > Security > Applications > Web Applications）。

定义 Web 应用程序时，将 Dispatch Class 设置为 REST 服务类的名称。

此外，将应用程序的名称指定为 REST 调用的 URL 的第一部分。示例名称为 /csp/mynamespace 或 /csp/myapp，但可以指定 URL 中允许的任何文本。

可以在一个命名空间中定义多个 REST 服务类。每个拥有自己入口点的 REST 服务类都必须拥有自己的 Web 应用程序。

创建 URL 映射

在 REST 服务类中，定义一个名为 UrlMap 的 XData 块，它将 REST 调用与实现该服务的方法相关联。它可以直接将调用发送到基于 URL 内容的方法，也可以将调用转发到基于 URL 的另一个 REST 服务类。如果 Web 应用程序正在处理少量相关服务，可以将调用直接发送到实现它的方法。但是，如果 Web 应用程序正在处理大量不同的服务，可以定义单独的 REST 服务类，每个类都处理一组相关的服务。然后将 Web 应用程序配置为使用中央 REST 服务类，该类将 REST 调用转发到其他适当的 REST 服务类。

如果 %CSP.REST 的子类将调用直接发送到方法，则 URLMap 包含一个 <Route>定义，其中包含一系列 <Route> 元素。每个 <Route> 元素指定一个类方法，为指定的 URL 和 HTTP 操作调用。通常 REST 使用 GET、POST、PUT 或 DELETE 操作，但可以指定任何 HTTP 操作。 URL 可以选择包含作为 REST URL

的一部分指定并作为参数传递给指定方法的参数。

如果 %CSP.REST 的子类将调用转发到 %CSP.REST 的其他子类，则 UrlMap 包含一个 <Route> 定义，其中包含一系列 <Route> 元素。 <Route> 元素将具有指定前缀的所有调用转发到另一个 REST 服务类，然后该服务类将实现该行为。它可以通过将调用直接发送到方法或将其转发到另一个子类来实现该行为。

重要提示：IRIS 将传入的 REST URL 与每个<Route>的 URL 属性和每个 <Map>的 Prefix 属性进行比较，从 URL 映射中的第一项开始，并在使用相同 HTTP 的第一个可能匹配处停止请求方法。因此<Route> 中元素的顺序很重要。如果传入的 URL 可以匹配 URL 映射的多个元素，IRIS 使用第一个匹配元素并忽略任何后续可能的匹配。

带有 <Route> 元素的 URLMap

IRIS 将传入的 URL 和 HTTP 请求方法与 URL 映射中的每个 <Route>元素进行比较。它调用在第一个匹配的 <Route> 元素中指定的方法。<Route> 元素包含三个部分：

- Url — 指定调用 REST 服务的 REST URL 的最后一部分的格式。 Url 由文本元素和以:（冒号）开头的参数组成。
- Method — 指定 REST 调用的 HTTP 请求方法：通常是 GET、POST、PUT 或 DELETE，但可以使用任何 HTTP 请求方法。应该选择适合服务正在执行的功能的请求方法，但 %CSP.REST 类不会对不同方法执行任何特殊处理。应该以全部大写字母指定 HTTP 请求方法。
- Call — 指定调用以执行 REST 服务的类方法。默认情况下，此类方法在 REST 服务类中定义，但可以显式指定任何类方法。

例如，考虑以下 <Route>：

```
<Route Url="/echo" Method="POST" Call="Echo" Cors="false" />
```

这指定 REST 调用将以 /echo 结束并使用 POST 方法。它将调用定义 REST 服务的 REST.DocServer 类中的 Echo 类方法。 Cors 属性是可选的。

完整的 REST URL 由以下部分组成：

- IRIS 服务器的服务器名称和端口。在本章中，示例中使用了服务器名称和端口 http://localhost:52773/。
- Web 应用程序页面上定义的 Web 应用程序的名称（单击System Administration > Security > Applications > Web Applications）。 （例如，/csp/samples/docserver）
- <Route>元素的 Url 属性。如果 Url 属性的一段前面有一个 :（冒号），则它表示一个参数。参数将匹配该 URL 段中的任何值。该值作为参数传递给方法。

对于前面的示例，TCP 跟踪实用程序显示的完整 REST 调用是：

```
POST /csp/samples/docserver/echo HTTP/1.1
Host: localhost:52773
```

如果要将实现 REST 服务的代码与 %CSP.REST 调度代码分开，可以在另一个类中定义实现 REST 服务的方法，并在 Call 元素中指定类和方法。

指定参数

以下 <Route> 定义在 URL 中定义了两个参数，命名空间和类：

```
<Route Url="/class/:namespace/:classname" Method="GET" Call="GetClass" />
```

REST 调用 URL 以 /csp/samples/docserver/class/ 开头，URL 的下两个元素指定两个参数。 GetClass() 方法将这些参数用作您要查询的命名空间和类名。例如，考虑这个 REST 调用：

```
http://localhost:52773/csp/samples/docserver/class/samples/Cinema.Review
```

此 REST 调用调用 GetClass() 方法并将字符串 “ samples ” 和 “ Cinema.Review ” 作为参数值传递。 GetClass() 方法具有以下签名：

```
/// ???????????
ClassMethod GetClass(pNamespace As %String,
                     pClassname As %String) As %Status
{
```

为单个 URL指定多个路由

对于给定的 URL，可以支持不同的 HTTP 请求方法。可以通过为每个 HTTP 请求定义单独的 ObjectScript 方法，或使用检查请求的单个 ObjectScript 方法来实现。

以下示例对单个 URL 的每个 HTTP 请求方法使用不同的方法：

```
<Route Url="/request" Method="GET" Call="GetRequest" />
<Route Url="/request" Method="POST" Call="PostRequest" />
```

使用这些路由，如果使用 HTTP GET 方法调用 URL /csp/samples/docserver/request，则会调用 GetRequest() 方法。如果使用 HTTP POST 方法调用它，则调用 PostRequest() 方法。

相反，可以使用以下 <Route> 定义：

```
<Route Url="/request" Method="GET" Call="Request" />
<Route Url="/request" Method="POST" Call="Request" />
```

在这种情况下，Request() 方法处理对 GET 或 POST 操作的调用。该方法检查 %request 对象，它是 %CSP.Request 的一个实例。在此对象中，URL 属性包含 URL 的文本。

路由图中的正则表达式

可以在路线图中使用正则表达式。 建议仅在没有其他方法来定义 REST服务以满足需求时才这样做。

在内部，用于在 URL 中定义参数的:parameter-name 语法是使用正则表达式实现的。指定为 :parameter-name 的每个段都转换为包含重复匹配组的正则表达式，特别是 ([^/]+)
正则表达式。此语法匹配任何字符串（长度非零），只要该字符串不包含 /（斜杠）字符。因此 GetClass() 示例，即 Url="/class/:namespace/:classname"，等效于：

```
<Route Url="/class/([^/]+)/([^/]+)" Method="GET" Call="GetClass" />
```

其中有两个匹配组指定了两个参数。

在大多数情况下，这种格式提供了足够的灵活性来指定 REST
URL，但高级用户可以直接在路由定义中使用正则表达式格式。 URL
必须与正则表达式匹配，并且由一对括号指定的每个匹配组定义一个要传递给该方法的参数。

例如，考虑以下路线图：

```
<Routes>
<Route Url="/Move/:direction" Method="GET" Call="Move" />
<Route Url="/Move2/(east|west|north|south)" Method="GET" Call="Move" />
</Routes>
```

对于第一个路由，参数可以有任何值。无论参数有什么值，都会调用 Move()
方法。对于第二条路线，参数必须是东西北或南之一；如果您使用其他参数值调用第二个路由，则不会调用 Move()
方法，并且 REST 服务会返回 404 错误，因为找不到资源。

这个简单的例子只是为了演示常用参数语法和正则表达式之间的区别。在此处讨论的情况下，不需要正则表达式，因为 Move() 方法可以（并且应该）检查参数的值并做出适当的响应。但是，在以下情况下，正则表达式很有帮助：

如果参数是可选的。在这种情况下，请使用正则表达式 ([^/]*) 而不是 :parameter-name 语法。例如：

```
<Route Url="/Test3/([^/]*)" Method="GET" Call="Test"/>
```

当然，被调用的方法也必须能够处理参数的空值。

- 如果参数是最后一个参数并且其值可以包含斜杠。在这种情况下，如果需要参数，请使用正则表达式 ((?s).+) 而不是 :parameter-name 语法。例如：

```
<Route Url="/Test4/((?s).+)" Method="GET" Call="Test"/>
```

或者，如果此参数是可选的，请使用正则表达式 ((?s).*) 而不是 :parameter-name 语法。例如：

```
<Route Url="/Test5/((?s).*)" Method="GET" Call="Test"/>
```

带有 <Map> 元素的 URLMap

IRIS 将传入 URL 与 URL 映射中每个 <Map> 元素中的前缀进行比较。它将传入的 REST 调用转发到第一个匹配的 <Map> 元素中指定的 REST 服务类。该类处理 URL 的其余部分，通常调用实现服务的方法。 <Map>
元素有两个属性：

- Prefix前缀 - 指定要匹配的 URL 段。传入的 URL 通常在匹配段之后有其他段。
- Forward后缀 - 指定将处理匹配段之后的 URL 段的另一个 REST 服务类。

考虑以下包含三个 <Map> 元素的 URLMap。

```
XData UrlMap
{
  <Routes>
```

```
    <Map Prefix="/coffee/sales" Forward="MyLib.coffee.SalesREST"/>
    <Map Prefix="/coffee/repairs" Forward="MyLib.coffee.RepairsREST"/>
    <Map Prefix="/coffee" Forward="MyLib.coffee.MiscREST"/>
  </Routes>
}
```

此 UrlMap 将 REST 调用转发到三个 REST 服务类之一：MyLib.coffee.SalesREST、MyLib.coffee.RepairsREST 或 MyLib.coffee.MiscREST。

调用这些 REST 服务之一的完整 REST URL 包含以下部分：

- IRIS 服务器的服务器名称和端口，例如 http://localhost:52773/
- Web 应用程序页面上定义的 Web 应用程序名称（单击System Administration > Security > Applications > Web Applications）。例如，这些 REST 调用的 Web 应用程序可以命名为 /coffeeRESTSvr
- <Map>元素的前缀。
- REST URL 的其余部分。这是接收转发的 REST 请求的 REST 服务类将处理的 URL。

例如，以下 REST 调用：

```
http://localhost:52773/coffeeRESTSvr/coffee/sales/reports/id/875
```

匹配带有前缀 /coffee/sales 的第一个 <Map> 并将 REST 调用转发到 MyLib.coffee.SalesREST 类。该类将查找 URL 其余部分的匹配项，“ /reports/id/875 ”。

作为另一个示例，以下 REST 调用：

```
http://localhost:52773/coffeeRESTSvr/coffee/inventory/machinetype/drip
```

匹配带有前缀 /coffee 的第三个 <Map> 并将 RES 调用转发到 MyLib.coffee.MiscREST 类。该类将查找 URL 其余部分的匹配项，“ /inventory/machinetype/drip ”。

注意：在此 URLMap 示例中，如果带有前缀="/coffee" 的 <Map> 是第一个映射，则所有带有 /coffee 的 REST 调用将被转发到 .coffee.MiscREST 类，即使它们匹配以下之一 <Map> 元素。 <Routes> 中的 <Routes>元素的顺序很重要。

#REST API #Caché

源 URL: https://cn.community.intersystems.com/post/%E7%AC%AC%E5%8D%81%E4%B8%89%E7%AB%A0-%E6%89%8B%E5%8A%A8%E5%88%9B%E5%BB%BA-rest-%E6%9C%8D%E5%8A%A1%EF%BC%88%E4%B8%80%EF%BC%89