KAPITOLA 6

Data ve Flex aplikacích

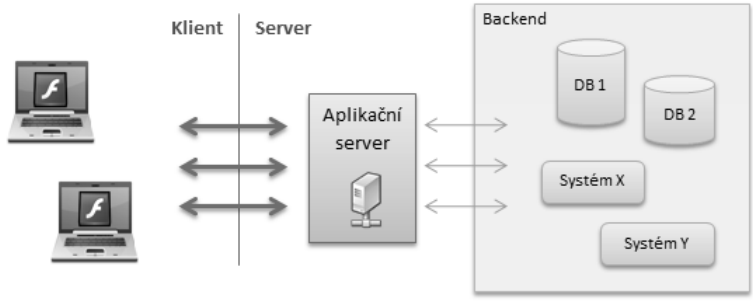
Práce s daty se intuitivně pojí s přistupováním k databázi a jiným vzdáleným zdrojům, data jsou ale všude, kam se člověk podívá – lokální proměnné drží data, uživatel je zadává z klávesnice, jiná se stahují ze serveru a tak dále. V této kapitole se tedy datům budeme důkladně věnovat a začneme klíčovou otázkou, odkud a jak data získávat.

Principy vzdáleného přístupu k datům

Flex má při přístupu ke vzdáleným datům dvě obecné charakteristiky, které ho odlišují od serverových technologií typu PHP, ASP.NET a dalších:

- *Do databáze či k jiným zdrojům dat se nepřistupuje přímo*, ale přes nějakou mezivrstvu (například PHP skript).
- Volání vzdálených služeb jsou *vždy asynchronní*.

První bod je dán obvyklou bezpečností politikou – málokdy bývají databáze či interní systémy přístupné široké množině klientů, většinou je to přesně naopak. Na veřejně dostupném serveru proto musí běžet takzvaný *aplikační server*, který zpracovává požadavky klientů a v nejjednodušším případě vrací data získaná z databází či jiných systémů. Způsob komunikace naznačuje obrázek 6.1 – důležité je na něm především to, že klient nikdy nemá přímý přístup k systémům v oblasti označené jako *backend*.



Obrázek 6.1 Role aplikačního serveru při přístupu k datům

Poznámka

Aplikační server může být jednoduchý PHP skript nasazený na webovém serveru, ale také pokročilý produkt podporující škálování, synchronizaci klientů a celou řadu dalších služeb. O některých konkrétních aplikačních serverech si povíme později v této kapitole.

Asynchronní charakter vzdálených volání ve Flexu znamená, že okamžik odeslání požadavku a přijetí odpovědi nejsou časově propojené události. V reálném světě je analogií například vyřizování klientů na pobočkách bank – člověk přijde, vyzvedne si lístek s číslem a jde se posadit. Na řadu pak přijde ve chvíli, kdy je přepážka volná, což může být hned, za malou či delší chvilku nebo obecně kdykoliv (možná v daný den už na klienta ani řada nevyjde). Vzdálená volání ve Flexu fungují velmi podobně – odeslání požadavku je jakoby vybráním lístku a na vrácení dat ze serveru je kód upozorněn notifikačním mechanizmem, práce s daty může pokračovat až potom.

V serverových technologiích lze většinou zvolit, jestli volání proběhne synchronně nebo asynchronně, ale ve Flexu tato volba není – všechna volání jsou *vždy* asynchronní. Nemožnost výběru může vypadat jako omezení, v praxi se ale jedná o užitečnou věc, protože se nemůže stát, že by Flex aplikace kvůli čekání na data „zamrzla". Tím odpadá těžký programátorský úkol, kterým je tvorba vícevláknových aplikací, kde je potřeba správě zamykat data, řešit problémy se synchronizací a podobně.

Poznámka

Máloco lze pronést absolutně a to platí i zde. Flex aplikace ve skutečnosti může k databázi přistoupit přímo, pokud je tato nezabezpečená a Flex aplikace použije nízkoúrovňová rozhraní Flash Playeru pro komunikaci na úrovni *socketů*. Rovněž synchronnost by šlo simulovat umělým zacyklením aplikace, dokud se data ze serveru nevrátí, to vše je ale spíše pro zajímavost – v reálu se tyto „techniky" samozřejmě nepoužívají (s výjimkou specializovaných aplikací na administraci interních systémů).

Přístup ke vzdáleným datům

Prakticky všechny Flex aplikace potřebují získávat data ze vzdálených serverů a ve čtvrté verzi Flex frameworku k tomu slouží dva základní přístupy:

- Komponenty pro vzdálený přístup (například `HTTPService` či `WebService`).
- Vlastnosti Flexu a Flash Builderu pro datově orientovaný vývoj (*data centric development*, *DCD*).

Vlastnosti pro datově orientovaný vývoj jsou novinkou Flexu 4 a Flash Builderu 4. Jejich hlavním cílem je zjednodušit vývoj datově orientovaných aplikací, to ale neznamená, že můžeme úplně zapomenout na původní komponenty pro vzdálený přístup, právě naopak – DCD je do budoucna určitě slibným přístupem, v současné verzi má ale některá omezení, a přímý přístup je tedy stále potřeba ovládat.

Základní komponenty pro přístup k datům

Flex framework obsahuje tři základní komponenty pro přístup k datům:

- `HTTPService`
- `WebService`
- `RemoteObject`

Poznámka

Tyto komponenty se řadí mezi nevizuální stále jsou však plnohodnotnými Flex komponentami – můžou mít vlastnosti, události, metody a tak dále.

Komponenta `HTTPService` umí pracovat s webovými zdroji, které jsou zpřístupněny skrze protokol *HTTP(S)*, pomocí této komponenty bychom tedy například mohli stáhnout HTML nějaké webové stránky nebo ze serveru načíst XML či JSON. `WebService` zase umí volat *SOAP webové služby*, které svou zlatou éru prožily začátkem tohoto tisíciletí, a pomocí `RemoteObject` můžeme přímo volat metody na vzdálených třídách, které jsou implementované v Javě, PHP, C# nebo jiném jazyku, pro který existuje serverový produkt zpřístupňující rozhraní přes protokol *AMF*. Postupně si tyto tři komponenty představíme.

KOMPONENTA HTTPSERVICE

Komponenta `HTTPService` slouží jako jednoduchý HTTP klient podporující operace `GET`, `POST` a pár dalších. V podstatě tato komponenta simuluje to, co umí běžný webový prohlížeč – nějaká data získat, jiná odeslat. Ve světě *AJAXového* vývoje je blízkou analogií `XMLHttpRequest`.

Tip

Pokud se potřebujete rychle seznámit s protokolem HTTP, dobrým zdrojem je například stránka *http://en.wikipedia.org/wiki/Hypertext_Transfer_Protocol*.

Komponentu `HTTPService` lze použít jak z MXML, tak z ActionScript kódu. Základní použití demonstruje následující ukázka:

```
<?xml version="1.0" encoding="utf-8"?>
<s:Application xmlns:fx="http://ns.adobe.com/mxml/2009"
         xmlns:s="library://ns.adobe.com/flex/spark"
         xmlns:mx="library://ns.adobe.com/flex/mx">
    <fx:Declarations>
        <s:HTTPService id="service" url="data/catalog.xml" />
    </fx:Declarations>

    <mx:DataGrid dataProvider="{service.lastResult.catalog.product}"
                 creationComplete="service.send()" />
</s:Application>
```

V tomto příkladu používáme dvě komponenty: datovou mřížku `DataGrid`, se kterou jsme se letmo setkali v kapitole o vizuálních komponentách, a `HTTPService`, komponentu pro přístup k datům. Jelikož je nevizuální, musí být uzavřena v elementu `<fx:Declarations>`, jinak by aplikace nešla zkompilovat.

V této základní aplikaci děláme tři hlavní věci:

1. Deklarujeme komponentu `HTTPService`.
2. Z ovladače události `creationComplete` datové mřížky voláme metodu `send()` naší služby.
3. `DataGrid` navážeme na vrácená data pomocí vlastnosti `dataProvider`.

Poznámka

Událost `creationComplete` patří mezi takzvané události životního cyklu komponent a vyvolává se v okamžiku, kdy je komponenta vytvořena a připravena k použití. Mezi další podobné události patří například:

- `preinitialize` – používá se k ovlivnění inicializace (v praxi ne příliš často).
- `initialize` – komponenta byla inicializována, ale ještě neproběhlo její umístění do uživatelského rozhraní. Zde je tedy vhodný okamžik k ovlivnění vlastností komponenty těsně před jejím vykreslením.
- `applicationComplete` – tuto událost vyvolává aplikace ve chvíli, kdy byla dokončena fáze startu (všechny zanořené kontejnery a jejich komponenty byly vytvořeny a zobrazeny).

Konkrétně událost `creationComplete` uvidíte ve Flex aplikacích ošetřovanou poměrně často.

Komponenta `HTTPService` se pomocí vlastnosti `url` odkazuje na zdroj, který má být za běhu načten. V příkladu se jedná o soubor `catalog.xml` ve složce `data` – tato cesta je relativní a vyhodnocuje se stejně jako v běžném HTML, pokud tedy aplikace bude nasazená na adrese *http://www.example.com/MyApp.swf*, vyhodnotí se URL z příkladu jako odkaz na *http://www.example.com/data/catalog.xml* (při lokálním ladění analogicky).

Druhou možností jsou absolutní URL, která se používají pro zdroje na jiných doménách. Načtení XML souboru z domény *flexkniha.cz* by se provedlo pomocí následujícího nastavení:

```
<s:HTTPService id="service" url="http://flexkniha.cz/priklady/catalog.xml" />
```

Přístup ke zdrojům na cizí doméně je v prostředí webu typicky omezen určitými pravidly a nejinak je tomu i v případě Flexu – přečtěte si proto prosím důležitý rámeček *Zdroje z jiných domén*.

Zdroje z jiných domén

Podobně jako ve světě HTML a JavaScriptu, i Flex aplikace to má při přístupu k datům na cizí doméně poměrně těžké. Flash Player totiž ctí zásadu stejného původu (*same origin policy*), což znamená, že ve výchozím stavu je možný přístup pouze ke zdrojům na vlastní doméně, nikoliv cizí.

Překonání tohoto omezení spočívá v nasazení souboru `crossdomain.xml` na cílovou doménu. Soubor vypadá takto:

```
<?xml version="1.0"?>
<!-- http://www.a.com/crossdomain.xml -->
<cross-domain-policy>
    <site-control permitted-cross-domain-policies="by-content-type"/>
    <allow-access-from domain="*.example.com"/>
</cross-domain-policy>
```

Při nasazení na doménu `a.com` a běhu Flex aplikace z domény `example.com` začnou vzdálená volání fungovat.

Co dělat, pokud k cizí doméně nemáme přístup? Zde v zásadě zbývá možnost jediná – na vlastní doménu nasadit serverový skript, který požadavek z naší Flex aplikace převezme, předá cílové službě, získá od ní data a vrátí je zpět do Flex aplikace. Takto fungujícímu serverovému skriptu se říká *proxy* (což v překladu znamená *prostředník*) a kromě toho, že by nebylo příliš složité si takový skript napsat vlastnoručně, lze použít i několik existujících řešení, například open source server *BlazeDS* od Adobe. V zásadě tedy není problém *same origin* politiku obejít, stejnou techniku koneckonců používají i *AJAXové* aplikace.

Ještě upozornění závěrem: při lokálním ladění (spouštění z lokálního disku) se politika stejného původu neuplatňuje, a budou vám tak fungovat i volání na domény, které soubor `crossdomain.xml` nemají. Pokud se tedy chyby začnou vyskytovat po nasazení na reálný server, víte, kde hledat…

Vraťme se nyní zpátky k příkladu. Komponentu `HTTPService` máme definovanou, nyní ji potřebujeme zavolat a vrácená data zobrazit. Zavolání je potřeba udělat z ActionScript kódu a v příkladu jsme použili ovladač události `creationComplete` na komponentě `DataGrid`, mohli jsme ale stejně dobře použít například tlačítko a jeho událost `click` nebo cokoliv jiného. Důležité je, že se zavolá metoda `service.send()`, což zahájí načítání dat.

Všimněte si, že to je v ovladači události vše. Pokud jste zvyklí na práci například v PHP, mohli byste čekat kód v tomto stylu:

```
$data = file_get_contents('http://www.example.com/data/catalog.xml');
$grid->data = $data; // pseudokód
```

Ekvivalentní metoda `send()` ale ve Flexu nemůže žádná data vrátit, protože ta budou díky asynchronní povaze k dispozici až někdy v budoucnu nebo také možná nikdy, pokud uživateli zrovna nefunguje internetové připojení, server neodpovídá nebo jsme například zadali špatné URL. Mechanizmus, kterým se data dostanou do komponenty `DataGrid`, tedy musí být jiný. Možnosti jsou dvě:

- Vlastnost `lastResult`
- Ošetření události `result`

Následující příklad ukazuje jednodušší a přímočařejší možnost, využití vlastnosti `lastResult`:

```
dataProvider="{service.lastResult.catalog.product}"
```

Mechanizmus vázání dat (*data binding*) zde v plné kráse ukazuje svoji sílu i eleganci – neřešíme, kdy přesně se data vrátí a jak tento okamžik ošetřit, jednoduše navážeme vlastnost `dataProvider` na proměnnou `service.lastResult` „tečka něco" (závisí na konkrétním zdroji, v našem případě má XML dokument kořenový element `catalog` a několik elementů `product`, to ale není na tomto místě důležité). Ve chvíli, kdy jsou data k dispozici, dojde k automatickému naplnění komponenty `DataGrid` a výsledek bude vypadat jako na obrázku 6.2.

camera	description	highlight1	highlight2	image	name	price	productId	qtyInStock	series	triband	video
false	Easy to use with	MMS	Large color disp	Nokia_6010.gif	Nokia 6010	99.99	1	2	6000	false	false
false	Light up the nigh	Glow-in-the-dark	Flashing lights	Nokia_3100_blu	Nokia 3100 Blue	139	2	1	3000	true	false
false	Light up the nigh	Glow-in-the-dark	Flashing lights	Nokia_3100_pir	Nokia 3100 Pink	139	3	7	3000	true	false
false	Designed for bo	Multimedia mes	Animated screer	Nokia_3120.gif	Nokia 3120	159.99	4	15	3000	true	false
true	The Nokia 3220	MIDI tones	Cut-out covers	Nokia_3220.gif	Nokia 3220	159.99	5	5	3000	false	false
true	Messaging is m	Infrared or Blueb	Built-in XHTML b	Nokia_3650.gif	Nokia 3650	199.99	6	8	3000	false	true

Obrázek 6.2 DataGrid naplněný daty ze serveru

Druhá možnost je ošetřit vrácená data manuálně, což znamená o něco více práce, ale současně více flexibility. V reálných aplikacích se tento přístup používá často:

```
<?xml version="1.0" encoding="utf-8"?>
<s:Application xmlns:fx="http://ns.adobe.com/mxml/2009"
               xmlns:s="library://ns.adobe.com/flex/spark"
               xmlns:mx="library://ns.adobe.com/flex/mx">
    <fx:Script>
        <![CDATA[
            import mx.rpc.events.ResultEvent;

            protected function onResult(event:ResultEvent):void {
                grid.dataProvider = event.result.catalog.product;
            }

        ]]>
    </fx:Script>
    <fx:Declarations>
        <s:HTTPService id="service" url="data/catalog.xml"
                       result="onResult(event)"/>
    </fx:Declarations>
```

```
    <mx:DataGrid id="grid" creationComplete="service.send()" />
</s:Application>
```

Z tohoto příkladu zmizelo vázání dat a naopak přibylo ošetření události `result` na komponentě `HTTPService`. Všimněte si, že způsob zápisu ošetření události je u `HTTPService` naprosto totožný, jako by byl například v případě tlačítka nebo jakékoliv jiné vizuální komponenty Flex frameworku. Jak víme z předchozích kapitol, Flash Player nedělá mezi ActionScript třídami rozdíly – `Button` s událostí `click` je pro něj zcela to stejné jako `HTTPService` s událostí `result`, i zápis tedy odpovídá.

V ovladači potom k výsledku přistupujeme přes objekt `event.result`, který obsahuje totéž co v předchozím příkladu `service.lastResult`. Výsledek manuálně přiřadíme do vlastnosti `dataProvider` komponenty `DataGrid`, čímž dostaneme stejný vizuální výsledek jako v předchozím příkladu.

Poznámka

V ovladači události používáme `event.result` namísto `service.lastResult`, protože tato služba mohla být teoreticky zavolána vícekrát v krátkém časovém rozmezí (ne v našem konkrétním příkladu, ale obecně) a kdo ví, co v okamžiku vrácení dat vlastnost `lastResult` obsahuje. Přistoupení k `event.result` je bezpečnější – zaručuje, že se vždy odkážeme na správnou množinu dat.

Stejně jako používáme událost `result` pro zpracování výsledku, událost `fault` nás informuje o tom, že operace selhala. Ukázkový kód může vypadat následovně:

```
protected function onFault(event:FaultEvent):void {
    Alert.show("Chyba při načátání XML souboru");
}

...

<s:HTTPService id="service" url="data/catalog1.xml"
               result="onResult(event)"
               fault="onFault(event)"/>
```

V ovladači události `fault` budeme patrně chtít uživateli zobrazit, že došlo k problému, ale můžeme dělat celou řadu dalších věcí, například událost zapsat do logu nebo zkusit vyvolat vzdálenou operaci znovu.

Poznámka

Logování je ve Flex aplikacích poměrně delikátní záležitost, protože RIA aplikace mají obecně dost omezené možnosti ukládání dat na uživatelův lokální disk. Ve Flexu lze použít takzvané sdílené objekty (*shared objects*), což je obdoba *cookies* v prohlížečích, nebo lze například Flash Player nakonfigurovat tak, aby výstup funkce `trace()` ukládal do souboru `flashlog.txt` v uživatelově profilu, to jsou však obě omezené a komplikované možnosti. Ideální by bylo zprávu odeslat na server, ale ten také nemusí fungovat. Jednodušší situaci mají tedy Adobe AIR aplikace – ty mohou data ukládat bez velkých omezení a logování například do souboru je v nich běžnou technikou. Data lze pak na server odeslat později či je od uživatele získat jinak.

V ovladači události `fault` máme k dispozici objekt `event` typu `FaultEvent`, který obsahuje následující zajímavé vlastnosti:

- `event.fault.faultCode` – kód chyby (jedná se o textový řetězec, například `"Server.Error.Request"`)
- `event.fault.faultString` – krátký popis chyby
- `event.fault.faultDetail` – popis chyby včetně technických detailů, hodí se tedy pro technickou podporu

Nejčastější důvody selhání vzdáleného volání

Ve Flexu je mnoho důvodů, proč může vzdálené volání selhat. Mezi ty nejčastější patří:

- Uživatel ztratí internetové připojení – například je na okraji signálu bezdrátové sítě.
- Server neodpovídá – je přetížen, obsahuje programátorskou chybu a podobně.
- Vzdálené volání porušuje bezpečnostní pravidla Flash Playeru.
- Serverový zdroj jsme zavolali nekorektně (od špatné URL až po špatně předané argumenty).

… a našlo by se mnoho dalších. Důležitým ponaučením z tohoto rámečku je tedy vždy počítat s možností selhání – ve Flexu (jako v každé RIA aplikaci) se to může stát každou chvíli.

Obecný mechanizmus fungování `HTTPService` jsme tedy prošli, nyní pár užitečných poznámek pro praxi:

- Komponentu `HTTPService` lze jako každou jinou třídu definovat v čistém ActionScriptu, čehož se využívá k zapouzdření logiky pro přístup k datům do samostatných tříd. U větších aplikací je tato technika více než doporučeníhodná.
- S jakými formáty si `HTTPService` rozumí? Automaticky si poradí s XML soubory, které umí buďto převést do hierarchie ActionScript objektů (čehož jsme využili v příkladu, když jsme k elementům XML souboru přistupovali jednoduše přes tečkovou notaci `lastResult.catalog.product`), nebo umí zpřístupnit XML soubor pomocí *E4X*. Dále si `HTTPService` poradí s formátem podobným tomu, který se používá v URL za otazníkem, tedy něco jako `klíč1=hodnota1&klíč2=hodnota2`, a nakonec přirozeně načte zdroj, který vrací text v libovolném formátu – ten už je potom ovšem potřeba zpracovat v aplikaci manuálně. Uvedené možnosti odpovídají jedné z hodnot vlastnosti `resultFormat`, která může být `object`, `array`, `xml`, `flashvars`, `text` nebo `e4x`.
- Pro použití jiné HTTP metody než výchozí `GET` slouží vlastnost `method`. Nejpopulárnější bude patrně `POST`, ale podporovány jsou i další, například `HEAD`, `PUT` nebo `DELETE`. Pokud jako součást HTTP POST požadavku potřebujeme odeslat nějaká data, můžeme je předat metodě `send()` jako první a jediný parametr.

Tip

V současné době vzrůstá popularita takzvaných REST architektur, které namísto komplikovaných webových služeb využívají jednoduchý přístup ke zdrojům pomocí protokolu HTTP a jeho základních přístupových metod `GET`, `POST`, `DELETE` a dalších. Komponenta `HTTPService` se tedy dobře hodí při implementaci klientské části REST aplikace.

Ačkoliv je `HTTPService` základní komponentou, využívá se v aplikacích poměrně velmi často, a je tedy dobré její fungování znát.

KOMPONENTA WEBSERVICE

Komponenta `WebService` slouží pro přístup k *SOAP webovým službám*, které samy sebe popisují pomocí jazyka *WSDL* (*Web Services Description Language*). WSDL je standardizovaný formát, díky kterému může komponenta `WebService` zjistit, jaké metody jsou na službě dostupné a jaké datové typy se mají použít pro parametry či při čtení odpovědi. *SOAP* je potom standardní formát pro kódování požadavků a odpovědí (je založený na XML).

S webovými službami častěji než s jinými narazíte na omezení politiky stejného původu (*same origin policy*), takže jen připomenu rámeček *Zdroje z jiných domén* – pokud nad vzdálenou doménou nemáte kontrolu, nejjednodušší je využít produkt typu *BlazeDS* a k webové službě přistupovat přes *proxy*.

V následujícím příkladu si ukážeme, jak komponentu `WebService` použít:

```
<?xml version="1.0" encoding="utf-8"?>
<s:Application xmlns:fx="http://ns.adobe.com/mxml/2009"
               xmlns:mx="library://ns.adobe.com/flex/mx"
               xmlns:s="library://ns.adobe.com/flex/spark">

    <fx:Declarations>
        <s:WebService id="service"
           wsdl="http://www.w3schools.com/webservices/tempconvert.asmx?WSDL" />
    </fx:Declarations>

    <s:Label text="{service.CelsiusToFahrenheit.lastResult}"
             creationComplete="service.CelsiusToFahrenheit(30)" />

</s:Application>
```

Velmi podobně jako v příkladu s `HTTPService` i zde v sekci `<fx:Declarations>` deklarujeme webovou službu s několika základními vlastnosti, voláme ji z nějakého ovladače události (`creationComplete` na komponentě `Label`) a výsledek zobrazíme v uživatelském rozhraní pomocí vázání dat a vlastnosti `lastResult`. Oproti komponentě `HTTPService` si všimněte několika základních rozdílů:

- Komponenta `WebService` se odkazuje na WSDL místo na službu samotnou.

- Nevoláme vždy stejnou metodu `send()`, ale kontextově správné jméno metody, které daná služba podporuje (tato ukázková služba z webu *W3Schools* podporuje dvě metody, `CelsiusToFahrenheit` a `FahrenheitToCelsius`).
- Rovněž `lastResult` není vlastnost přímo na objektu `WebService`, nýbrž na jednotlivých metodách dané webové služby – jedině tak můžeme rozlišit mezi více metodami jedné webové služby.

Jinak je chování v principu velmi podobné `HTTPService`, což ostatně dělá programování ve Flexu příjemné.

KOMPONENTA REMOTEOBJECT

Skrze komponentu `RemoteObject` můžeme volat metody na vzdálených objektech (tím jsou myšleny například instance Java nebo PHP tříd), jako by byly lokální. O *serializaci* a *deserializaci* požadavků a odpovědí je postaráno automaticky, takže z pohledu klientského kódu je odpovědí přímo datový typ ActionScriptu a není potřeba například ručně procházet sadu XML elementů a postupně je konvertovat.

Původní implementace `RemoteObject` na serveru předpokládala *ColdFusion* (serverová technologie Adobe) nebo třídy jazyka Java, postupně ale vznikly i implementace pro další technologie, takže dnes lze `RemoteObject` používat spolu se serverovou stranou implementovanou v PHP, .NETu, Pythonu a dalších populárních technologiích.

Klíčovým pojítkem je komunikační protokol a formát v jednom, *AMF* (*Action Message Format*), který je binární a efektivní při serializaci / deserializaci ActionScript objektů (podle některých měření 3x až 10x výkonnější než textové formáty typu XML).

Poznámka

Specifikace AMF je volně dostupná, takže se není potřeba obávat problémů s uzavřeností.

Pokud bychom měli serverovou třídu s metodou `getProducts()`, v klientském kódu bychom k ní přistoupili velmi podobně, jako když komponenta `WebService` načítá data z webové služby:

```
<mx:RemoteObject id="service" .../>
<mx:DataGrid creationComplete="service.getProducts()"
             dataProvider="{service.getProducts.lastResult}" />
```

Z pohledu aplikačního programátora je tedy kód prakticky stejný jako u komponenty `WebService`, takže se nemusíme učit žádný nový koncept.

Poznámka

Java je nejen syntakticky podobná ActionScriptu, ale v mnoha ohledech používá i stejné konvence (třídy používají *PascalCasing*, vlastnosti a metody *camelCasing* a podobně), takže je převod mezi Javou a ActionScriptem velmi přirozený.

Tím jsme prošli základní komponenty pro přístup k datům. Pojďme se nyní podívat na jiný přístup k vývoji aplikací konzumujících data.

Datově orientovaný vývoj ve Flash Builderu

Od verze 4 jsou ve Flexu a Flash Builderu zabudované nástroje, které přístup k datům usnadňují. Adobe jim říká vlastnosti pro datově orientovaný vývoj (*data centric development*, zkráceně *DCD*) a my se na ně nyní podíváme.

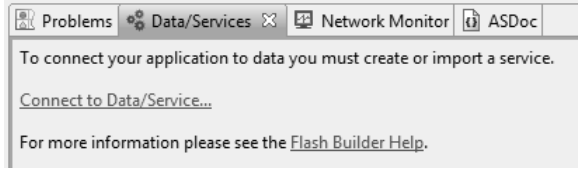
Zatímco u základních datových komponent `HTTPService`, `WebService` a `RemoteObject` jsou všechny informace o tom, co chceme udělat, víceméně pouze v naší hlavě, v případě *DCD* řekneme Flash Builderu, s jakým zdrojem budeme pracovat, a on se pokusí na základě pokusných dotazů o službě zjistit co nejvíce informací a vygenerovat nám lokální objekty pro snadnější přístup. Datově orientované nástroje fungují dobře nad následujícími zdroji:

- HTTP služby zpřístupňující data v duchu architektury REST
- SOAP webové služby
- Speciální servery jako *BlazeDS* nebo *LiveCycle Data Services*

Krásné na datově orientovaných vlastnostech je, že ačkoliv se například REST služba a SOAP webová služba liší jak ve formátu přenosu dat, tak ve způsobu vyvolávání různých typů operací, z pohledu Flex vývojáře se obě služby konzumují zcela stejně. Demonstrujme si to na dvou příkladech, které jsme o pár stránek dříve implementovali pomocí „surových" komponent `HTTPService` a `WebService`.

První příklad načítal XML soubor `catalog.xml`, ukažme si implementaci pomocí datově orientovaných nástrojů Flash Builderu.

V prvním kroku musíme Flash Builderu říct, s jakým datovým zdrojem budeme pracovat. Učiníme tak na panelu **Data/Services**, který se ve výchozím rozložení nachází ve spodní skupině panelů (viz obrázek 6.3).



Obrázek 6.3 Panel Data/Services

Zde zvolíme **Connect to Data/Service**, načež se objeví dialog 6.4. V něm vybereme položku **XML**.

Následující dialog je důležitý a umožňuje nám nakonfigurovat celou řadu charakteristik služby. V první řadě musíme zadat cestu k našemu XML souboru, jak ukazuje obrázek 6.5.