文章
姚 鑫 · 六月 5, 2021 阅读大约需 7 分钟

第六章 Caché JSON 使用JSON适配器

第六章 Caché JSON 使用JSON适配器

JSON适配器是一种将ObjectScript对象(registered, serial or persistent)映射到JSON文本或动态实体的方法。本章涵盖以下主题：

- 导出和导入-介绍启用JSON的对象并演示%JSON.Adaptor导入和导出方法
- 带参数映射-描述控制如何将对象属性转换为JSON字段的属性参数。
- 使用扩展数据映射块-介绍将多个参数映射应用到单个类的方法。
- 格式化JSON-演示如何使用%JSON.ForMatter格式化JSON字符串。
- %JSON快速参考-提供本章中讨论的每个%JSON类成员的简要说明。

Exporting and Importing

从JSON序列化或序列化到JSON的任何类都需要子类%JSON.Adaptor，它包括以下方法：
- %JSONExport()将启用JSON的类序列化为JSON文档，并将其写入当前设备。
- %JSONExportToStream()将启用JSON的类序列化为JSON文档并将其写入流。
- %JSONExportToString()将启用JSON的类序列化为JSON文档并将其作为字符串返回。
- %JSONImport()将JSON作为字符串或流导入，或者作为%DynamicAbstractObject的子类导入，并返回启用JSON的类的实例。

为了演示这些方法，本节中的示例将使用这两个类：

启用JSON的类Model.Event和Model.Location

```
Class Model.Event Extends (%Persistent, %JSON.Adaptor)
{
  Property Name As %String;
  Property Location As Model.Location;
}


Class Model.Location Extends (%Persistent, %JSON.Adaptor)
{
  Property City As %String;
  Property Country As %String;
}
```

如所见，我们有一个持久Event类，有一个Location属性类型为Model.Location。这两个类都继承自%JSON.Adaptor。这使能够填充对象图，并将其直接导出为JSON字符串。

将对象导出为JSON字符串

```
/// d ##class(PHA.TEST.Xml).SaveEvent()
```

```
ClassMethod SaveEvent()
{
    set event = ##class(Model.Event).%New()
    set event.Name = "Global Summit"
    set location = ##class(Model.Location).%New()
    set location.Country = "United States of America"
    set event.Location = location
    do event.%JSONExport()
}
```

此代码显示以下JSON字符串：

```
DHC-APP>d ##class(PHA.TEST.Xml).SaveEvent()
{"Name":"Global Summit","Location":{"Country":"United States of America"}}
```

可以使用%JSONExportToString()而不是%JSONExport()将JSON字符串赋给变量：

```
/// d ##class(PHA.TEST.Xml).SaveEventString()
ClassMethod SaveEventString()
{
    set event = ##class(Model.Event).%New()
    set event.Name = "Global Summit"
    set location = ##class(Model.Location).%New()
    set location.Country = "United States of America"
    set event.Location = location
    do event.%JSONExportToString(.jsonEvent)
    w jsonEvent,!
}
```

```
DHC-APP>d ##class(PHA.TEST.Xml).SaveEventString()
{"Name":"Global Summit","Location":{"Country":"United States of America"}}
```

最后，可以通过%JSONImport()将JSON字符串转换回对象。此示例从上一个示例中获取字符串变量jsonEvent，并将其转换回Model.Event对象：

将JSON字符串导入到对象中

```
/// d ##class(PHA.TEST.Xml).SaveEventStringImport()
ClassMethod SaveEventStringImport()
{
    set event = ##class(Model.Event).%New()
    set event.Name = "yx"
    set location = ##class(Model.Location).%New()
    s location.City = "tianjin"
    set location.Country = "United States of America"
    set event.Location = location
    do event.%JSONExportToString(.jsonEvent)


    set eventTwo = ##class(Model.Event).%New()
    do eventTwo.%JSONImport(jsonEvent)
    write eventTwo.Name,!,eventTwo.Location.City
```

```
}


DHC-APP>d ##class(PHA.TEST.Xml).SaveEventStringImport()
yx
tianjin
```

导入和导出都适用于任意嵌套的结构。

使用参数映射

可以通过设置相应的参数为每个单独的属性指定映射逻辑。

可以通过指定属性参数来更改Model.Event类(在上一节中定义)的映射：

```
  Class Model.Event Extends (%Persistent, %JSON.Adaptor)
  {
    Property Name As %String(%JSONFIELDNAME = "eventName");
    Property Location As Model.Location(%JSONINCLUDE = "INPUTONLY");
  }
```

此映射引入了两个更改：

- 属性名称将映射到名为eventName的JSON字段。
- Location属性仍将由%JSONImport()用作输入，但将被%JSONExport()和其他导出方法忽略。

以前，在未修改的Model.Event类的实例上调用%JSONExport()，并返回以下JSON字符串：

```
  {"Name":"Global Summit","Location":{"City":"Boston","Country":"United States of Ame
rica"}}
```

如果对重新映射的Model.Event实例调用%JSONExport()(使用相同的属性值)，将返回以下字符串：

```
DHC-APP>d ##class(PHA.TEST.Xml).SaveEvent()
{"eventName":"Global Summit"}
```

有各种参数可用于调整映射：

- %JSONFIELDNAME(仅限属性)设置要用作JSON内容中的字段名称的字符串(默认情况下，值为属性名称)。
- %JSONIGNOREINVALIDFIELD控制对JSON输入中意外字段的处理。
- %JSONIGNORENULL允许开发人员覆盖字符串属性的空字符串的默认处理。
- %JSONINCLUDE(仅限属性)指定该属性是否包含在JSON输出或输入中(有效值为"inout"(默认)，"outputonly"，"inputOnly"，或"none")。
- %JSONNULL指定了如何为字符串属性存储空字符串。
- %JSONREFERENCE指定如何将对象引用投影到JSON字段。
  选项包括OBJECT(默认值)、ID、OID和GUID。

使用XData映射块

可以在特殊的XData
mapping块中指定映射，并在调用导入或导出方法时应用映射，而不是在属性级别上设置映射参数。

```
Class Model.Event Extends (%Persistent, %JSON.Adaptor)
{

Property Name As %String;

Property Location As Model.Location;

XData OnlyLowercaseTopLevel
{
<Mapping xmlns="http://www.intersystems.com/jsonmapping">
        <Property Name="Name" FieldName="eventName"/>
        <Property Name="Location" Include="INPUTONLY"/>
      </Mapping>
}

Storage Default
{
<Data name="EventDefaultData">
<Value name="1">
<Value>%%CLASSNAME</Value>
</Value>
<Value name="2">
<Value>Name</Value>
</Value>
<Value name="3">
<Value>Location</Value>
</Value>
</Data>
<DataLocation>^Model.EventD</DataLocation>
<DefaultData>EventDefaultData</DefaultData>
<IdLocation>^Model.EventD</IdLocation>
<IndexLocation>^Model.EventI</IndexLocation>
<StreamLocation>^Model.EventS</StreamLocation>
<Type>%Storage.Persistent</Type>
}

}
```

这里有一个重要的区别:XData块中的JSON映射不会改变默认行为，但是可以通过在导入和导出方法的可选参数%mappingName中指定块名称来应用它们。
例如:

```
  do event.%JSONExport("OnlyLowercaseTopLevel")
```

```
/// d ##class(PHA.TEST.Xml).SaveEventXData()
ClassMethod SaveEventXData()
{
    set event = ##class(Model.Event).%New()
    set event.Name = "Global Summit"
    set location = ##class(Model.Location).%New()
    set location.Country = "United States of America"
    set event.Location = location
```

```
    do event.%JSONExport("OnlyLowercaseTopLevel")
}
```

显示

```
  {"eventName":"Global Summit"}
```

就像在属性定义中指定了参数一样。

如果没有具有提供名称的扩展数据块，将使用默认映射。使用这种方法，可以配置多个映射并分别引用每个调用所需的映射，从而使可以更好地控制，同时使您的映射更加灵活和可重用。

定义到扩展数据映射块

支持JSON的类可以定义任意数量的附加映射。每个映射都在以下形式的单独扩展数据块中定义：

```
  XData {MappingName}
  {
    <Mapping  {ClassAttribute}="value" [...] xmlns="http://www.intersystems.com/jsonm
apping".>
      <{Property Name}="PropertyName" {PropertyAttribute}="value" [...] />
      [... more Property elements]
    </Mapping>
  }
```

其中，{MappingName}、{ClassAttribute}、{Property Name}和{PropertyAttribute}的定义如下：
- MappingName %JSONREFERENCE参数或Reference属性使用的映射的名称。
- ClassAttribute 指定映射的类参数。可以定义以下类属性：
  - Mapping -要应用的扩展数据映射块的名称。
  - IgnoreInvalidField-指定类参数%JSONIGNOREINVALIDFIELD。
  - NULL-指定类参数%JSONNULL。
  - IgnoreNull-指定类参数%JSONIGNORENULL。
  - Reference -指定类参数%JSONREFERENCE。
- PropertyName 要映射的属性的名称。
- PropertyAttribute 指定映射的特性参数。可以定义以下特性属性：
  - FieldName-指定属性参数%JSONFIELDNAME(默认情况下与属性名称相同)。
  - Include -指定属性参数%JSONINCLUDE(有效值为"inout"(默认值)，"outputonly"， "inputOnly"，或"none")。
  - Mapping -要应用于对象属性的映射定义的名称。
  - NULL-覆盖类参数%JSONNULL。
  - IgnoreNull-覆盖类参数%JSONIGNORENULL。
  - Reference -覆盖类参数%JSONREFERENCE。

格式化JSON

%JSON.ForMatter是一个具有非常简单接口的类，允许将动态对象、数组和JSON字符串格式化为更易于阅读的表示形式。所有方法都是实例方法，因此始终从检索实例开始：

```
  set formatter = ##class(%JSON.Formatter).%New()
```

此选择背后的原因是，可以将格式化程序配置为只使用一次某些字符作为行终止符和缩进(例如，空格与制表符；请参阅本节末尾的属性列表)，然后在需要的任何地方使用它。

Format()方法接受动态实体或JSON字符串。下面是一个使用动态对象的简单示例：

```
/// d ##class(PHA.TEST.Xml).FormatterJson()
ClassMethod FormatterJson()
{
    s formatter = ##class(%JSON.Formatter).%New()
    s dynObj = {"type":"string"}
    do formatter.Format(dynObj)
}
```

```
DHC-APP>d ##class(PHA.TEST.Xml).FormatterJson()
{
  "type":"string"
}
```

Format方法可以将输出定向到当前设备、字符串或流：

- Format()使用指定的缩进格式化JSON文档并将其写入当前设备。
- FormatToStream()使用指定的缩进格式化JSON文档并将其写入流。
- FormatToString()使用指定的缩进格式化JSON文档并将其写入字符串，或者将启用JSON的类序列化为JSON文档并将其作为字符串返回。

此外，以下属性可用于控制缩进和换行符：

- Indent 缩进指定是否应缩进JSON输出
- IndentChars 指定用于每个缩进级别的字符序列(默认为每个级别一个空格)。
- LineTerminator 行终止符指定缩进时终止每行的字符序列。

#JSON #Caché #Ensemble #InterSystems IRIS

源URL: https://cn.community.intersystems.com/post/%E7%AC%AC%E5%85%AD%E7%AB%A0-cach%C3%A9-json-%E4%BD%BF%E7%94%A8json%E9%80%82%E9%85%8D%E5%99%A8