Chamadas síncronas e assíncronas da API de serviços do WRT

Ao trabalhar com a API de serviços WRT, em geral é possível escolher entre versões **síncronas** e **assíncronas** para o mesmo método. Isto permite ter a mesma funcionalidade numa aplicação porém com um impacto ligeiramente diferente para o restante do seu código.

Este artigo descreve como manipular chamadas síncronas e assíncronas e explica quando cada tipo de chamada deve ser utilizada.

Chamadas síncronas

Comportamento

Chamadas síncronas requerem apenas um argumento, um objeto *criteria* que contém informações relevantes para o método da API de serviços chamado. Dado que a chamada é síncrona, o código de execução JavaScript irá na verdade ser **bloqueado** até que o método retorne.

Valores retornados

Uma chamada a um método síncrono retorna um objeto contendo 2 propriedades:

- **ReturnValue**: Este é o real valor retornado pelo método chamado, sendo assim você deve chegar a definição do método para saber sua estrurura. Como um exemplo: chamando o método IDataSource.GetList() da Api de serviços sobre contatos, esta propriedade irá armazenar as informações de contato atuais. Se um método não retorna qualquer informação (ex, o método ILocation.CancelNotification() da API de serviços de localização), esta propriedade não inclui o objeto retornado.
- **ErrorCode**: Um número que especifica um código de erro. Valor zero significa que não ocorreu erro. A tabela de erros completa está disponível [aqui
- **ErrorMessage**:Uma mensagem descrevendo o erro retornado. Messangens de erro são diferentes para cada API de serviço. Por exemplo, esta é a lista completa de mensagens de erro para a API de serviços de contados.

Prós e contras

- **Pró: codificação simplificada**: você recupera o valor esperado na mesma chamada de método sem a necessidade de implementar um mecanismo separado de callback.
- **Contra: execução bloqueada** até que o método chamado retorne: Se a operação solicitada levar alguns segundos para ser executada, isto poderia causar um problema de usabilidade.

Chamdas assíncronas

Comportamento

Chamadas assíncronas requerem dois argumentos:

- O mesmo objeto *criteria* passado em uma chamada síncrona.
- uma função *callback*: O método a ser chamado quando o método retornar.

Sendo asssíncrono, seu código continua a sua execução normal sem a necessidade de aguardar pelo retorno do método chamado. Quando a informação retornada estiver disponível, o método de *callback* será chamado para manipular o retorno.

Valores retornados

O objeto retornado de uma chamada assíncrona é diferente do retornado por uma chamada síncrona e irá conter as seguintes propriedades:

- **TransactionID**: Um identificador único que identifica a chamada assíncrona. Você pode utilizar este valor, por exemplo, em seu método de *callback* para identificar uma chamada espercífica.
- **ErrorCode**: Um número que especifica um código de erro, como em uma chamada síncrona.
- **ErrorMessage**: Uma mensagem descrevendo o erro retornado, como em uma chamada síncrona.

Método de callback

O método de **callback** é o método que é chamado quando a chamada assíncrona for finalizada e a informação a ser retornada estiver disponível. Ele deve ser definido para aceitar três argumentos:

- **transId**: O ID da chamada assíncrona.
- **eventCode**: Um número representando o status da *callback*. Valores possíveis são:
  - **2**: Evento completado
  - **4**: Evento de erro
  - **9**: Evento em progresso
- **result**: O valor real retornado pelo método chamado. Este valor é o mesmo que o retornado em uma chamada síncrona, então ele irá conter as mesmas três propriedades:
  - **ReturnValue** (opcional, dependendo do método chamado)

(c) Copyright Nokia 2013. All rights reserved.

- **ErrorCode**
- **ErrorMessage**

Em geral as seguintes tarefas são implementadas em uma função de *callback*:

- **Checagem do ID de transação**: para identificar a chamada assíncrona.
- **Checagem do código de evento**: para verificar se o evento realmente foi finalizado (código 2), se ainda está em progresso (código 9), ou apresenta um erro (código 4).
- **Checagem do código de erro**: De forma semelhante a uma chamada síncrona, antes de efetuar a manipulação das propriedades do objeto **ReturnValue** retornado, você deve verificar se houve algum erro na chamada do método verificando a propriedade **ErrorCode**.

Prós e Contras

- **Pró: execução do código não é bloqueada**: Você pode fazer alo a mais (ex.: animações, outras chamadas de método) enquanto o método chamado é executado.
- **Contra: Manipulação manual da sincronia** no seu código, definindo ações apropriadas dentro de um método de *callback* e gerenciando cuidadosamento os estados da aplicação e a UI.

Exceções

Nem todos os método suportam ambos os tipos de chamadas síncrona e assíncroa.

Por exemplo, o método IMessaging.GetList() da API de serviços de mensagem suporta apenas chamadas síncronas, enquanto que o método IDataSource.GetList() da API de serviços de gerenciamanto de mídia suporta apenas assíncrono.

Estudo de caso: recuperando a lista de contatos

Este exemplo mostra como recuperar a lista de contatos do dispositivo, usando tanto chamada síncrona quanto assíncrona. A manipualação de erros não é realizada neste exemplo, porém você deve realiza-lo em uma aplicação real.

Primeiro, instanciae o objeto *Service*:

```
var so = device.getServiceObject("Service.Contact", "IDataSource");
```

Agora definir o objeto *criteria* que irá ser usado para recuperar os contatos. Este objeto é o mesmo para chamadas síncronas e assíncronas.

```
var criteria = new Object();
criteria.Type = "Contact";
```

Então, definimos a função que irá realizar o tratamento dos contatos recuperados. Esta função pode ser igualmente utilizada para ambas as chamadas:

```
function handleContacts(contacts)
{
 try
 {
  contacts.reset();

  var contact;

  while((contact = contacts.getNext()) != undefined)
  {
   //handle current contact
  }
 }
 catch(e)
 {
  alert('Error while showing contacts');
 }
}
```

Versão síncrona

A versão de chamada síncrona é bem direta. Você apenas precisa chamar o método GetList() e passar a informação retornada para o método handleContacts():

```
var result = so.IDataSource.GetList(criteria);

handleContacts(result.ReturnValue);
```

Versão assíncrona

Diferentemente da anterior, você deve definir um método de *callback* que irá ser chamado quando a chamada assíncrona ao método GetList() retornar. Em aplicações reais, você deveria também verificar os ID de transação para checar se a chamada f está realmente chamando o método de *callback*, e um **eventCode** para verificar se o evento está completo (o valor deve ser 2).

```
function contactsCallback(transId, eventCode, result)
{
 handleContacts(result.ReturnValue);
}
```

Então, o método anterior (síncrono) irá mudar, tendo como segundo argumento a funçao *contactsCallback*. Aqui também, você deve verificar o erro

(c) Copyright Nokia 2013. All rights reserved.

retornado por esta chamada, e armazenar a propriedade **TransactionID** que irá ser usada no método de *callback*.

```
var result = so.IDataSource.GetList(criteria, contactsCallback);
```

Leituras complementares

- WRT Service API reference
- Web Runtime Wiki section
- Web Runtime technology landing page

(c) Copyright Nokia 2013. All rights reserved.

(c) Copyright Nokia 2013. All rights reserved.