Databáze a XML v .NET/C#

ADO.NET

Entity Framework

LINQ to XML

Soubory

- Třídy ze jmenného prostoru `System.IO`
  - `File`
  - `Directory`
  - `Specifické formáty jako databáze nebo z knihoven`
- Proudy
  - `StreamReader, StreamWriter`

ADO.NET

Historie

ADO.NET Entity Framework

ADO.NET

- následník ADO (ActiveX Data Objects)
- hiearchie rozhraní a abstraktních tříd
- DataReader
- DataSet
- Typovaný DataSet
- LINQ to SQL
- ADO.NET Entity Framework

ADO.NET hierarchie tříd

- rozhraní IDbXXX a abstrakní třídy DbXXX
  - `IDbConnection, IDbCommand, IDbDataReader`
  - `DBConnection, DbCommand, DbDataReader`
- na těchto třídách jsou založeny konkrétní tříd - Data Providers
  - např. pro MS SQL server existují třídy `SqlConnection`, `SqlCommand`, `SqlDataReader`, pro Oracle existují `OracleConnection`, …

Databázové spojení

- třída `DbConnection` a její potomci
- reprezentuje připojení k databázi
- spojení je určeno podle connection stringu předaného konstruktoru nebo pomocí vlastnosti `ConnectionString`
  - lze nastavovat v konfiguraci
  - závisí na konkrétním datovém zdroji
  - příklad: `"server=localhost;user=root;database=test"`
  - seznam na http://connectionstrings.com/
  - programově se zpřístupní pomocí třídy `ConfigurationManager`
- používá `Open` a `Close` metody pro zavření a otevření

Databázový příkaz

- třída `DbCommand` a potomci
- představuje příkaz předaný databázi databázi (obvykle SQL)
- vlastnosti `Connection`, `CommandText` , `CommandType`
- provedení příkazu:
  - `ExecuteNonQuery`
  - `ExecuteScalar`
  - `ExecuteReader`
- kolekce parametrů `Parameters`
  - předání parametrů do SQL dotazu – brání SQL injection
  - každý provider má vlastní třídu parametrů
  - vlastnosti: `ParameterName`, `Value`, `DbType`, `Direction`

Data Readers

- třída `IDataReader`
- sekvenční přístup k datům (readonly, move forward)
- vyvolán z `DbCommand` pomocí `ExecuteReader`
- data jsou čtena po řádcích
- přístup ke sloupcům pomocí indexerů
  - `this[int]` – podle pořadí sloupce
  - `this[string]` – podle názvů sloupce
- pro běžné datové typy jsou k dispozici vlatnosti Get<Typ>

Transakce

- třída `DbTransaction`
- metody `Commit` a `Rollback`
- vytváří se z `Connection` metodou `BeginTransaction`
- metody `Rollback` a `Commit`
- k příkazu (`DbCommand`) se přidá nastavením vlastnosti `Transaction` před voláním `Execute` metody
- vlastnost `IsolationLevel`

Transakce

- pohodlněji je lze řídit pomocí třídy `TransactionScope`
- třídy podporující transakce jsou schopny rozeznat, zda transakce běží a připojit se k nim automaticky
- transakce je možné skládat dohromady a vnořovat do sebe
- `TransactionScope` implementuje `IDisposable`
  - používá se společně blokem `using`
- potvrzení transakce se provede pomocí metody `Complete`
- volání `Dispose` značí konec transakce
  - nezavolání `Complete` před skončením platnosti proměnné `TransactionScope` vyvolá zrušení transakce

Transakce

- enum `TransactionScopeOption` předávaný konstruktoru určuje rozsah transakce
  - `Required`
    - vyžaduje transakci, jestliže již okolo existuje použije ji nebo vytvoří novou
  - `RequiresNew`
    - vyžaduje transakci, vždy vytváří novou
  - `Supress`
    - nikdy se neúčastní transakce, i když už transakce okolo existuje

Dataset

- in-memory reprezentace databáze
  - tabulky, řádky, sloupce, vazby
- pro odpojená řešení
- k naplnění se používá třída `DataAdapter`
  - pomocí zadaných SQL příkazů převádí obsah mezi databází a datasetem
- typovaný dataset
  - generován dataset designerem
  - silně typovaný, sloupce jako vlasnosti řádků

ORM – Object-relational mapping

- co je to?
  - abstrakce
  - mezi objektovým světem a relačním je velký rozdíl
    - tabulky – dvourozměrné
    - objekty – dědičnost, zapouzdření, polymorfismus
  - práce s relačními daty jako by to byly objekty v paměti
  - skrývá komplexnost práce s daty
  - jednotný způsob práce
- proč?
  - produktivita
  - nezávislost na databázi (téměř)

ORM - dědičnost

- table per hierarchy
  - různé třídy v téže tabulce
  - problém s prázdnými sloupci
- table per type
  - každá třída má vlastní tabulku
  - mohu společné sloupce přesunout do tabulky, která reprezentuje předka (s cizím klíčem potomek → předek)
  - problém, když mám ID předka → JOINY

Způsob práce

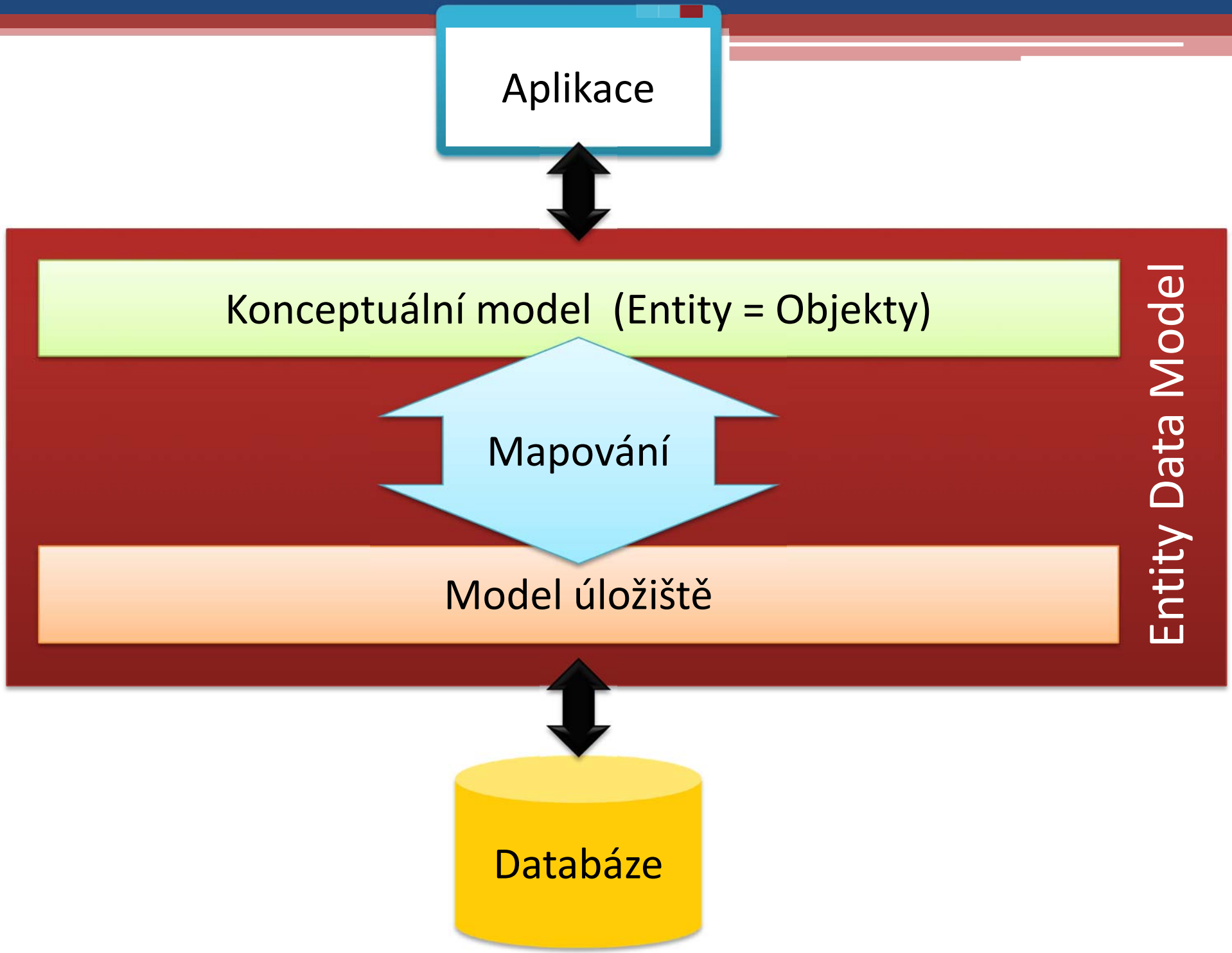
- Entity Framework 1
  - .NET Framework 3.5 SP1
  - jen Database First přístup
- Entity Framework 4
  - .NET Framework 4
  - přidává Model First přístup
- Entity Framework 4.1
  - přidává Code First přístup

ADO.NET Entity Framework

- ORM nástroj od Microsoftu

Entity Data Model

- aplikační model
- nezávislý na databázi
- skládá se ze 3 vrstev
  - konceptuální datový model
    - CSDL – Conceptual Schema Definition Language
  - mapovací model
    - MSDL – Mapping Schema Definition Language
  - model úložiště
    - SSDL - Store Schema Definition Language


Aplikace
Konceptuální model (Entity = Objekty)
Mapování
Model úložiště
Entity Data Model
Databáze

Entity Data Model

- při změně databáze se změní jen model úložiště a mapování
- konceptuální vrstva zůstáne nezměněna
- ideálně: aplikační logika si změny vůbec nevšimne

Entity Type

- základ EDM
- obdoba tříd v objektovém programování
  - entity jsou instancemi entity typů
  - `Person`, `Order`, `OrderItem`
- jedinečný název a klíč
- vlastnosti
  - jméno + typ + omezení (Nullable, MaxLength, Precision)
  - skalární vs. komplexní
- navigační vlastnosti
  - jméno + asociace + konec asociace

Primitivní datové typy

- sada datových typů použitelných v konceptuálním modelu
- zastupují datové typy používané datovým úložištěm
- EDM nepodporuje kolekce primitivních datových typů

- Binary
- Boolean
- Byte
- Double
- DateTime
- DateTimeOffset
- Decimal
- Double
- Float
- Guid
- Int16
- Int32
- Int64
- SByte
- String
- Time

Association Type

- popisují vztah mezi dvěma entitami
- má dva konce (na každé entitě jeden)
- jedinečný název
- omezení referenční integrity (cizí klíč)
- každý konec má násobnost
  - one (1)
  - zero or one (0..1)
  - many (*)

Entity Container

- logické seskupení entitních množin, asociací a funkcí

Třída `ObjectContext`

- reprezentuje Entity Container
- používá se pro dotazování a manipulaci s entitami
- zahrnuje
  - spojení k databázi
  - metadata popisující model (EDM)
  - `ObjectStateManager` pro sledování stavu objektů
- nejsou vláknově bezpečné
  - na webu vytvářejte *per request*, na desktopu *per form*
- SaveChanges pro uložení změn do databáze, provádí se v transakci

Třída `ObjectStateManager`

- udržuje informace o stavu objektů v kontextu
- vytváří třídu `ObjectStateEntry` pro každý objekt
- uchovává se:
  - klíč
  - stav entity
  - aktuální a původní hodnoty vlastností
  - jména změněných vlastností
- u POCO entit není stav sledován automaticky a musí se volat `DetectChanges` na kontextu pro synchronizaci objektového grafu s `ObjectStateManager`

Stav entity

Unchanged	Objekt nebyl od svého připojení ke kontextu změněn.
Added	Objekt byl přidán a `SaveChanges` ještě nebyla zavolána. Po zavolání je stav změnen na `Unchanged`.
Modified	Jedna z vlastností byla modifikována a `SaveChanges` ještě nebyla zavolána. Po zavolání je stav změnen na `Unchanged`.
Deleted	Objekt byl odstraněn z kontextu a `SaveChanges` ještě nebyla zavolána. Po odstranění je stav změněn na `Detached`.
Detached	Objekt existuje, jeho stav ale není sledován. Tento stav má entita po svém vytvoření nebo po volání metody `Detach`.

Třídy entit

- vzniklé děděním z třídy `EntityObject`
  - `ObjectContext` sleduje vztahy mezi objekty a jejich stav
  - `EntityObject` má silnou závislost na EF
- POCO (Plain Old CLR Object)
  - obyčejná třída
  - neví nic o způsobu uložení
  - nemá automatickou detekci změn, EF musí použí snapshot
  - potřebuje více paměti

Třídy entit

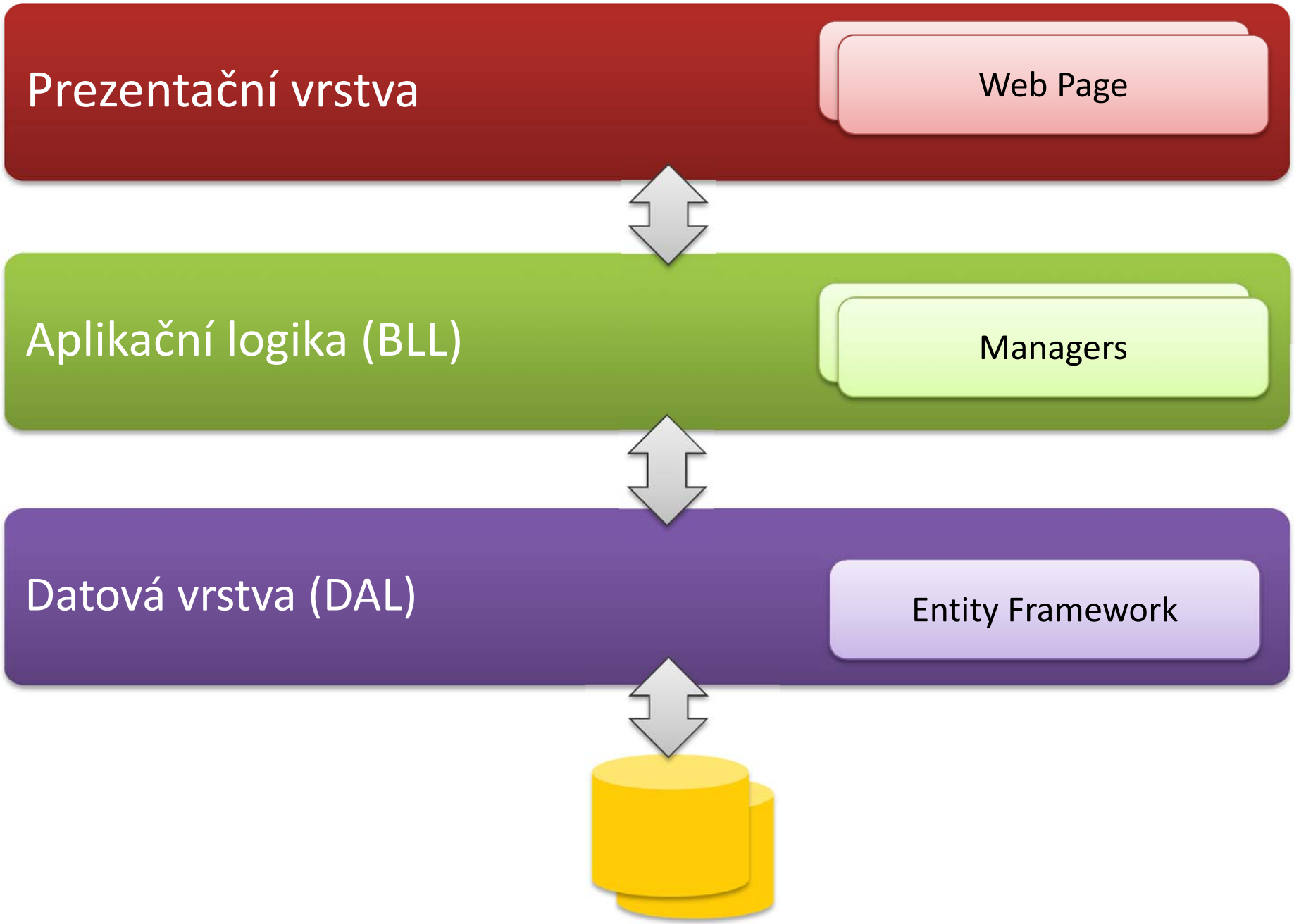
- POCO proxy
  - efektivní, instantní sledování změn a opožděné načtení
  - stejná funkcionalita jako Entity Object, ale stále POCO
  - třídy musí splňovat jisté požadavky
  - POCO proxies dědí od POCO tříd a jsou vytvářeny runtime
- Self-Tracking Entities
  - sledují svůj vlastní stav
  - nepotřebují Object Context
  - pro přenos entit bez přítomnosti Object Contextu

Přístup k datům

- object builder metody
  - třída `ObjectQuery`
  - vytvářená pro nějaký kontext
  - řetězení metod se převádí na dotaz pro databázi
- LINQ to Entities
  - LINQ pro přístup k entitám
  - převádí se na `ObjectQuery`
- Entity SQL
  - jazyk podobný SQL pro dotazy na entity a jejich vztahy
- Nativní SQL dotazy
  - metody kontextu `ExecuteStoreQuery` a `ExecutyStoreCommand`

Načtení objektů

1. entita je součástí dotazu
   - pokud použijete navigační vlastnost při vytvoření dotazu
2. explicitní načtení
   - voláním metody `Load` nad navigační vlastností nebo referencí nebo `LoadProperty` na `ObjectContext`
3. lazy loading
   - nastavení vlastnosti `LazyLoadingEnabled` kontaineru na `true`
   - objekty se dotáhnou při prvním použití
4. eager loading
   - zahrnutí do dotazu pomocí `Include`
   - pokud víte, že to budete potřebovat


Prezentační vrstva
Web Page
Aplikační logika (BLL)
Managers
Datová vrstva (DAL)
Entity Framework

Alternativy

- NHibernate
  - port Hibernate z Javy
  - http://nhforge.org/
- FluentNHibernate
  - alternativní konfigurace pro NHibernate
  - Nahrazuje XML statickým C# kódem
  - http://fluentnhibernate.org/
- SubSonic
  - http://subsonicproject.com/

XML

LINQ to XML

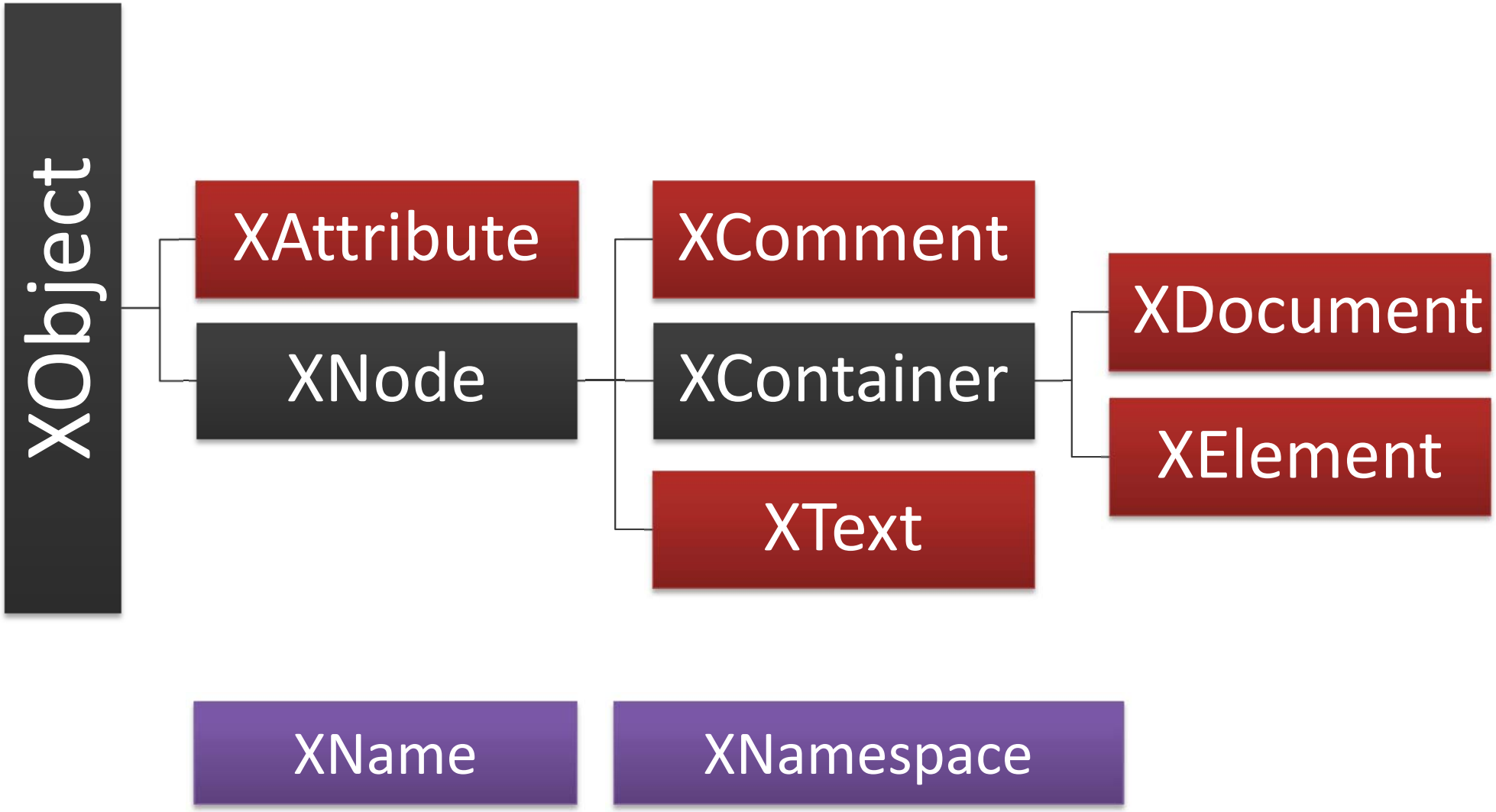
Přístup k XML v .NET

- Forward-only, non-cached
  - `XmlReader`, `XmlWriter`
- Serializace
  - XML serializace
- In-Memory
  - Document Object Model (W3C standard)
    - strom v paměti, `XmlDocument`
  - XPath
    - XPathDocument, XPathNavigator
  - DataSet
    - duální popis dat, jako XML dokument i jako relační model
- LINQ to XML

LINQ to XML

- in-memory přístup
- de-facto přepracovaný XML DOM
- lehčí a jednodušší model než DOM
- deklarativní i procedurální přístup
- pracuje jak s celým dokumentem tak i s fragmenty

Třídy LINQ to XML



Třídy LINQ to XML

- XObject
  - uzel nebo atribut XML dokumentu
  - vlastnosti `Document`, `Parent`
- XContainer
  - uzel obsahující další uzly
  - předek pro `XDocument` a `XElement`
- XAttribute
  - atribute XML dokumentu
  - vlastnosti `Name`, `Value`
- `XName`
  - název pro element, dovoluje složit s definicí jmenného prostoru v proměnné typu v `XNamespace` (pomocí operátoru +)

Vytvoření XML dokumentu

- načtením ze souboru nebo proudu
  - `XDocument.Load`
  - `XElement.Load`
- z `XmlReaderu`
  - `XElement.ReadFrom`
  - `XDocument.ReadFrom`
- z textového řetězce
  - `XDocument.Parse`
  - `XElement.Parse`

Vytvoření XML dokumentu

- funkcionální konstrukce
  - jedno z přetížení kostruktoru má jako své parametry jméno prvku a obsah elementu
  - obsah elementu může být:
    - `XObject`, `String`, `decimal`, `IEnumerable<T>`, …
  - navíc existuje implicitní konverze `String` na `XName`

```
public XElement(XName name,
                params Object[] content
)
public XDocument(params Object[] content)
```

```
XDocument doc
  = new XDocument("people",
      new XElement("person",
        new XAttribute("id", 10),
        new XElement("name", "Prófa")),
      new XElement("person",
        new XAttribute("id", 11),
        new XElement("name", "Šmudla"))
    );
```

```
<people>
  <person id="10"><name>Prófa</name></person>
  <person id="11"><name>Šmudla</name></person>
</people>
```

Klonování a připojování

- při přidávání nového uzlu do existujícího XML stromu
  a. pokud není element nemá žádného rodiče, pak se prostě připojí do XML stromu,
  b. jestliže už rodiče má, pak se vytvoří jeho klon a ten se připojí do XML stromu

Konverze v LINQ to XML

- LINQ to XML hodně používá explicitní konverze z `XElement` a `XAttribute` na základní datové typy a jejich `Nullable` varianty

- `int`
- `int?`
- `bool`
- `bool?`
- `string`
- `DateTime`
- `Guid`
- `Guid?`
- `double`
- …

age je nepovinný atribut elementu person:

```
<person id="10" age="25" />
```

```
XElement person = …
XAttribute ageAttr = person.Attribute("age")
int? age;
if (ageAttr == null)
  age = null;
else
  age = Int32.Parse(ageAttr.Value);
```

```
XElement person = …
int? age = (int?) person.Attribute("age");
```

Dotazování XML stromů

- pomocí metod si můžete zpřístupnit obsah XML dokumentu
  - `.Element`
  - `.Elements`
  - `.Attribute`
  - `.Attributes`
  - `.Ancestors`
  - `.Descendants`
  - …

Změna XML dokumentu

- změnu XML stromu provedete pomocí metod
  - `.Add`,`.AddBeforeSelf`
  - `.SetValue`,`.SetElementValue`,`.SetAttributeValue`
  - `.Remove`,`.RemoveAll`
  - `.ReplaceWith`
- uložení změn do souboru, proudu nebo `TextWriteru`
  - `.Save`