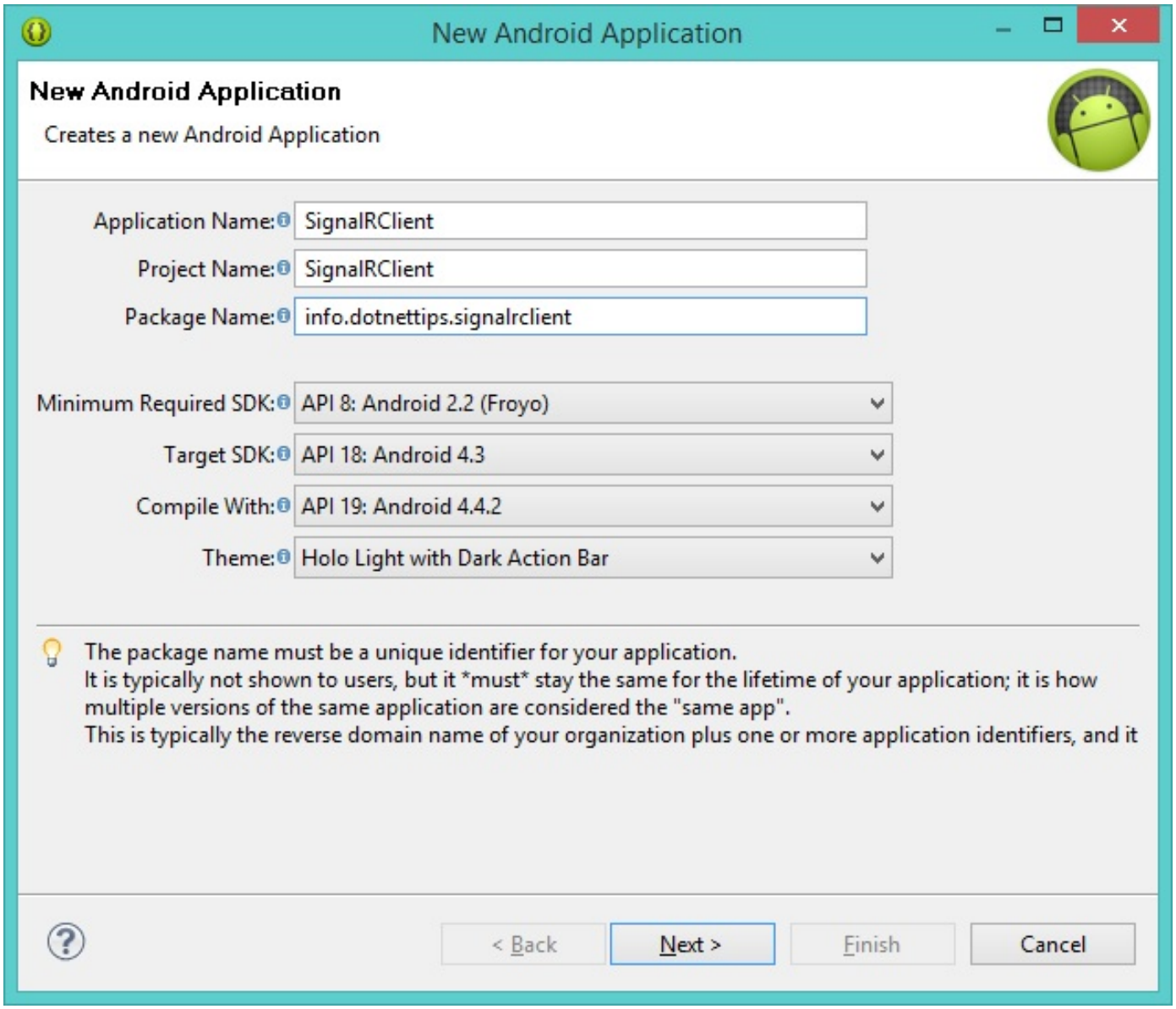
عنوان: **استفاده از SignalR در اندروید**
نویسنده: سیروان عفیفی
تاریخ: ۱۳۹۳/۰۷/۲۱ ۱۳:۲۵
آدرس: www.dotnettips.info
گروه‌ها: SignalR, Android, Java

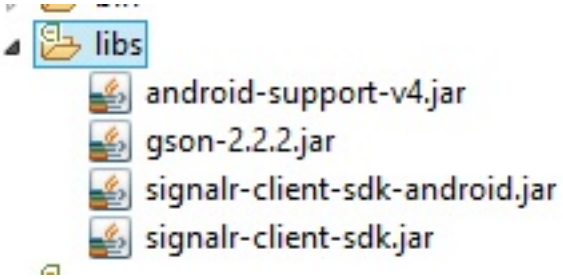
همانطور که مطلع هستید، بخش سورس باز مایکروسافت برای برنامه‌نویس‌های جاوا نیز SDK ی جهت استفاده از SignalR ارائه کرده است. در اینجا می‌توانید مخزن کد آن را در گیت‌هاب مشاهده کنید. هنوز مستنداتی برای این SDK به صورت قدم به قدم ارائه نشده است. لازم به ذکر است که مراجعه به قسمت‌های نوشته شده در اینجا نیز می‌تواند منبع خوبی برای شروع باشد. در ادامه نحوه استفاده از این SDK را با هم بررسی خواهیم کرد.
ابتدا در سمت سرور یک Hub ساده را به صورت زیر تعریف می‌کنیم:

```
public class ChatHub : Hub
{
        public void Send(string name, string message)
        {
            Clients.All.messageReceived(name, message);
        }
}
```

برای سمت کلاینت نیز یک پروژه Android Application داخل Eclipse به صورت زیر ایجاد می‌کنیم:



خوب، برای استفاده از SignalR در پروژه‌ی ایجاد شده باید کتابخانه‌های زیر را به درون پوشه libs اضافه کنیم، همچنین باید ارجاعی به کتابخانه Gson نیز داشته باشیم.



قدم بعدی افزودن کدهای سمت کلاینت برای SignalR می‌باشد. دقت داشته باشید که کدهایی که در ادامه مشاهده خواهید کرد دقیقاً مطابق دستورالمعل‌هایی است که قبلاً مشاهده کرده‌اید. برای اینکار داخل کلاس MainActivity.java کدهای زیر را اضافه کنید:

```
Platform.loadPlatformComponent( new AndroidPlatformComponent() );
HubConnection connection = new HubConnection(DEFAULT_SERVER_URL);
HubProxy hub = connection.createHubProxy("ChatHub");
connection.error(new ErrorCallback() {

    @Override
    public void onError(final Throwable error) {
        runOnUiThread(new Runnable() {
            public void run() {
                Toast.makeText(getApplicationContext(), error.getMessage(), Toast.LENGTH_LONG).show();
            }
        });

    }
});
hub.subscribe(new Object() {
    @SuppressWarnings("unused")
    public void messageReceived(final String name, final String message) {

        runOnUiThread(new Runnable() {
            public void run() {
                Toast.makeText(getApplicationContext(), name + ": " + message,
Toast.LENGTH_LONG).show();
            }
        });
    }
});
connection.start()
.done(new Action<Void>() {

    @Override
    public void run(Void obj) throws Exception {
        runOnUiThread(new Runnable() {
            public void run() {
                Toast.makeText(getApplicationContext(), "Done Connecting!", Toast.LENGTH_LONG).show();
            }
        });
    }
});
connection.received(new MessageReceivedHandler() {
    @Override
    public void onMessageReceived(final JsonElement json) {
        runOnUiThread(new Runnable() {
            public void run() {
                JsonObject jsonObject = json.getAsJsonObject();
                JsonArray jsonArray =jsonObject.getAsJsonArray("A");
                Toast.makeText(getApplicationContext(), jsonArray.get(0).getAsString() + ": " +
jsonArray.get(1).getAsString(), Toast.LENGTH_LONG).show();
            }
        });
```

```
    }
});
```

همانطور که مشاهده می‌کنید توسط قطعه کد زیر SKD مربوطه در نسخه‌های قدیمی اندروید نیز بدون مشکل کار خواهد کرد:

```
Platform.loadPlatformComponent( new AndroidPlatformComponent() );
```

در ادامه توسط متد createHubProxy ارجاعی به هابی که در سمت سرور ایجاد کردیم، داده‌ایم:

```
HubProxy hub = connection.createHubProxy("ChatHub");
```

در ادامه نیز توسط یک روال رویدادگردان وضعیت اتصال را چک کرده‌ایم. یعنی در زمان بروز خطا در نحوه ارتباط یک پیام بر روی صفحه نمایش داده می‌شود:

```
connection.error(new ErrorCallback() {

    @Override
    public void onError(final Throwable error) {
        runOnUiThread(new Runnable() {
            public void run() {
                Toast.makeText(getApplicationContext(), error.getMessage(), Toast.LENGTH_LONG).show();
            }
        });

    }
});
```

در ادامه نیز توسط کد زیر متد پویایی که در سمت سرور ایجاد کرده بودیم را جهت برقراری ارتباط با سرور اضافه کرده‌ایم:

```
hub.subscribe(new Object() {
    @SuppressWarnings("unused")
    public void messageReceived(final String name, final String message) {

        runOnUiThread(new Runnable() {
            public void run() {
                Toast.makeText(getApplicationContext(), name + ": " + message,
Toast.LENGTH_LONG).show();
            }
        });
    }
});
```

برای برقراری ارتباط نیز کدهای زیر را اضافه کرده‌ایم. یعنی به محض اینکه با موفقیت اتصال با سرور برقرار شد پیامی بر روی صفحه‌نمایش ظاهر می‌شود:

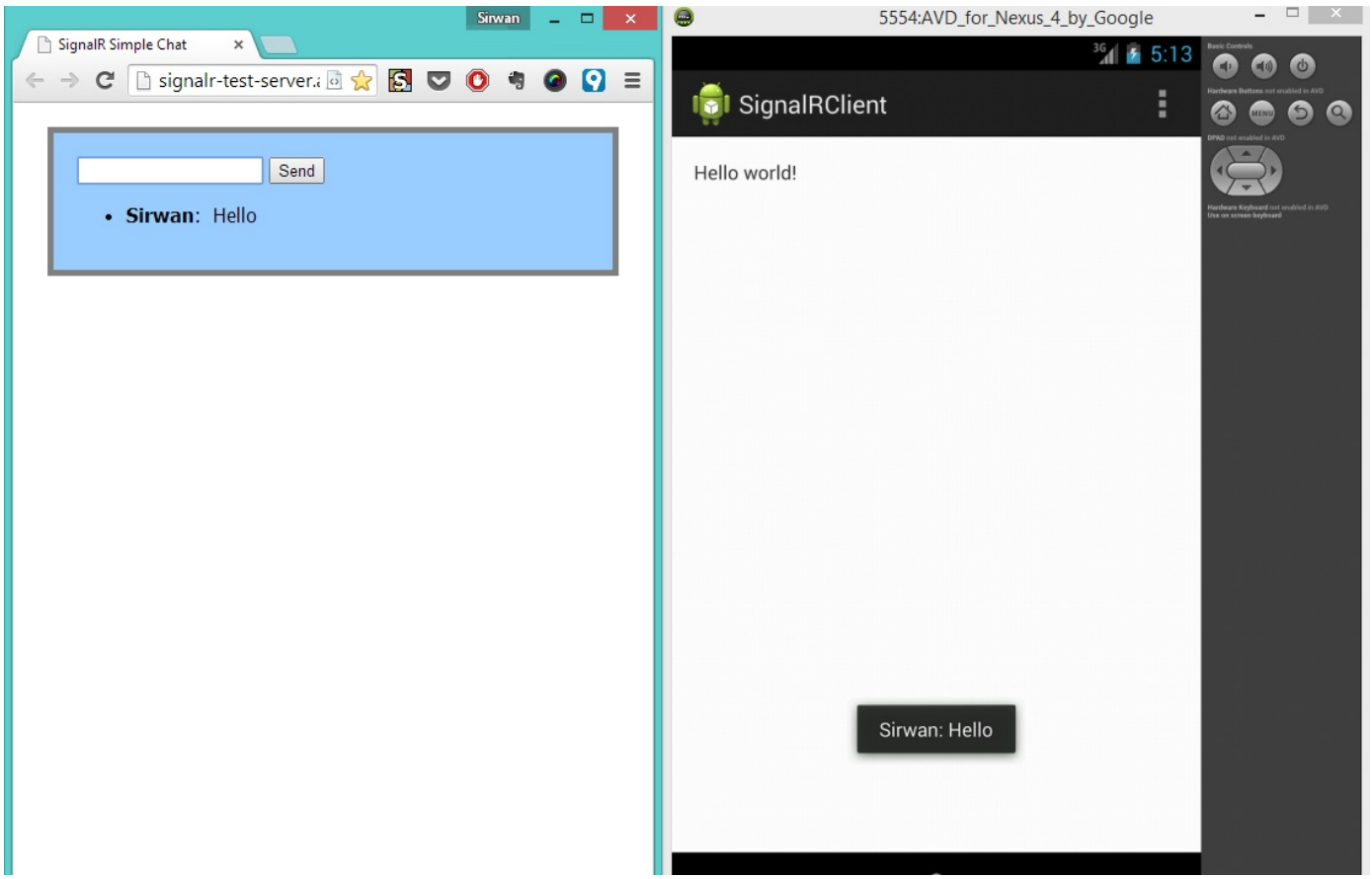
```
connection.start()
.done(new Action<Void>() {

    @Override
    public void run(Void obj) throws Exception {
        runOnUiThread(new Runnable() {
            public void run() {
                Toast.makeText(getApplicationContext(), "Done Connecting!", Toast.LENGTH_LONG).show();
            }
        });
    }
});
```

در نهایت نیز برای نمایش اطلاعات دریافت شده کد زیر را نوشته‌ایم:

```
connection.received(new MessageReceivedHandler() {
    @Override
    public void onMessageReceived(final JsonElement json) {
```

```
        runOnUiThread(new Runnable() {
            public void run() {
                JsonObject jsonObject = json.getAsJsonObject();
                JsonArray jsonArray =jsonObject.getAsJsonArray("A");
                Toast.makeText(getApplicationContext(), jsonArray.get(0).getAsString() + ": " +
jsonArray.get(1).getAsString(), Toast.LENGTH_LONG).show();
            }
        });
    }
});
```

همانطور که عنوان شد کدهای فوق دقیقاً براساس قواعد و دستورالعمل استفاده از SignalR در سمت کلاینت می‌باشد.



نظرات خوانندگان

نویسنده: رشیدیان
تاریخ: ۱۳۹۳/۰۷/۲۱ ۱۵:۴۶

ممنون - بسیار عالی
یک سئوال: آیا از این طریق میشه به همون قابلیت‌های Push Notification در GCM دست یافت؟
و اینکه چقدر این روش قابل اتکا هست؟

نویسنده: سیروان عفیفی
تاریخ: ۱۳۹۳/۰۷/۲۱ ۱۶:۵۱

دقیقاً یکی از استفاده‌هایی که برای خودم داره بحث Push Notification و ارسال پیام به کاربران متصل هست.

عنوان: **کار با وب سرویس جاوایی تشخیص ایمیل‌های موقتی در دات نت**
نویسنده: وحید نصیری
تاریخ: ۱۳۹۳/۱۰/۲۷ ۱۲:۴۰
آدرس: www.dotnettips.info
گروه‌ها: ASP.Net, MVC, Security, WCF, xml, Validation, Java

یکی از وب سرویس‌های سایت name api ، امکان تشخیص موقتی بودن ایمیل مورد استفاده‌ی جهت ثبت نام در یک سایت را فراهم می‌کند. آدرس WSDL آن نیز در اینجا قرار دارد. اگر مطابق معمول استفاده از سرویس‌های وب در دات نت، بر روی ارجاعات پروژه کلیک راست کرده و گزینه‌ی Add service refrence را انتخاب کنیم و سپس آدرس WSDL یاد شده را به آن معرفی کنیم، بدون مشکل ساختار این وب سرویس دریافت و برای استفاده‌ی از آن به یک چنین کدی خواهیم رسید:

```
var client = new SoapDisposableEmailAddressDetectorClient();

var context = new soapContext
{
    //todo: get your API key here: http://www.nameapi.org/en/register/
    apiKey = "test"
};
var result = client.isDisposable(context, "DaDiDoo@mailinator.com");

if (result.disposable.ToString() == "YES")
{
    Console.WriteLine("YES! It's Disposable!");
}
```

متد isDisposable ارائه شده‌ی توسط این وب سرویس، دو پارامتر context که در آن باید API Key خود را مشخص کرد و همچنین آدرس ایمیل مورد بررسی را دریافت می‌کند. اگر به همین ترتیب این پروژه را اجرا کنید، با خطای Bad request از طرف سرور متوقف خواهید شد:

```
Additional information: The remote server returned an unexpected response: (400) Bad Request.
```

اگر به خروجی این وب سرویس در فیدلر مراجعه کنیم، چنین شکلی را خواهد داشت:

```
<html><head><title>Bad Request</title></head><body><h1>Bad Request</h1><p>No api-key
provided!</p></body></html>
```

عنوان کرده‌است که api-key را، در درخواست وب خود ذکر نکرده‌ایم.
اگر همین وب سرویس را توسط امکانات سایت http://wsdlbrowser.com بررسی کنید، بدون مشکل کار می‌کند. اما تفاوت در کجاست؟
خروجی ارسالی به سرور، توسط سایت http://wsdlbrowser.com به این شکل است:

```
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:ns1="http://disposableemailaddressdetector.email.services.v4_0.soap.server.nameapi.org/">
  <SOAP-ENV:Body>
    <ns1:isDisposable>
      <context>
        <apiKey>test</apiKey>
      </context>
      <emailAddress>sdsdg@site.com</emailAddress>
    </ns1:isDisposable>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

و نمونه‌ی تولید شده‌ی توسط WCF (امکان Add service reference در حقیقت یک WCF Client را ایجاد می‌کند) به صورت زیر می‌باشد:

```
<s:Envelope xmlns:s="http://schemas.xmlsoap.org/soap/envelope/">
  <s:Body xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema">
     <isDisposable
```

```
xmlns="http://disposableemailaddressdetector.email.services.v4_0.soap.server.nameapi.org/">
          <context xmlns=""><apiKey>test</apiKey></context>
          <emailAddress xmlns="">DaDiDoo@mailinator.com</emailAddress>
      </isDisposable>
  </s:Body>
</s:Envelope>
```

از لحاظ اصول XML، خروجی تولیدی توسط WCF هیچ ایرادی ندارد. از این جهت که نام فضای نام مرتبط با http://schemas.xmlsoap.org/soap/envelope/ را به s تنظیم کرده‌است و سپس با استفاده از این نام، Envelope را تشکیل داده‌است. اما ... این وب سرور جاوایی دقیقا با نام SOAP-ENV کار می‌کند و فضای نام ns1 بعدی آن. کاری هم به اصول XML ندارد که باید بر اساس نام xmlns ذکر شده، کار Parse ورودی دریافتی صورت گیرد و نه بر اساس یک رشته‌ی ثابت از پیش تعیین شده. بنابراین باید راهی را پیدا کنیم تا بتوان این s را تبدیل به SOAP-ENV کرد.

برای این منظور به سه کلاس ذیل خواهیم رسید:

```
public class EndpointBehavior : IEndpointBehavior
{
    public void AddBindingParameters(ServiceEndpoint endpoint, BindingParameterCollection
bindingParameters)
    { }

    public void ApplyDispatchBehavior(ServiceEndpoint endpoint, EndpointDispatcher endpointDispatcher)
    { }

    public void Validate(ServiceEndpoint endpoint)
    { }

    public void ApplyClientBehavior(ServiceEndpoint endpoint, ClientRuntime clientRuntime)
    {
        clientRuntime.MessageInspectors.Add(new ClientMessageInspector());
    }
}

public class ClientMessageInspector : IClientMessageInspector
{
    public void AfterReceiveReply(ref Message reply, object correlationState)
    { }

    public object BeforeSendRequest(ref Message request, System.ServiceModel.IClientChannel channel)
    {
        request = new MyCustomMessage(request);
        return request;
    }
}

/// <summary>
/// To customize WCF envelope and namespace prefix
/// </summary>
public class MyCustomMessage : Message
{
    private readonly Message _message;

    public MyCustomMessage(Message message)
    {
        _message = message;
    }

    public override MessageHeaders Headers
    {
        get { return _message.Headers; }
    }

    public override MessageProperties Properties
    {
        get { return _message.Properties; }
    }

    public override MessageVersion Version
    {
        get { return _message.Version; }
    }

    protected override void OnWriteStartBody(XmlDictionaryWriter writer)
    {
```

```
        writer.WriteStartElement("Body", "http://schemas.xmlsoap.org/soap/envelope/");
    }

    protected override void OnWriteBodyContents(XmlDictionaryWriter writer)
    {
        _message.WriteBodyContents(writer);
    }

    protected override void OnWriteStartEnvelope(XmlDictionaryWriter writer)
    {
        writer.WriteStartElement("SOAP-ENV", "Envelope", "http://schemas.xmlsoap.org/soap/envelope/");
        writer.WriteAttributeString("xmlns", "ns1", null, value:
"http://disposableemailaddressdetector.email.services.v4_0.soap.server.nameapi.org/");
    }
}
```

که پس از تعریف client به نحو ذیل معرفی می‌شوند:

```
var client = new SoapDisposableEmailAddressDetectorClient();
client.Endpoint.Behaviors.Add(new EndpointBehavior());
```

توسط EndpointBehavior سفارشی، می‌توان به متد **Envelope** OnWriteStart دسترسی یافت و سپس s آن‌را با SOAP-ENV درخواستی این وب سرویس جایگزین کرد. اکنون اگر برنامه را اجرا کنید، بدون مشکل کار خواهد کرد و دیگر پیام یافت نشدن API-Key را صادر نمی‌کند.

کدهای کامل این مثال را از اینجا می‌توانید دریافت کنید.