RTTI, JSON, und WebSockets

Ein Vortrag von Simon Ameis
auf der Lazaruskonferenz 2021 in Oldenburg

RTTI

- Run-Time Type Information
- Informationen über Datentypen, Klassen, Methoden etc. zur Laufzeit
- Werden vom Compiler generiert
- Seit FPC 3.2.0 „experimental extended RTTI“ für COM-Interfaces

Methoden auflisten

```
procedure ListMethods(aTypeInfo: PTypeInfo);
var
  T: TRttiType;
  InterfaceType: TRttiInterfaceType;
  Methods: specialize TArray<TRttiMethod>;
  m: TRttiMethod;
begin
  T := RTTIContext.GetType(aTypeInfo);
  if not (T is TRttiInterfaceType) then exit;

  InterfaceType := TRttiInterfaceType(T);

  Methods := InterfaceType.GetMethods;
  for m in Methods do
  begin
    WriteLn('Methode gefunden:');
    WriteLn('Unit      :', InterfaceType.DeclaringUnitName);
    WriteLn('Interface:', InterfaceType.Name);
    WriteLn('Methode   :', m.Name);
  end;
end;
```

JSON

- JavaScript Object Notation
- Unit fpJSON

```
{
  "method":"Unit.Interface.Method",
  "params":["a", 5]
}

uses
  fpJSON;
var
  Data: TJSONData;
begin
   Data := GetJSON('{"method": "Unit.Interface.Method"}');
   // Data as TJSONObject
   // ...
end;
```

WebSocket

- Bidirektionale asynchrone Verbindung
- Server kann auch unaufgefordert Daten senden
- Protokollstart abwärtskompatibel zu HTTP(S)
- Package: LazWebsockets

WebSocket Server

```
program websocketserver;

{$mode objfpc}{$H+}

uses {$IFDEF UNIX} cMem, cthreads, {$ENDIF}
  Classes, SysUtils, WebSocketServer,
var
  socket: TWebSocketServer;
begin
  socket := TWebSocketServer.Create(8080);
  try
    socket.FreeHandlers := True;
    socket.AcceptingMethod := samThreaded;
    socket.RegisterHandler('*', '*', TThreadedWebsocketHandler.Create, True, True);
    socket.Start;
  finally
    socket.Free;
  end;
end.
```

WebSocket Client

```
var
  RPC: TJSWebSocket;
begin
  RPC := TJSWebSocket.new('wss://myserver:8080/api');
  RPC.send('Hallo Welt');
  RPC.send('{ "method":"Unit.Interface.Method", "params":["a", 5] }');
end;
```

WebSocket

Was kommt heraus, wenn man RTTI, JSON und WebSocket kombiniert?

Remote Procedure Call

RTTI – Interface definieren

```
{$Interfaces COM}

type
  IMyInterface = Interface(IUnknown)
    procedure Method1(a: Int64; s: String);
    procedure Method2(Hello: Int64; World: String);
  end;

  TMyInterfaceImplementingClass = class(TInterfacedObject, IMyInterface)
  published
    procedure Method1(a: Int64; s: String);
    procedure Method2(Hello: Int64; World: String);
  end;

  {$M+}
  TMyOtherInterfaceImplementingClass = class(TObject, IMyInterface)
  published
    procedure Method1(a: Int64; s: String);
    procedure Method2(Hello: Int64; World: String);
    { Methoden von IUnknown (Windows) }
    function QueryInterface(constref iid : tguid;out obj) : longint; stdcall;
    function _AddRef : longint;stdcall;
    function _Release : longint;stdcall;
  end;
```

RTTI – Methoden aufrufen

```
var
  MyInstance: TMyInterfaceImplementingClass;
  Values: TValueArray;
  Context: TRttiContext;
  InterfaceType: TRttiInterfaceType;
  Methods: specialize TArray<TRttiMethod>;
  Method: TRttiMethod;
begin
  MyInstance := TMyInterfaceImplementingClass.Create;
  // Alle published Methoden von MyInstance haben zwei Parameter:
  SetLength(Values, 2);
  Values[0] := MyInstance;  // impliziter Self-Parameter
  Values[1] := Int64(5);    // Int64-Paramter

  // RTTI zum Interface holen
  Context := TRTTIContext.Create;
  InterfaceType := TRttiInterfaceType(Context.GetType(TypeInfo(IMyInterface))).;

  // Methoden aufrufen
  Methods := InterfaceType.GetMethods;
  for Method in Methods do
    Method.Invoke(InvokeValue, Values);
end;
```

Remote-Aufruf im Webbrowser

```
{$mode objfpc}
{$ModeSwitch typehelpers}
{$modeswitch externalclass}

interface

uses
  sysutils, JS, web;

type
TRemoteCall = class (TJSObject)
private
  fMethod: TJSString external name 'method';
  fParams: TJSArray external name 'params';
public
  property Method: TJSString read fMethod;
  property Params: TJSArray read fParams;
end;

TRPCSocket = class helper for TJSWebSocket
public
  procedure RemoteCall(call: TRemoteCall);
end;
```

```
{
  "method":"Unit.Interface.Method",
  "params":["a", 5]
}
```

```
procedure TRPCSocket.RemoteCall(
  call: TRemoteCall);
var
  MessageStr: String;
begin
  MessageStr := TJSJSON.stringify(call);
  Self.send(MessageStr);
end;
```

Remote-Aufruf im Webbrowser

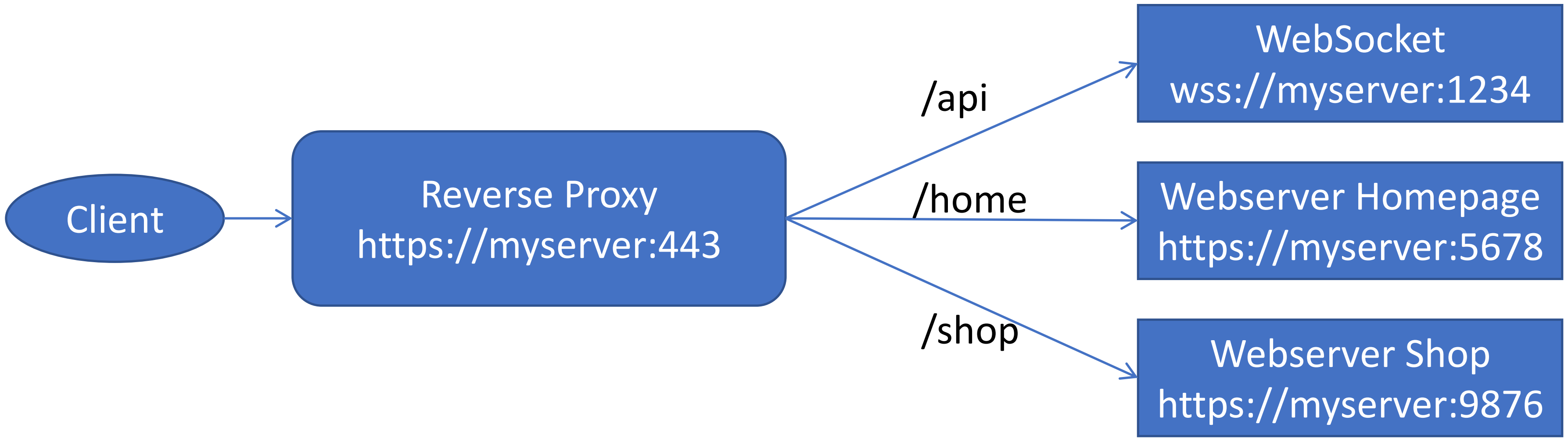
```
IProxy_ILoginHandler = interface
['{7084A201-BF1D-4A14-95E7-FEB59481A299}']
  procedure Login( const Username: AnsiString;
                   const Method: AnsiString;
                   const AuthToken: AnsiString);
end;

TProxy_ILoginHandler = class(TObject, IProxy_ILoginHandler)
  procedure Login( const Username: AnsiString;
                   const Method: AnsiString;
                   const AuthToken: AnsiString);
end;

var
  ILoginHandler: IProxy_ILoginHandler;
```

Remote-Aufruf im Webbrowser

```
procedure Tproxy_IloginHandler.Login(
  const Username: AnsiString;
  const Method: AnsiString;
  const AuthToken: AnsiString);
var
  Call: TRemoteCall;
  Params: TJSArray;
  i: NativeInt;
begin
  Params := TJSArray.new;
  for i := 0 to JSArguments.Length - 1 do
    Params.push(JSArguments.Elements[i]);
  Call := TRemoteCall.new('ILoginHandler.Login', Params);
  RPC.RemoteCall(Call);
end;
```

Fragen?

Ansprechen mehrerer Server


Client
Reverse Proxy
https://myserver:443
/api
/home
/shop
WebSocket
wss://myserver:1234
Webserver Homepage
https://myserver:5678
Webserver Shop
https://myserver:9876