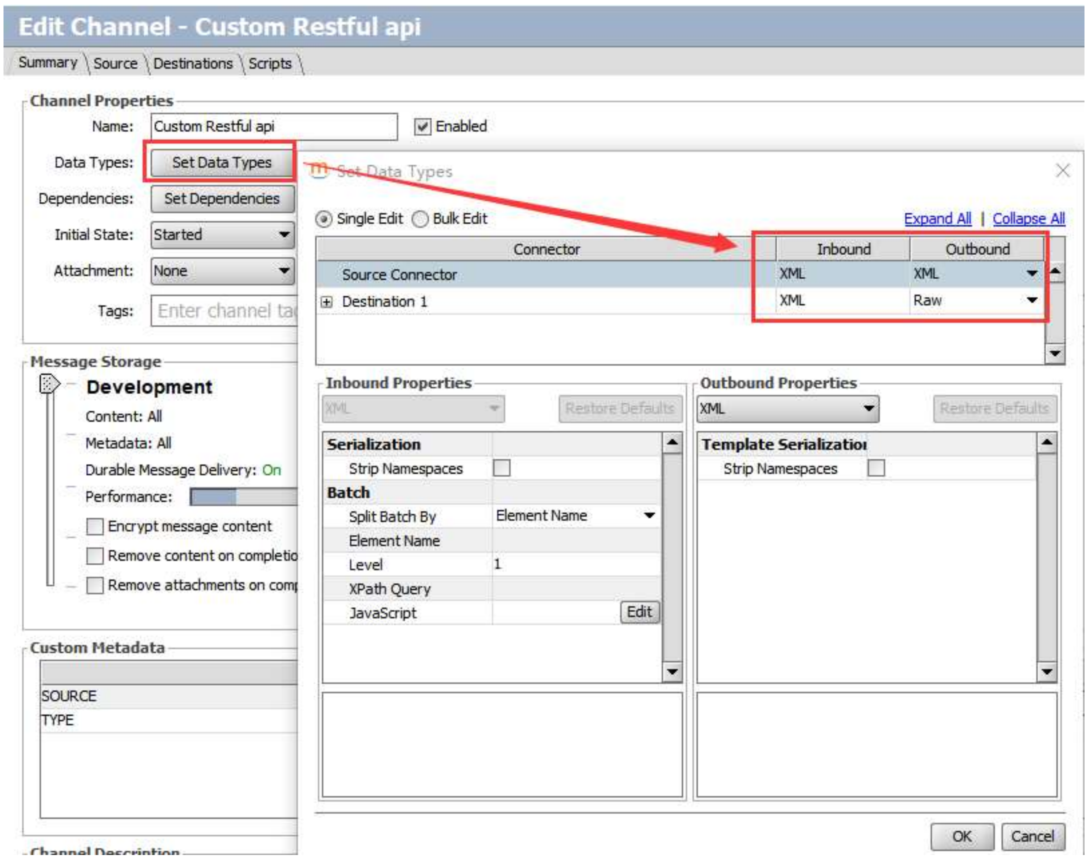
第11課-Channel Study For Create Custom Restful Service

这节课我们一起学习利用Mirth Connect的HTTP Listener源通道与JavaScript Writer目的通道搭建自定义Restful风格webapi服务。

1.新建名为'Custom Restful api'的信道,指定源通道与目的通道的输入输出消息格式


Edit Channel - Custom Restful api

Summary | Source | Destinations | Scripts

Channel Properties
Name: Custom Restful api [✓ Enabled]
Data Types: Set Data Types
Dependencies: Set Dependencies
Initial State: Started
Attachment: None
Tags: Enter channel ta

Message Storage
Development
Content: All
Metadata: All
Durable Message Delivery: On
Performance:
Encrypt message content
Remove content on completio
Remove attachments on comp

Custom Metadata
SOURCE
TYPE

Set Data Types
(● Single Edit) ○ Bulk Edit — Expand All | Collapse All

Connector	Inbound	Outbound
Source Connector	XML	XML
Destination 1	XML	Raw

Inbound Properties: XML — Restore Defaults

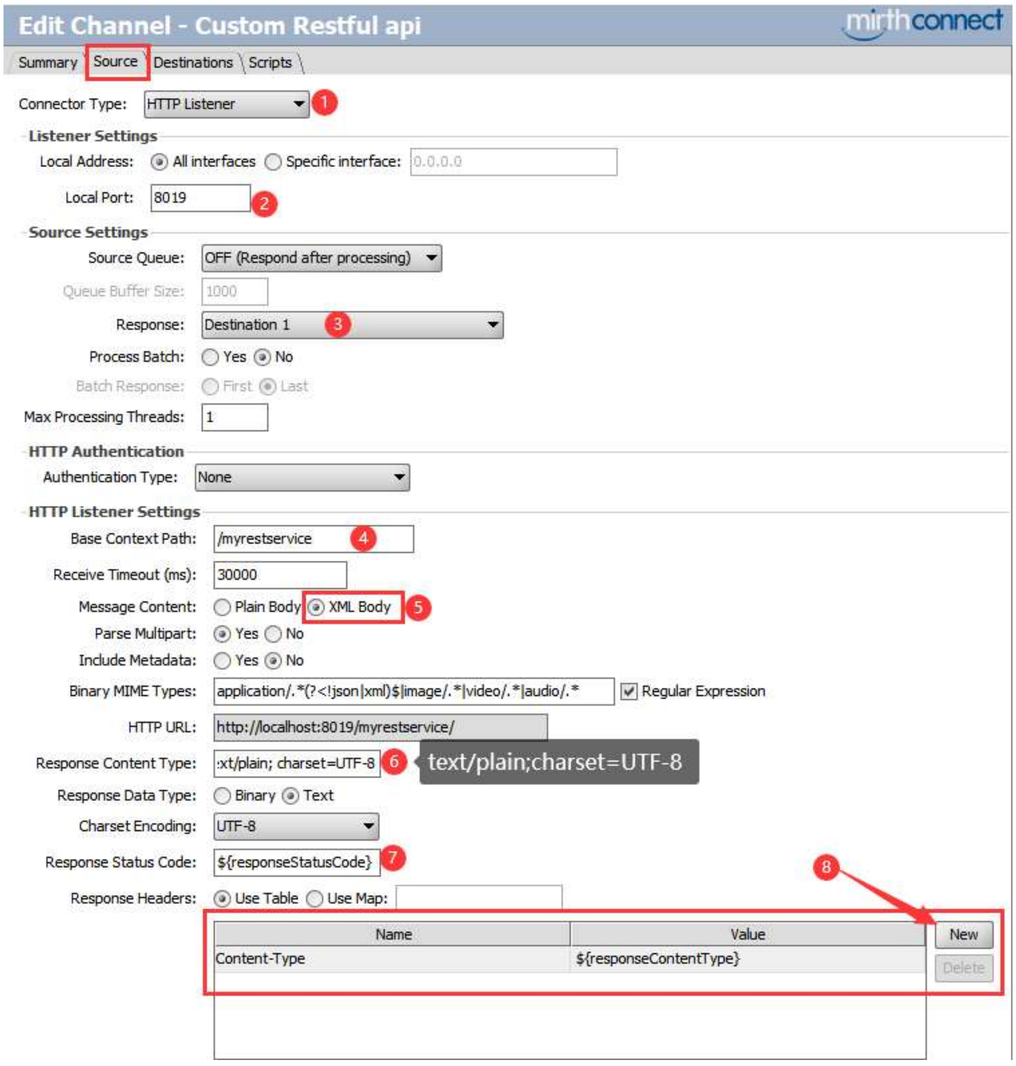
Serialization	
Strip Namespaces	☐
Batch	
Split Batch By	Element Name
Element Name	
Level	1
XPath Query	
JavaScript	Edit

Outbound Properties: XML — Restore Defaults

Template Serialization	
Strip Namespaces	☐

OK Cancel

2.设置HTTP Listener类型源通道参数



Connector Type: HTTP Listener (1)

Listener Settings

Local Address: [✓ All interfaces] Specific interface: 0.0.0.0

Local Port: 8019 (2)

Source Settings

Source Queue: OFF (Respond after processing)

Queue Buffer Size: 1000

Response: Destination 1 (3)

Process Batch: Yes [✓ No]

Batch Response: First [✓ Last]

Max Processing Threads: 1

HTTP Authentication

Authentication Type: None

HTTP Listener Settings

Base Context Path: /myrestservice (4)

Receive Timeout (ms): 30000

Message Content: Plain Body [✓ XML Body] (5)

Parse Multipart: [✓ Yes] No

Include Metadata: Yes [✓ No]

Binary MIME Types: application/.*(?<!json|xml)$|image/.*|video/.*|audio/.* [✓ Regular Expression]

HTTP URL: http://localhost:8019/myrestservice/

Response Content Type: :xt/plain; charset=UTF-8 (6) text/plain;charset=UTF-8

Response Data Type: Binary [✓ Text]

Charset Encoding: UTF-8

Response Status Code: ${responseStatusCode} (7)

Response Headers: [✓ Use Table] Use Map: (8)

Name	Value
Content-Type	${responseContentType}

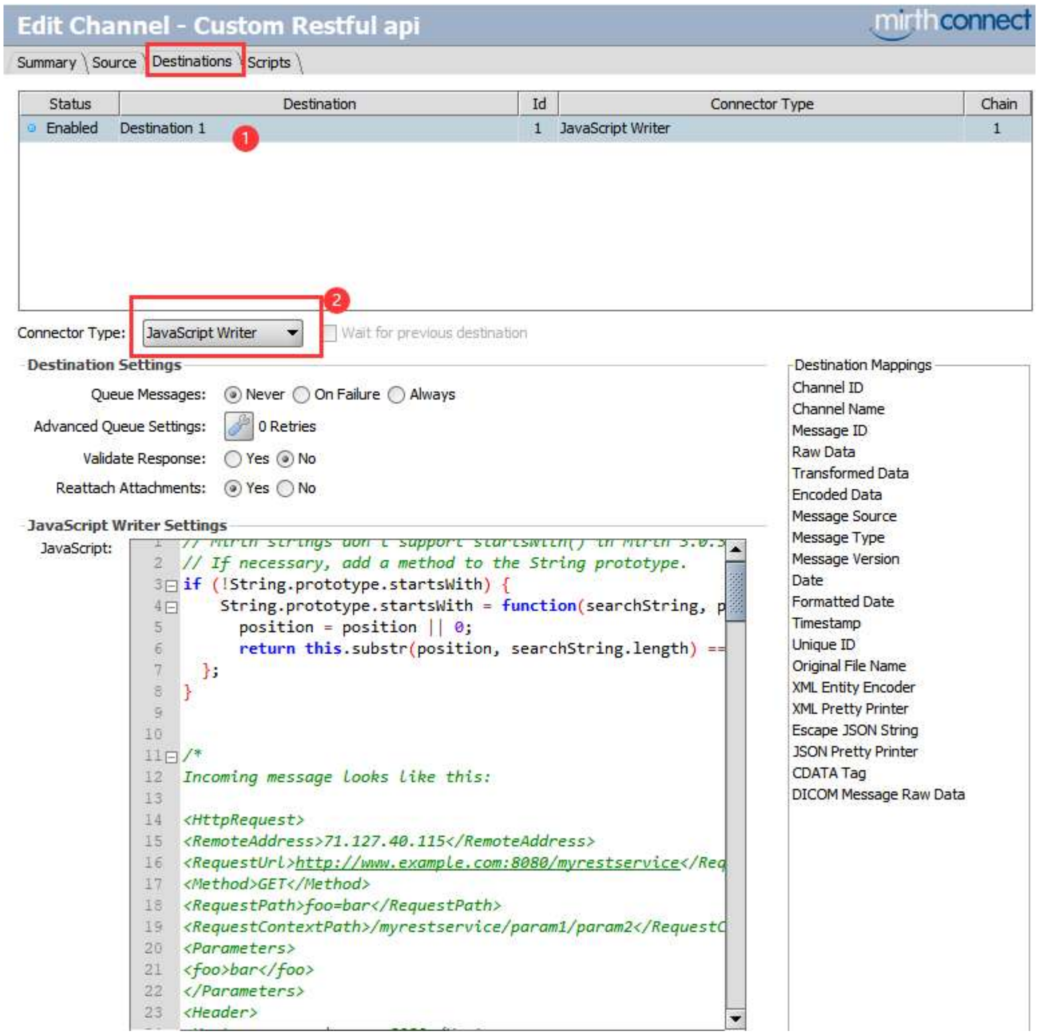
1. 把 "Response" 响应指定为 destination 1
2. 输入'Base context path' 为 /myrestservice
3. 设置 "Message Content" 为 XML Body
4. 设置默认"Response Content Type" 为text/plain; charset=UTF-8我们将在目的通道中通过channel map重写它的值为application/xml或application/json
5. 设置 "Response Status Code" 响应码为 ${responseStatusCode}我们将在目的通道中通过channel map重写它的值为200(成功)或500(失败)
6. 在 "Response Header" 中添加一个变量 "Content-Type" ，指定其值为 ${responseContentType}我们将在目的通道中通过channel map重写它的值为application/xml或application/json

3.设置JavaScript Writer目的通道参数并编写JS实现脚本

1. HL7传输协议(161)
2. 第壹課-Install：Mirth Conı 安装步骤(99)
3. 开篇:Mirth Connect系统集 5)
4. HL7标准的版本(75)
5. 第三課：信道学习Source C Destinations File Writer(60)

评论排行榜

1. 第八課-Channel Study For R Lib(1)



Status	Destination	Id	Connector Type	Chain
Enabled	Destination 1	1	JavaScript Writer	1

Connector Type: JavaScript Writer ☐ Wait for previous destination

Destination Settings

Queue Messages: [✓ Never] On Failure Always

Advanced Queue Settings: 0 Retries

Validate Response: Yes [✓ No]

Reattach Attachments: [✓ Yes] No

JavaScript Writer Settings

JavaScript:

```
// Mirth strings don't support startsWith() in Mirth 3.0.3
// If necessary, add a method to the String prototype.
if (!String.prototype.startsWith) {
    String.prototype.startsWith = function(searchString, p
      position = position || 0;
      return this.substr(position, searchString.length) ==
  };
}


/*
Incoming message looks like this:

<HttpRequest>
<RemoteAddress>71.127.40.115</RemoteAddress>
<RequestUrl>http://www.example.com:8080/myrestservice</Req
<Method>GET</Method>
<RequestPath>foo=bar</RequestPath>
<RequestContextPath>/myrestservice/param1/param2</RequestC
<Parameters>
<foo>bar</foo>
</Parameters>
<Header>
```

Destination Mappings
- Channel ID
- Channel Name
- Message ID
- Raw Data
- Transformed Data
- Encoded Data
- Message Source
- Message Type
- Message Version
- Date
- Formatted Date
- Timestamp
- Unique ID
- Original File Name
- XML Entity Encoder
- XML Pretty Printer
- Escape JSON String
- JSON Pretty Printer
- CDATA Tag
- DICOM Message Raw Data

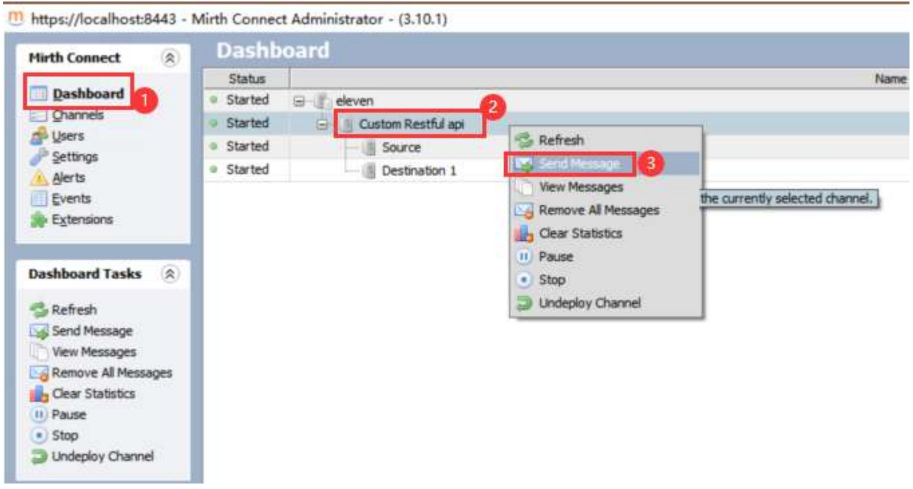
```
// Mirth strings don't support startsWith() in Mirth 3
// If necessary, add a method to the String prototype.
if (!String.prototype.startsWith) {
    String.prototype.startsWith = function(searchString, position){
      position = position || 0;
      return this.substr(position, searchString.length) === searchString;
  };
}


/*
Incoming message looks like this:

<HttpRequest>
<RemoteAddress>71.127.40.115</RemoteAddress>
<RequestUrl>http://www.example.com:8080/myrestservice</RequestUrl>
<Method>GET</Method>
<RequestPath>foo=bar</RequestPath>
<RequestContextPath>/myrestservice/param1/param2</RequestContextPath>
<Parameters>
<foo>bar</foo>
</Parameters>
<Header>
<Host>www.example.com:8080</Host>
<Accept-Encoding>identity</Accept-Encoding>
<User-Agent>Wget/1.18 (darwin15.5.0)</User-Agent>
<Connection>keep-alive</Connection>
<Accept>application/xml</Accept>
</Header>
<Content/>
</HttpRequest>

<HttpRequest>
<RemoteAddress>71.127.40.115</RemoteAddress>
<RequestUrl>http://www.example.com:8080/myrestservice</RequestUrl>
<Method>GET</Method>
<RequestPath>foo=bar</RequestPath>
```

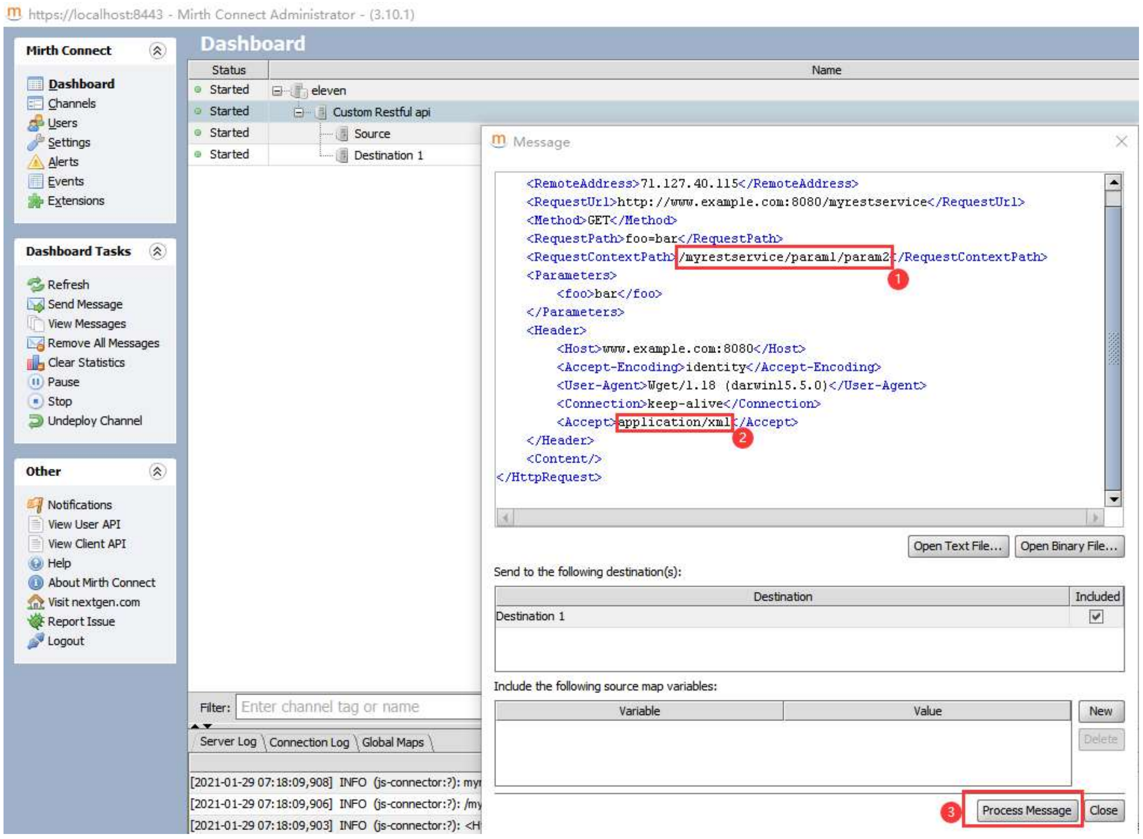
```
<RequestContextPath>/myrestservice/param1/param2</RequestContextPath>
<Parameters>
<foo>bar</foo>
</Parameters>
<Header>
<Host>www.example.com:8080</Host>
<Accept-Encoding>identity</Accept-Encoding>
<User-Agent>Wget/1.18 (darwin15.5.0)</User-Agent>
<Connection>keep-alive</Connection>
<Accept>application/json</Accept>
</Header>
<Content/>
</HttpRequest>
*/

// Just in case we fail, set a sane responseContentType
channelMap.put('responseContentType', 'text/plain');

var msg = XML(connectorMessage.getRawData());
 logger.info(msg);
// Get the REST data from the "context path" which is actually
// the "path info" of the request, so it will start with '/myrestservice'.
var rest = msg['RequestContextPath'];
logger.info(rest);
var myServicePrefix = '/myrestservice';
var minimumURLParameterCount = 4; // This is the minimum you require to do your work
var maximumExpectedURLParameterCount = 5; // however many you expect to get
var params = rest.substring(myServicePrefix).split('/', maximumExpectedURLParameterCount);
if(params.length < minimumURLParameterCount)
 return Packages.com.mirth.connect.server.userutil.ResponseFactory.getErrorResponse('Too few parameters in request');
var mrn = params[1]; // params[0] will be an empty string
logger.info(mrn);
// Now, determine the client's preference for what data type to return (XML vs. JSON).
// We will default to XML.
var clientWantsJSON = false;
var responseContentType = 'text/xml';

// If we see any kind of JSON before any kind of XML, we'll use
// JSON. Otherwise, we'll use XML.
//
// Technically, this is incorrect resolution of the "Accept" header,
// but it's good enough for an example.
var mimeTypes = msg['Header']['Accept'].split(/\s*,\s*/);
for(var i=0; i<mimeTypes.length; ++i) {
  var mimeType = mimeTypes[i].toString();
  if(mimeType.startsWith('application/json')) {
    clientWantsJSON = true;
    responseContentType = 'application/json';
    break;
  } else if(mimeType.startsWith('application/xml')) {
    clientWantsJSON = false;
    responseContentType = 'application/xml';
    break;
  } else if(mimeType.startsWith('text/xml')) {
    clientWantsJSON = false;
    responseContentType = 'text/xml';
    break;
  }
}

var xml;
var json;

if(clientWantsJSON)
  json = { status : '' };
else
  xml = new XML('<response></response>');

try {
    /*
      Here is where you do whatever your service needs to actually do.
     */

  if(clientWantsJSON) {
     json.data = { foo: 1,
                   bar: 'a string',
                   baz: [ 'list', 'of', 'strings']
                 };
    } else {
      xml['@foo'] = 1;
      xml['bar'] = 'a string';
      xml['baz'][0] = 'list';
      xml['baz'][1] = 'of';
      xml['baz'][3] = 'strings';
    }

    // Set the response code and content-type appropriately.
    // http://www.mirthproject.org/community/forums/showthread.php?t=12678

    channelMap.put('responseStatusCode', 200);

    if(clientWantsJSON) {
      json.status = 'success';
      var content = JSON.stringify(json);
      channelMap.put('responseContent', content);
      channelMap.put('responseContentType', responseContentType);
      return content;
    } else {
      channelMap.put('responseContentType', responseContentType);
      var content = xml.toString();
      channelMap.put('responseContent', content);
      return content;
    }
  }
  catch (err)
  {
    channelMap.put('responseStatusCode', '500');
    if(clientWantsJSON) {
      json.status = 'error';
      if(err.javaException) {
        // If you want to unpack a Java exception, this is how you do it:
        json.errorType = String(err.javaException.getClass().getName());
        json.errorMessage = String(err.javaException.getMessage());
      }

      channelMap.put('responseContentType', responseContentType);

      // Return an error with our "error" JSON
      return Packages.com.mirth.connect.server.userutil.ResponseFactory.getErrorResponse(JSON.stringify(json));
    } else {
      if(err.javaException) {
        xml['response']['error']['@type'] = String(err.javaException.getClass().getName());
        xml['response']['error']['@message'] = String(err.javaException.getMessage());
      }

      channelMap.put('responseContentType', responseContentType);

      // Return an error with our "error" XML
      return Packages.com.mirth.connect.server.userutil.ResponseFactory.getErrorResponse(xml.toString());
    }
  }
```

我们通过目的通道以上JS脚本，学习到以下特别重要的知识：

1. 获取输入请求的原始消息并自动格式化为XML格式： `var xml = new XML(connectorMessage.getRawData())`
2. 设置响应类型，如：`channelMap.put('responseContentType', 'application/json')`
3. 设置响应码，如：`channelMap.put('responseStatusCode', '200')`
4. 设置响应内容并通过JS脚本返回XML实体或者Json实体的字符串格式值
5. 异常处理通过JS脚本调用Mirth的API函数Packages.com.mirth.connect.server.userutil.ResponseFactory.getErrorResponse(string)返回字符串格式错误消息

4.部署信道并测试





```
    <RemoteAddress>71.127.40.115</RemoteAddress>
    <RequestUrl>http://www.example.com:8080/myrestservice</RequestUrl>
    <Method>GET</Method>
    <RequestPath>foo=bar</RequestPath>
    <RequestContextPath>/myrestservice/param1/param2</RequestContextPath>
    <Parameters>
        <foo>bar</foo>
    </Parameters>
    <Header>
        <Host>www.example.com:8080</Host>
        <Accept-Encoding>identity</Accept-Encoding>
        <User-Agent>Wget/1.18 (darwin15.5.0)</User-Agent>
        <Connection>keep-alive</Connection>
        <Accept>application/xml</Accept>
    </Header>
    <Content/>
</HttpRequest>
```

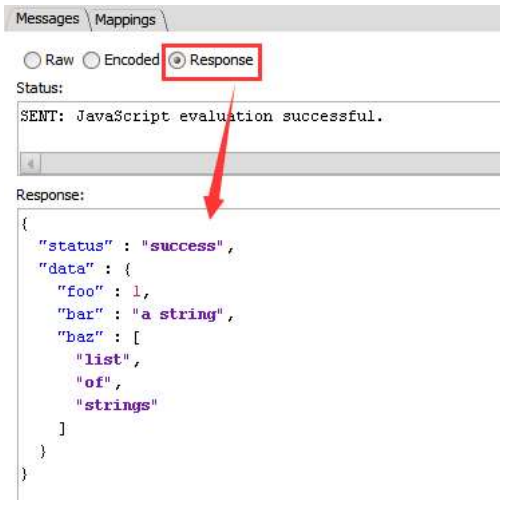
Destination	Included
Destination 1	✓

Variable	Value

```
[2021-01-29 07:18:09,908] INFO (js-connector:?): my
[2021-01-29 07:18:09,906] INFO (js-connector:?): /my
[2021-01-29 07:18:09,903] INFO (js-connector:?): <H
```

发送消息要区分application/json和application/xml，可以看到响应值格式会相应变化

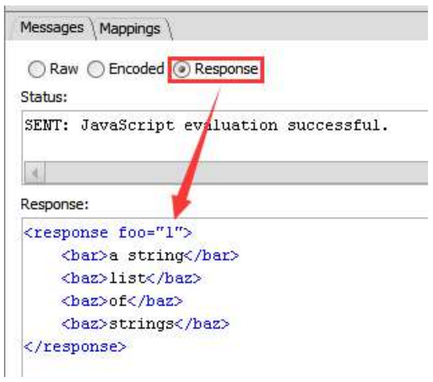
```
<HttpRequest>
    <RemoteAddress>71.127.40.115</RemoteAddress>
    <RequestUrl>http://www.example.com:8080/myrestservice</RequestUrl>
    <Method>GET</Method>
    <RequestPath>foo=bar</RequestPath>
    <RequestContextPath>/myrestservice/param1/param2</RequestContextPath>
    <Parameters>
        <foo>bar</foo>
    </Parameters>
    <Header>
        <Host>www.example.com:8080</Host>
        <Accept-Encoding>identity</Accept-Encoding>
        <User-Agent>Wget/1.18 (darwin15.5.0)</User-Agent>
        <Connection>keep-alive</Connection>
        <Accept>application/json</Accept>
    </Header>
    <Content/>
</HttpRequest>
```



```
SENT: JavaScript evaluation successful.
```

```
{
  "status" : "success",
  "data" : {
    "foo" : 1,
    "bar" : "a string",
    "baz" : [
      "list",
      "of",
      "strings"
    ]
  }
}
```

```xml
<HttpRequest>
    <RemoteAddress>71.127.40.115</RemoteAddress>
    <RequestUrl>http://www.example.com:8080/myrestservice</RequestUrl>
    <Method>GET</Method>
    <RequestPath>foo=bar</RequestPath>
    <RequestContextPath>/myrestservice/param1/param2</RequestContextPath>
    <Parameters>
        <foo>bar</foo>
    </Parameters>
    <Header>
        <Host>www.example.com:8080</Host>
        <Accept-Encoding>identity</Accept-Encoding>
        <User-Agent>Wget/1.18 (darwin15.5.0)</User-Agent>
        <Connection>keep-alive</Connection>
        <Accept>application/xml</Accept>
    </Header>
    <Content/>
</HttpRequest>
```



```
SENT: JavaScript evaluation successful.
```

```xml
<response foo="1">
    <bar>a string</bar>
    <baz>list</baz>
    <baz>of</baz>
    <baz>strings</baz>
</response>
```

大功告成！！！

本课程总结：

通过JS脚本编程，自定义实现Restful风格webapi.

欢迎大家持续关注潤沁網路大學本系列Mirth Connect课程的教学

潤沁網路大學

分类: Mirth

标签: Mirth

好文要顶 关注我 收藏该文

潤沁網路大學
关注 - 0
粉丝 - 2

0 0

« 上一篇： 第十課-Change Mirth Connect To Mariadb
» 下一篇： What is an HL7 ADT Message?

posted @ 2021-01-29 19:59 潤沁網路大學 阅读(29) 评论(0) 编辑 收藏

刷新评论 刷新页面 返回顶部

发表评论

编辑 预览

支持 Markdown

自动补全

提交评论 退出

[Ctrl+Enter快捷键提交]

园子动态:
· 发起一个开源项目：博客引擎 fluss
· 云计算之路-新篇章-出海记：开篇
· 博客园2005年6月1日首页截图

最新新闻:
· 黄峥勇退：一年之内卸任CEO和董事长 想去"寻找幸福"
· 百度二次上市，三重价值
· 快手三年游戏路，路在何处?
· 谷歌涂鸦庆祝爱尔兰圣帕特里克节
· NASA的SMA轮胎技术即将商用 30倍于钢的可恢复应变
» 更多新闻...

潤沁網路大學