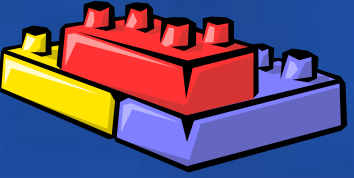


您的潜力，我们的动力

Microsoft
微软（中国）有限公司

Enterprise Library
Data Access Application Block

莫淘

Architect Evangelist

DPE Group

Microsoft Corporation

msdn
Microsoft® Developer Network
MSDN Web Casts

msdn

提要

- 介绍
- 使用Database Block的几个步骤
  - Defining your configuration
  - Creating an instance of the Database object
  - Executing SQL
- 几点进阶级应用
  - Selecting the right option for data access
  - Using transactions
- 高级话题
  - Key extensibility points

patterns & practices

Architecture Guidance for the Enterprise

Proven Based on field experience

Authoritative Offer the best advice available

Accurate Technically validated and tested

Actionable Provide the steps to success

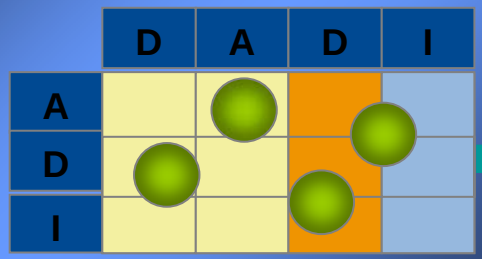
Relevant Address real-world problems based on customer scenarios

Available online: http://www.microsoft.com/practices

Books available: http://www.amazon.com/practices

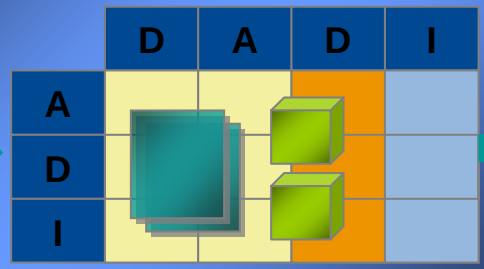
Patterns

Atomic solutions to recurring problems



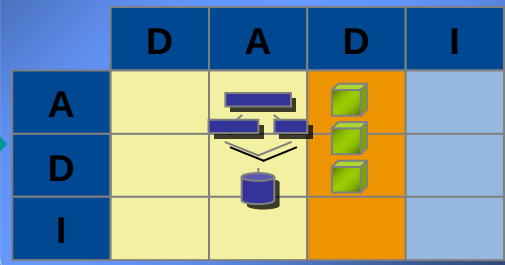
Application Blocks

Sub-system-level guidance for common services



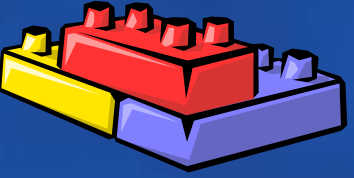
Reference Architectures

System-level guidance for common customer scenarios



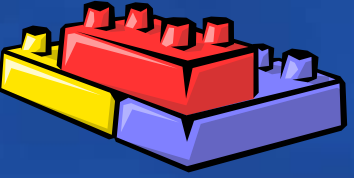
Guides

Guidance for broad horizontal topics such as security, performance, deployment and operations



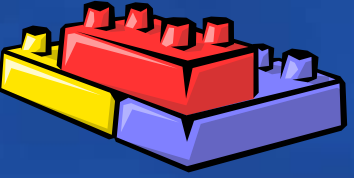
感同身受?

- 重复代码——Writing (cutting and pasting) the same data access code throughout your data access layer
- 硬编码——Matching stored procedure parameter definitions with the calling application code
- 对连接池的担心——Wondering if your code is properly closing connections
- 自己写组件简化对存储过程的调用——Writing a component to make it simpler to call stored procedures
- 如何存储连接串——Wrestling with where/how to store connection string information



我们需要的是...

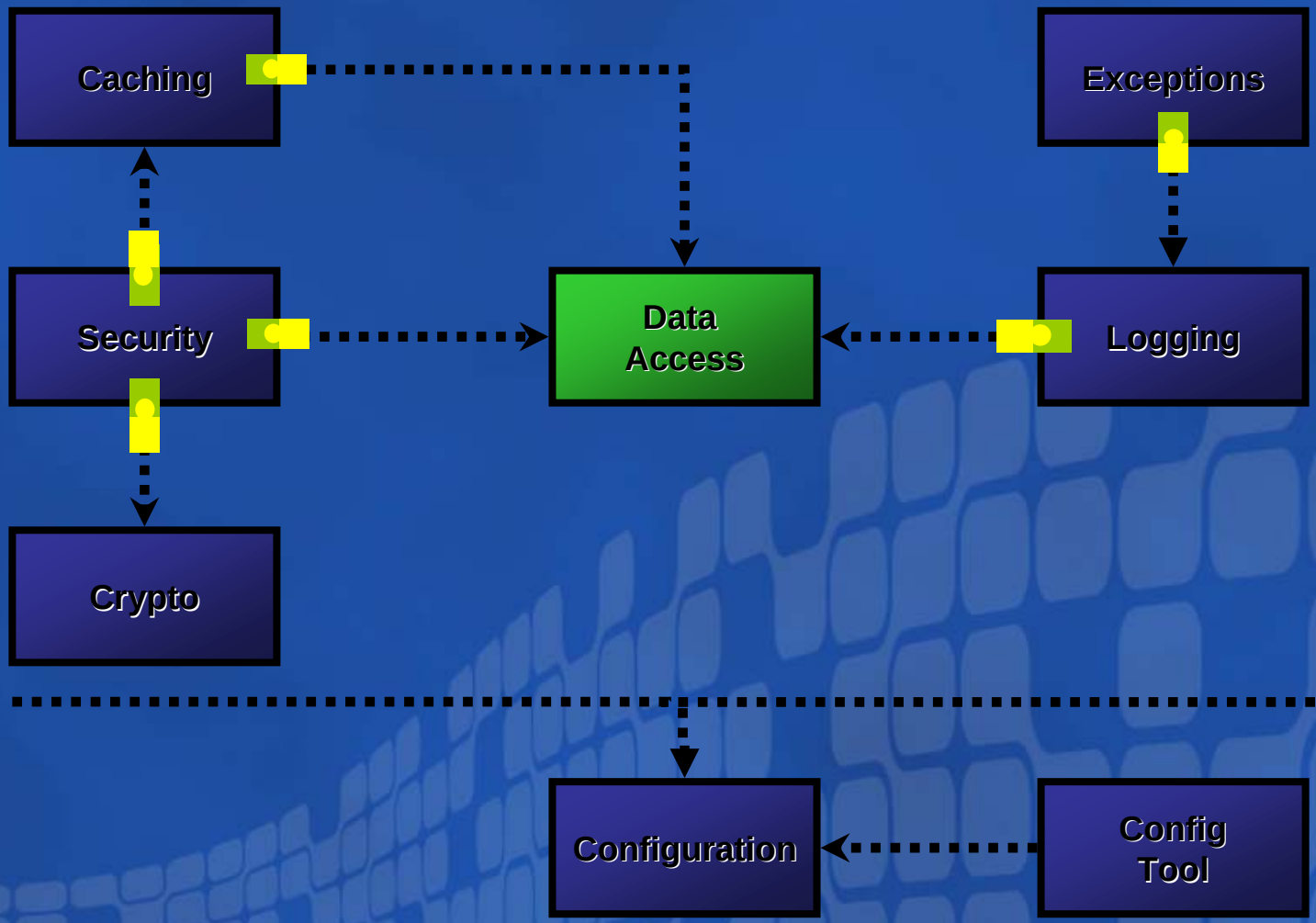
- 一种简单有效的方法——A simple and efficient way of working with commonly used databases
- 多种数据库之间的透明——Transparency when developing for multiple types of databases
- 逻辑和物理数据库的“松耦合”——A way to place an indirection between a logical database instance and a physical database instance
- 简单的校验方式——An easy way to adjust and validate the database configuration settings

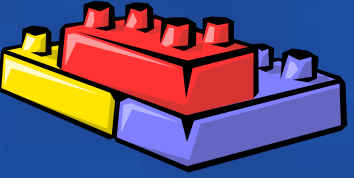


Data Access Application Block能够.....

- 提供最佳实践
  - Provides access to the most often used features of ADO.NET with applied best practices
- 改善一致性
  - Write code that works against multiple database brands (caveats apply!)
- 改善安全性
  - Leverages the configuration application block to securely store connection strings
- 改善易用性
  - Easily call a stored procedure with one line of code

Enterprise Library v1



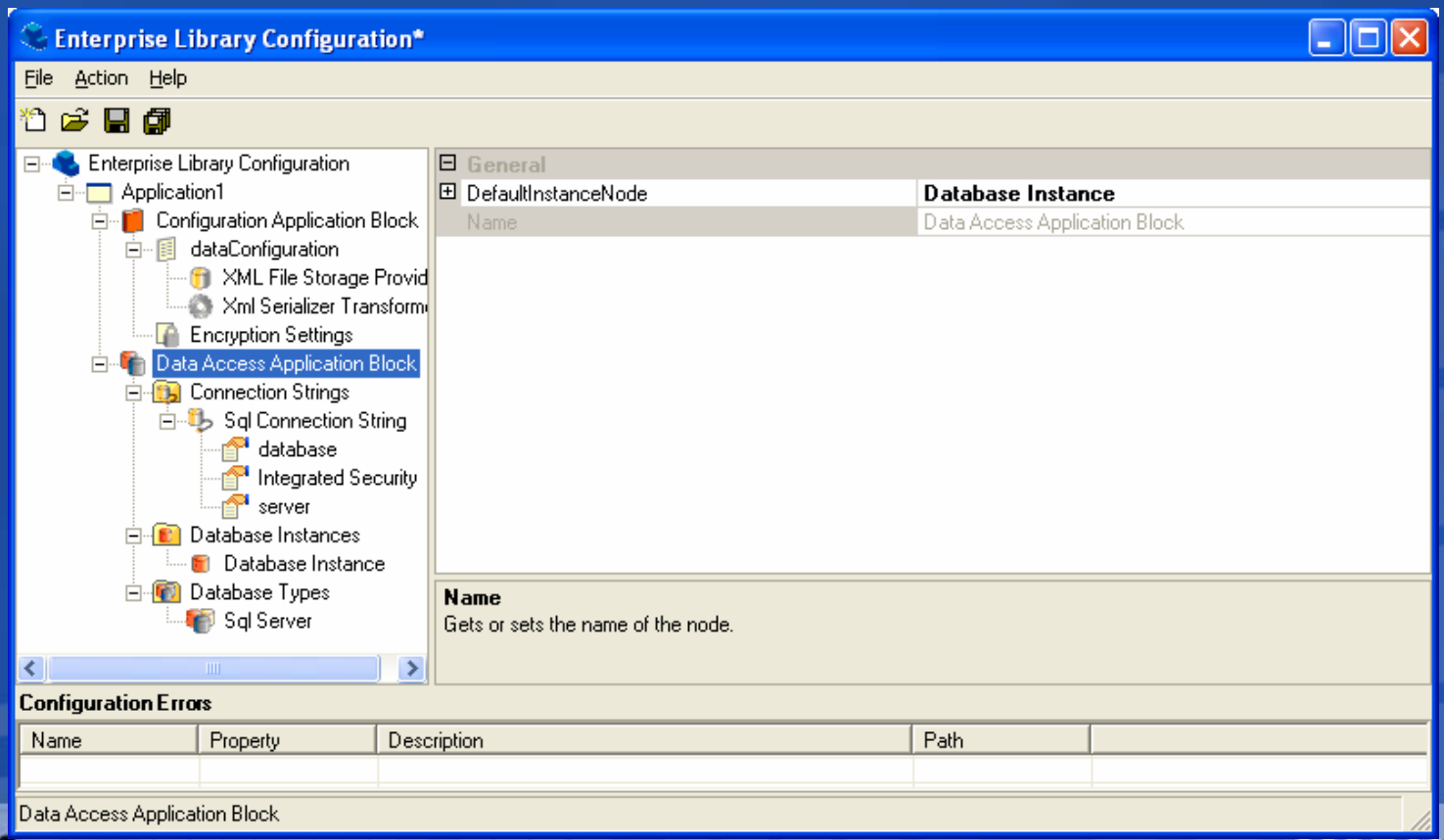


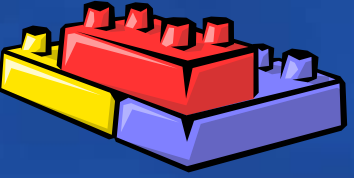
简单的几个步骤

...in 3 easy steps

Step 1: 定义配置

- You will need an app.config (or web.config) file for your application
- Use the Enterprise Library Configuration tool to create the configuration for the data access application block
  - Use a post-build step to copy config files to the runtime directory

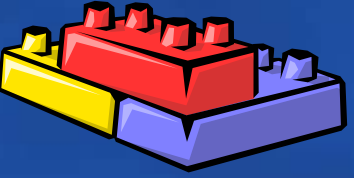




Step 2: 创建数据库实例

- Previous DAAB was accessed with static methods
- New Enterprise Library Data Access Application Block uses the Plugin [Fowler] pattern to create providers.
  - Allows us to support SQL Server, Oracle and IBM DB2

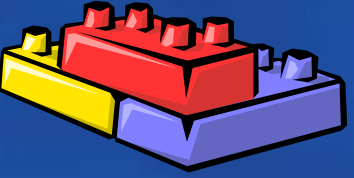
```
' Create the default database instance
Dim db As Database = DatabaseFactory.CreateDatabase()

' Use a named instance to map to configuration
Dim salesDb As Database =
DatabaseFactory.CreateDatabase("Sales")
```



Step 3: 执行SQL命令

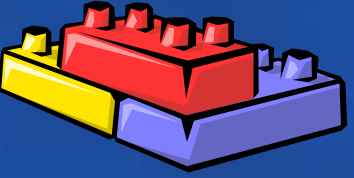
- 支持的 SQL
  - Dynamic SQL
  - Stored Procedure
  - Transaction Support
- 选择运行类型
  - Data Set
  - Data Reader
  - Scalar
- 数据库联接
  - Connections and Connection string managed automatically

```
// Invoke a SQL Command
productDataSet = db.ExecuteDataSet(CommandType.Text,
      "SELECT ProductID, ProductName FROM Products");
```



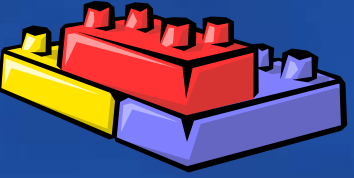
进阶级应用...

...this is where it gets interesting



威胁和对策

- 配置数据的泄露
  - The most sensitive configuration data used by data access code is the database connection string. If a compromised connection string includes a user name and password, the consequences can be greater still.
- 几个弱点
  - Use of SQL authentication, which requires credentials to be specified in the connection string
  - Embedded connection strings in code
  - Clear text connection strings in configuration files
  - Failure to encrypt a connection string
- 对策
  - Use Windows authentication so that connection strings do not contain credentials.
  - **Encrypt the connection strings** and restrict access to the encrypted data.

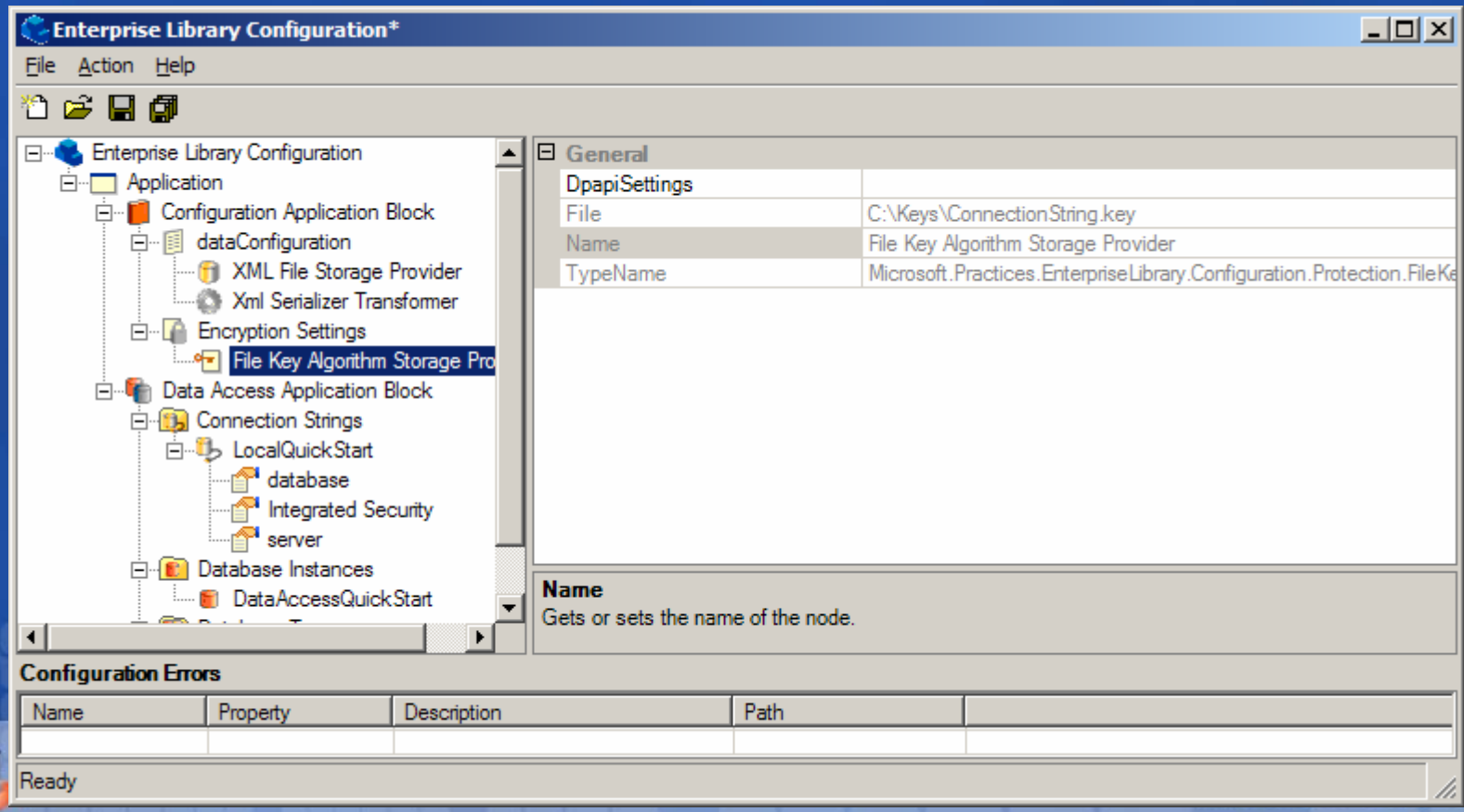


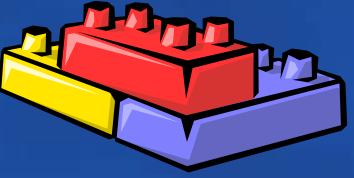
存储连接字符串

- 提供指导——Enterprise Library provides *applied guidance* through proven practices engineered in code
- 与程序块协作——Connection strings are managed through configuration with the Configuration Application Block
- 通过XML存储——With the default XML Storage Provider
  - Connection strings are saved in the file dataConfiguration.config
  - Configuration files are saved as plain text by default
- 使用解密机制——Enterprise Library cryptography functionality which can be used to encrypt the connection string automatically
  - In just 2 easy steps!

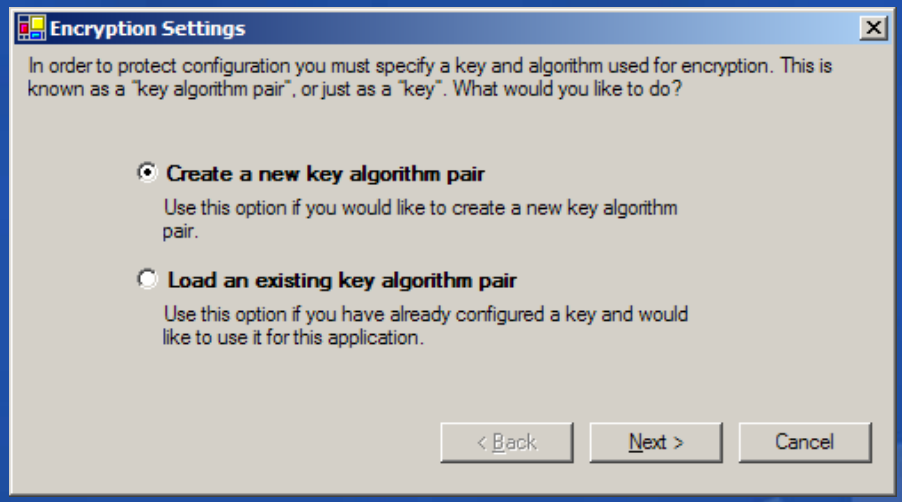
安全的连接串

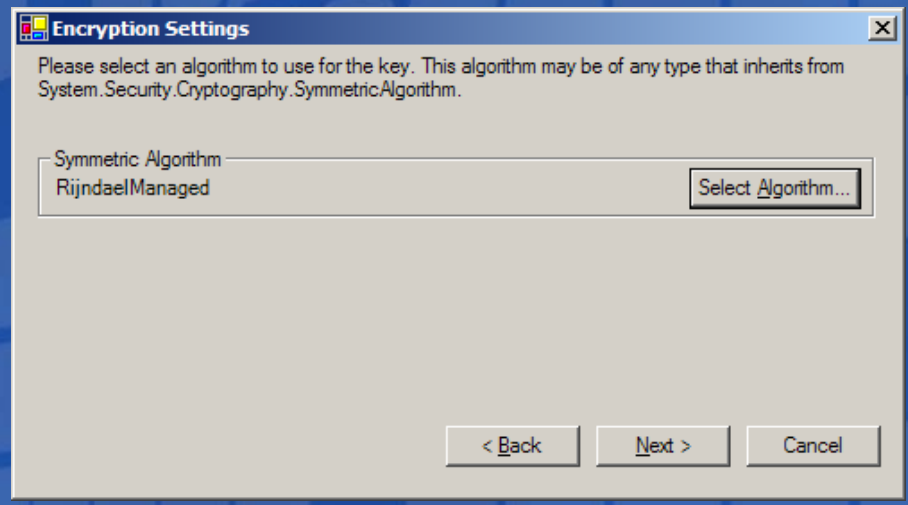
- The encryption configuration determines *how* the application block configuration will be encrypted

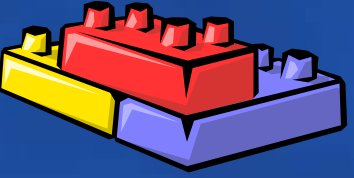




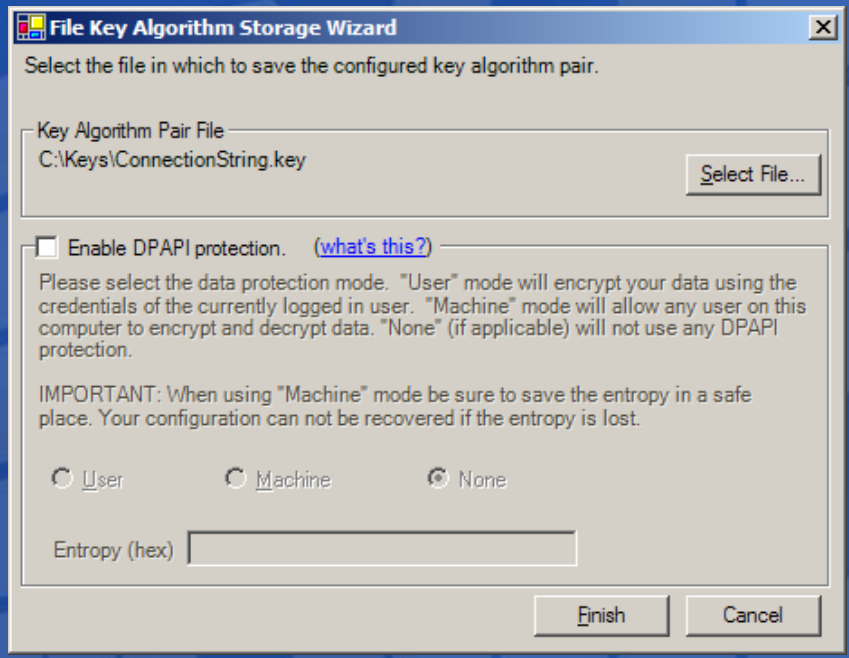
Step 1a: 设置加密





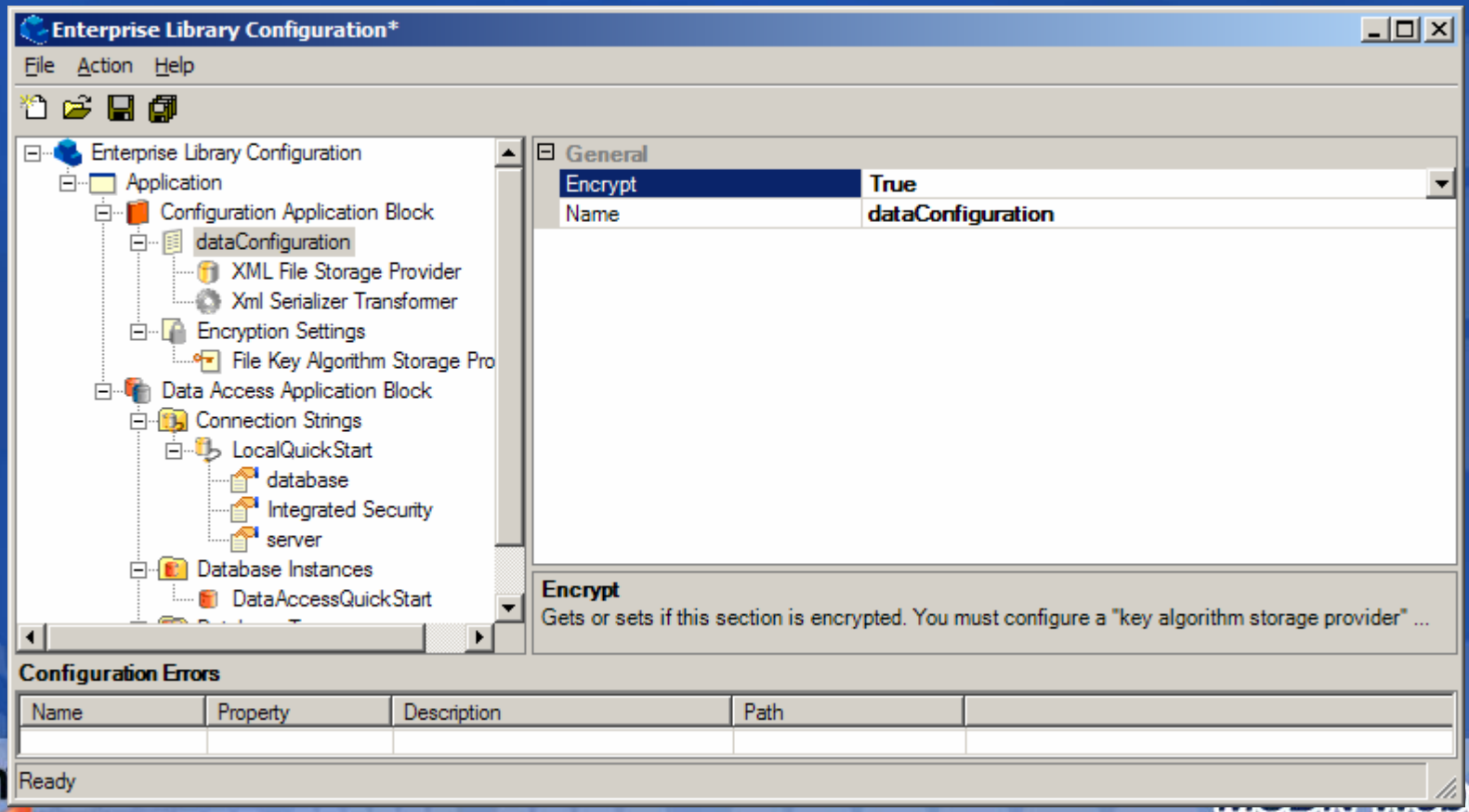


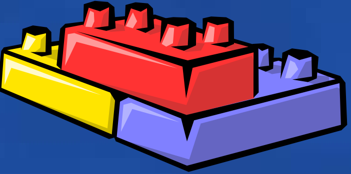
Step 1b: 设置加密



Step 2: 指定机密的部分

- *Whether* to encrypt configuration information is determined by each application block's configuration settings





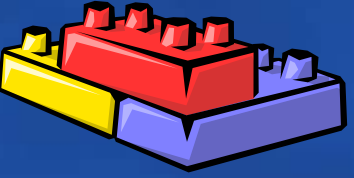
结果……

- From this…

```
<?xml version="1.0" encoding="utf-8"?>
<dataConfiguration>
  <xmlSerializerSection …>
    <enterpriseLibrary.databaseSettings …>
      <databaseTypes>
        <databaseType name="Sql Server" type="Microsoft.Practices.EnterpriseLibrary.Data.Sql.SqlDatabase,
Microsoft.Practices.EnterpriseLibrary.Data" />
      </databaseTypes>
      <instances>
        <instance name="DataAccessQuickStart" type="Sql Server" connectionString="LocalQuickStart" />
      </instances>
      <connectionStrings>
        <connectionString name="LocalQuickStart">
          <parameters>
            <parameter name="database" value="EntLibQuickStarts" isSensitive="false" />
            <parameter name="Integrated Security" value="True" isSensitive="false" />
            <parameter name="server" value="localhost" isSensitive="false" />
          </parameters>
        </connectionString>
      </connectionStrings>
    </enterpriseLibrary.databaseSettings>
  </xmlSerializerSection>
</dataConfiguration>
```

- To this…

[illegible]



使用什么校验方式

- 从安全性的角度——Security

  "When your application connects to a SQL Server database, you have a choice of Windows authentication or SQL authentication. **Windows authentication is more secure** because it does not send credentials over the network"

- 从性能的角度——Performance

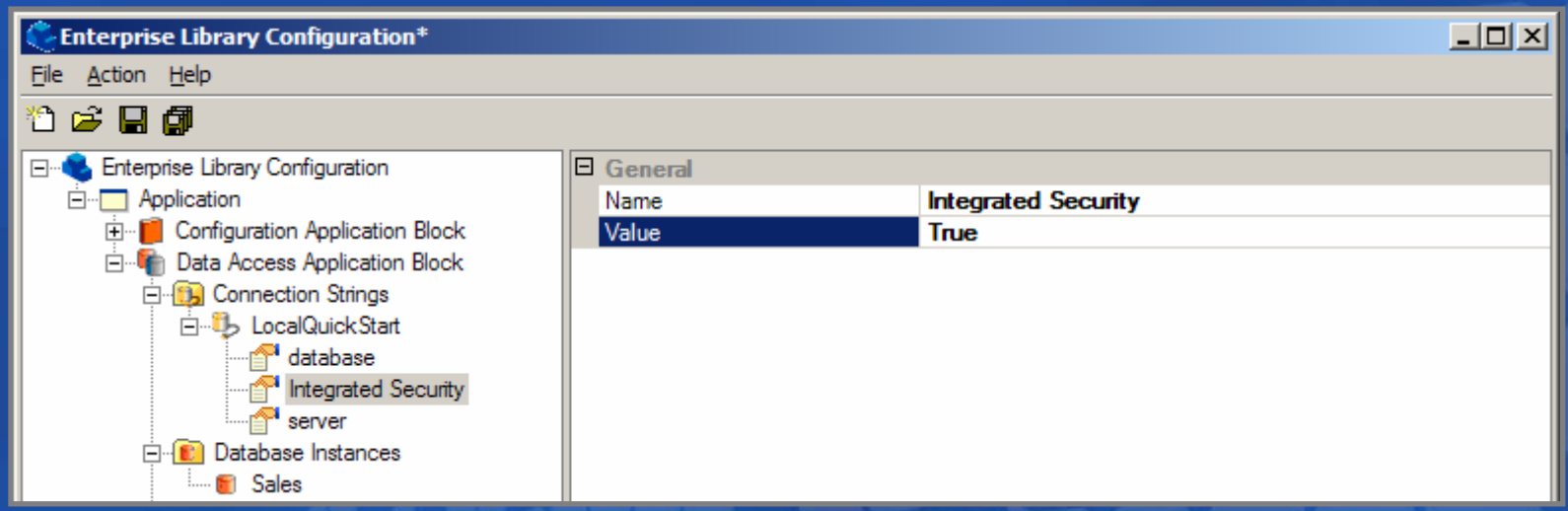
  "From a security perspective, you should **use Windows authentication**. From a performance perspective, you should use a **fixed service account and avoid impersonation**. The fixed service account is typically the process account of the application. By using a fixed service account and a consistent connection string, you help ensure that database connections are pooled efficiently. You also help ensure that the database connections are shared by multiple clients. Using a fixed service account and a consistent connection string is a major factor in helping application scalability."

配置使用 Windows 验证

- Desired connection string

```
"database=Sales;Integrated Security=True;server=localhost;"
```

- Using the Configuration Console



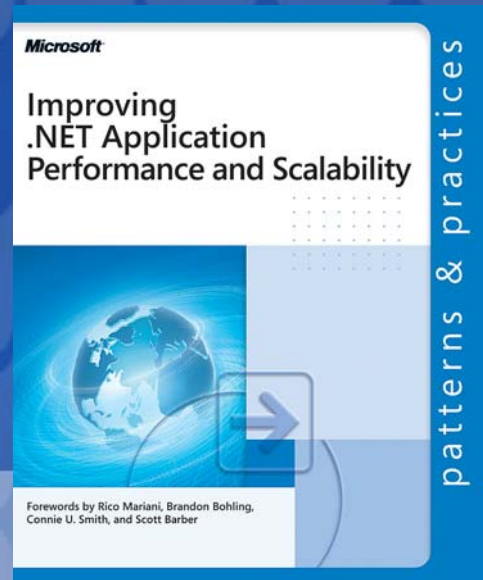
- Sample code

```
Database db = DatabaseFactory.CreateDatabase(“Sales”);
```

改善.NET应用的性能

您的潜力，我们的动力
Microsoft
微软(中国)有限公司

- 从数据库联接的角度——**Connections**
- Database connections are an expensive and limited resource. Your approach to connection management can significantly affect the overall performance and scalability of your application. Issues to consider include acquiring and releasing connections, pooling, and authentication. To improve database connection performance and scalability, apply the following strategies to your connection management policy:
  - **Open and close the connection in the method**.
  - **Explicitly close connections**.
  - **Pool connections**



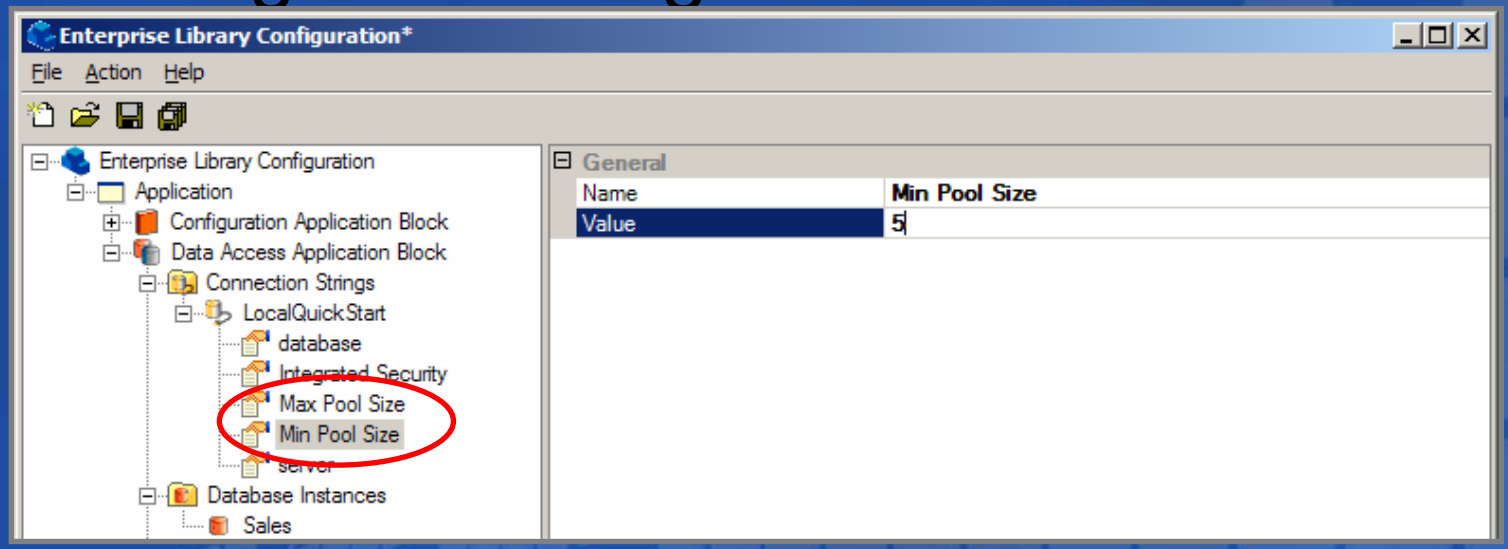
Improving .NET Application Performance and Scalability
Chapter 12 – Improving ADO.NET Performance

Configuring Connection Pooling

- Desired connection string

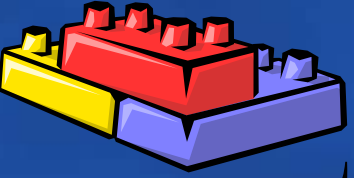
```
"database=EntLibQuickStarts;Integrated Security=True;
Max Pool Size=75;Min Pool Size=5;server=localhost;"`
```

- Using the Configuration Console



- Sample code

```
Database db = DatabaseFactory.CreateDatabase(“Sales”);
```



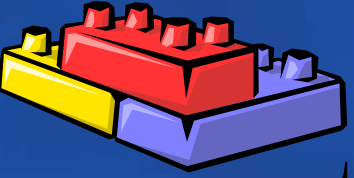
必要时使用，尽早释放(1)

Your application should share expensive resources efficiently by acquiring the resources late, and then releasing them as early as possible. To do so:

- Open database connections right when you need them. Close the database connections as soon as you are finished. Do not open them early, and do not hold them open across calls.
- Acquire locks late, and release them early.

Close Disposable Resources

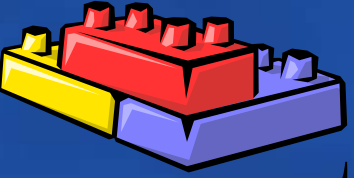
- Usually, disposable resources are represented by objects that provide a **Dispose** method or a **Close** method. Make sure that you call one of these methods as soon as you are finished with the resource.



必要时使用，尽早释放(2)

- Enterprise Library Data Access Application Block makes implementation of this principle easy and automatic
  - Whenever possible, the application block handles connection management. The application block's method opens a connection and closes it prior to returning. This reduces both the amount of client code required and the possibility of leaving connections open

```
Public Function GetProducts() As DataSet
       Dim db As Database = DatabaseFactory.CreateDatabase()
       Return db.ExecuteDataSet("GetProductsList")
      ' Connection is closed automatically
End Function
```



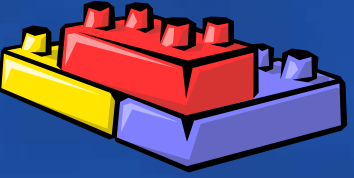
必要时使用，尽早释放(3)

- In the case of the **ExecuteDataReader** method, the **DataReader** object is executed using the **CommandBehavior.CloseConnection** method, which automatically closes connections when the **DataReader** is closed

```
Database db = DatabaseFactory.CreateDatabase();
using (IDataReader dataReader = db.ExecuteReader("GetCust"))
{
  while (dataReader.Read())
  {  // ...  }
}
// DataReader is disposed, causing the connection to close

' VB.NET
Dim dataReader as DataReader
dataReader = db.ExecuteReader("GetCust")
' You must explicitly close with VB.NET
dataReader.Close()
```

调用存储过程与调用SQL的比较

- You should **use stored procedures instead of embedded SQL** statements for a number of reasons:
  - Stored procedures generally result in improved performance because the database can optimize the data access plan used by the procedure and cache it for subsequent reuse.
  - Stored procedures can be individually secured within the database. A client can be granted permissions to execute a stored procedure without having any permissions on the underlying tables.
  - Stored procedures result in easier maintenance because it is generally easier to modify a stored procedure than it is to change a hard-coded SQL statement within a deployed component.
  - Stored procedures add an extra level of abstraction from the underlying database schema. The client of the stored procedure is isolated from the implementation details of the stored procedure and from the underlying schema.
  - Stored procedures can reduce network traffic, because SQL statements can be executed in batches rather than sending multiple requests from the client.



使用存储过程(1)

- Create a Database Object

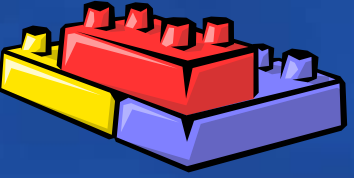
```
Dim db As Database = DatabaseFactory.CreateDatabase(“Sales”)
```

- Option 1: Pass the stored procedure name to the method

```
Dim ds As DataSet = db.ExecuteDataSet("GetAllProducts")
```

- Option 2: Create a command wrapper object

```
Dim dbc As DBCommandWrapper = _
  db.GetStoredProcCommandWrapper("GetAllProducts")

Dim ds As DataSet =  db.ExecuteDataSet(dbc)
```



使用存储过程(2)

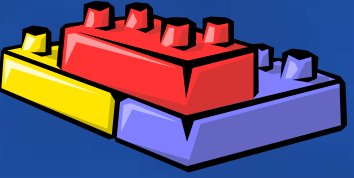
——Input变量传递

- Input-only parameters

```
Dim ds As DataSet = db.ExecuteDataSet("GetProductsByCategory“, 12)
```

- Input-only parameters with command

```
Dim dbc As DBCommandWrapper = _
  db.GetStoredProcCommandWrapper("GetProductsByCategory")

dbCommandWrapper.AddInParameter("@CategoryID", DbType.Int32, 12)

Dim ds As DataSet =  db.ExecuteDataSet(dbc)
```

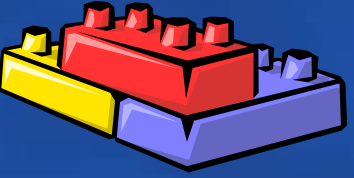


使用存储过程(2)

——output变量传递

- Output parameters require command wrapper

```
Dim dbc As DBCommandWrapper = _
  db.GetStoredProcCommandWrapper("GetProductDetails")

dbCommandWrapper.AddInParameter("@ProductID", DbType.Int32, productID)

dbCommandWrapper.AddOutParameter("@ProductName", DbType.String, 50)

dbCommandWrapper.AddOutParameter("@UnitPrice", DbType.Currency, 8)

Dim ds As DataSet =  db.ExecuteDataSet(dbc)
```

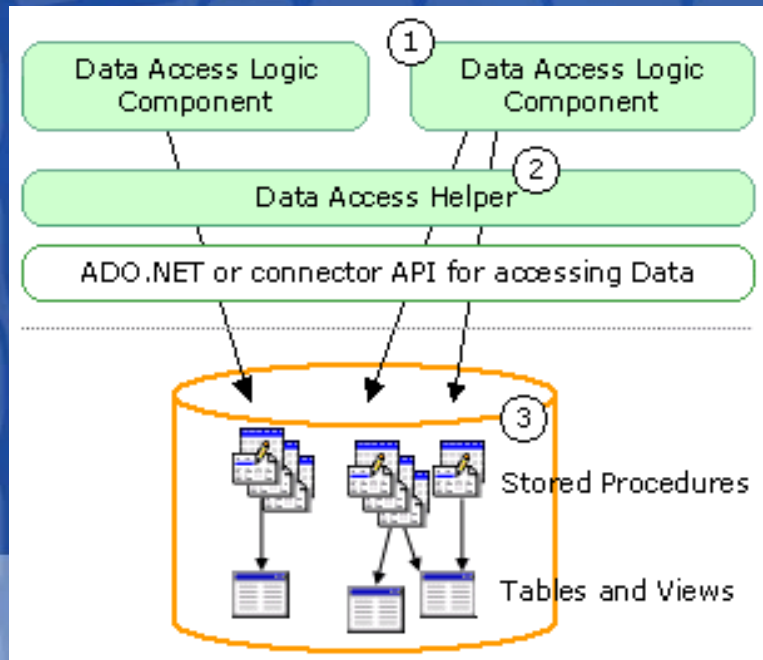


您的潜力，我们的动力

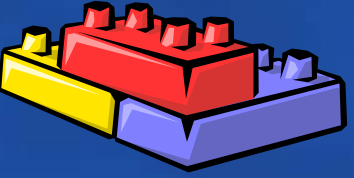
Microsoft
微软(中国)有限公司

Data Layer Helper

- "When your application contains multiple data access logic components, it can be useful to use a generic data access helper component to manage database connections, execute commands, cache parameters, and so on. The data access logic components provide the logic required to access specific business data, while the generic data access helper utility component centralizes data access API development and data connection configuration, and helps to reduce code duplication. A well designed data access helper component should have no negative impact on performance, and provides a central place for data access tuning and optimization."



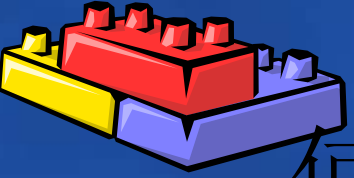
Application Architecture for .NET
[Designing the Components of an Application or Service]



使用 DataSet 返回多行

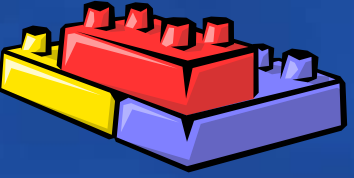

```
Public Function GetProductsInCategory(ByRef Category As Integer) As DataSet

    Dim db As Database = DatabaseFactory.CreateDatabase()

    ' Invoke the stored procedure with one line of code!
    return db.ExecuteDataSet("GetProductsByCategory", Category)

End Function
```

- ExecuteDataSet creates a new DataSet
- LoadDataSet can be used to fill an existing DataSet



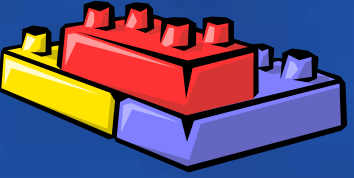
使用 DataReader 返回多行


```
private void DisplayProducts(int category)
{
      Database db = DatabaseFactory.CreateDatabase("Sales");

      using (IDataReader dataReader =
       db.ExecuteReader("GetProductsByCategory", category))
      {
            resultsDataGrid.DataSource = dataReader;
            resultsDataGrid.DataBind();
      }
}
```



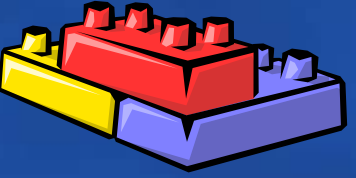
几种方法的比较 (1)

- 推荐使用 **DataSet** 的情形:
  - You require a disconnected memory-resident cache of data, so that you can pass it to another component or tier within your application
  - You require an in-memory relational view of the data for XML or non-XML manipulation
  - You are working with data retrieved from multiple data sources, such as multiple databases, tables, or files
  - You want to update some or all of the retrieved rows and submit updates to the database
  - You want to perform data binding against a control that requires a data source that supports **IList**



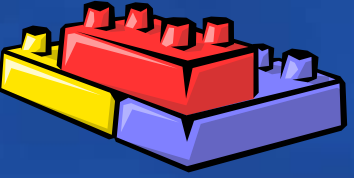
几种方法的比较(2)

- 推荐使用 **DataReader**的情形:
  - You are dealing with large volumes of data—too much to maintain in a single cache
  - You want to reduce the memory footprint of your application
  - You want to avoid the object creation overhead associated with the **DataSet**
  - You want to perform data binding with a control that supports a data source that implements **IEnumerable**
  - You wish to streamline and optimize your data access
  - You are reading rows containing binary large object (BLOB) columns



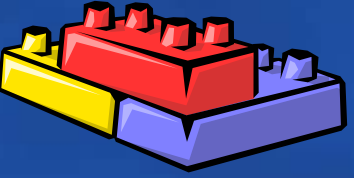
使用 Output 变量传递数值

```
public string GetProductDetails(int productID)
{
    Database db = DatabaseFactory.CreateDatabase();

    string sqlCommand = "GetProductDetails";
    DBCommandWrapper dbCommandWrapper =

db.GetStoredProcCommandWrapper(sqlCommand);

    dbCommandWrapper.AddInParameter("@ProductID", DbType.Int32, productID);
    dbCommandWrapper.AddOutParameter("@ProductName", DbType.String, 50);
    dbCommandWrapper.AddOutParameter("@UnitPrice", DbType.Currency, 8);

    db.ExecuteNonQuery(dbCommandWrapper);
    string results = string.Format(CultureInfo.CurrentCulture, "{0}, {1},
        {2:C} ", dbCommandWrapper.GetParameterValue("@ProductID"),
        dbCommandWrapper.GetParameterValue("@ProductName"),
        dbCommandWrapper.GetParameterValue("@UnitPrice"));
    return results;
}
```



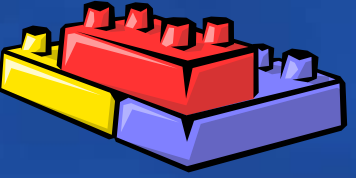
返回单行数据的几种方法间的比较

- You can use a **DataSet** or **DataTable.** However, unless you specifically require **DataSet/DataTable** functionality (for example, with data binding), you should avoid creating these objects.
- If you need to retrieve a single row, use one of the following options:
  1. Use **ExecuteNonQuery** to obtain stored procedure output parameters
  2. Use **ExecuteDataReader** to return an object that implements IDataReader



返回单一值

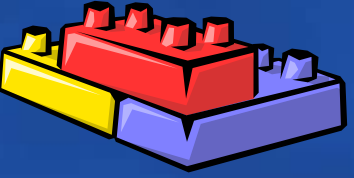

```
public string GetProductName(int productID)
{
    Database db = DatabaseFactory.CreateDatabase();

    string productName =
       (string) db.ExecuteScalar("GetProductName", productID);

    return productName;
}
```



返回单值数据的几种方法间的比较

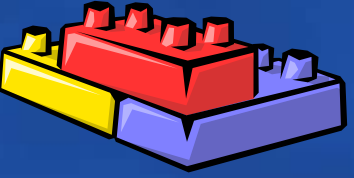
1. Use the **ExecuteScalar** method with a stored procedure
2. Use **ExecuteNonQuery** with stored procedure output or return parameter
3. Use **ExecuteDataReader**

- From strictly a performance perspective, you should use a **ExecuteNonQuery** with stored procedure output or return parameter. Tests have shown that the stored procedure approach offers consistent performance across low and high-stress conditions (from fewer than 100 simultaneous browser connections to 200 browser connections).



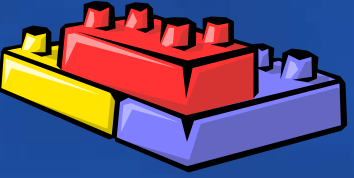
利用事务更改数据


```
Database db = DatabaseFactory.CreateDatabase();

using (IDbConnection connection = db.GetConnection())
{
  connection.Open();
  IDbTransaction transaction = connection.BeginTransaction();

  try
  {
      db.ExecuteNonQuery(transaction, "CreditAccount", srcAccount, amount );
      db.ExecuteNonQuery(transaction, "DebitAccount", dstAccount, amount );

      transaction.Commit();
  }
  catch
  {
       transaction.Rollback();
  }

connection.Close();
```



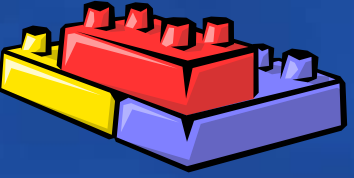
事务

- Perform transactions only when you need to acquire locks across a set of operations and need to enforce ACID rules
- Keep transactions as short as possible to minimize the amount of time that you hold database locks
- Never place a client in control of transaction lifetime
- Don't use a transaction for an individual SQL statement. SQL Server automatically runs each statement as an individual transaction
- Use System.EnterpriseServices to provide automatic distributed transaction support



用Dataset执行批量更新

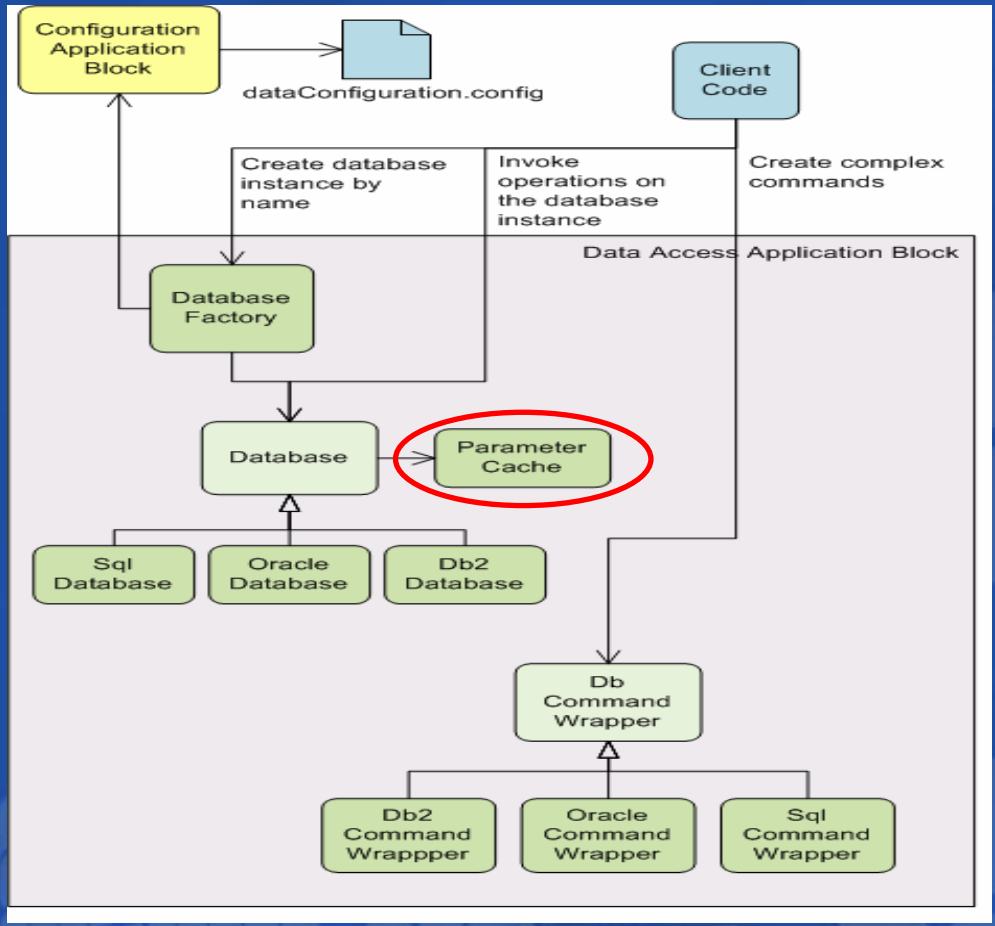

```
DBCommandWrapper iCmd = db.GetStoredProcCommandWrapper("AddProduct");
iCmd.AddInParameter("@ProductName", DbType.String,"ProductName",...);
iCmd.AddInParameter("@CategoryID", DbType.Int32, "CategoryID",...);
iCmd.AddInParameter("@UnitPrice", DbType.Currency, "UnitPrice",...);

DBCommandWrapper dCmd = db.GetStoredProcCommandWrapper("DeleteProduct");
dCmd.AddInParameter("@ProductID", DbType.Int32, "ProductID",...);

DBCommandWrapper uCmd = db.GetStoredProcCommandWrapper("UpdateProduct");
uCmd.AddInParameter("@ProductID", DbType.Int32, "ProductID",...);
uCmd.AddInParameter("@ProductName", DbType.String, "ProductName",...);
uCmd.AddInParameter("@LastUpdate", DbType.DateTime, "LastUpdate",...);

int rowsAffected = db.UpdateDataSet(productsDataSet, "Products",
        insertCommandWrapper, updateCommandWrapper, deleteCommandWrapper,
        UpdateBehavior.Standard);
```

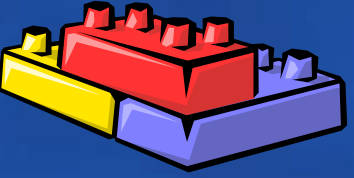


透明性和便携性

- Data Access Block includes classes to abstract database brands
- Supports
  - SQL Server, Oracle and IBM DB2
  - Build your own provider
- Use the same API for each database type
  - Makes developers portable
- You can build portable code
  - Test carefully – SQL implementation differences exist

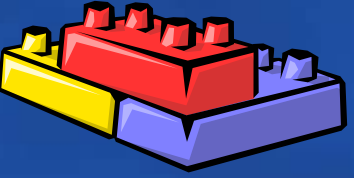
Data Access Diagram





关键特性

- Custom database system
  - Create a new database class that derives from **Database**
  - Create a new command wrapper class that derives from **DBCommandWrapper**
- Plus…
  - Anything and everything – you have the source code!
  - Please post extensions and suggestions to the community
    - http://workspaces.gotdotnet.com/entlib



Additional Resources

- .NET Data Access Architecture Guide
- Improving Web Application Security
- Improving .NET Application Performance and Scalability
- Application Architecture for .NET
- PatternShare.org
- Enterprise Library Community
  http://go.microsoft.com/fwlink/?linkid=39209&clcid=0x09
- www.ronjacobs.com
  - Slides
  - Tech Tips
  - Podcasts

Announcing: Enterprise Library 1.0

http://www.microsoft.com/practices

Download it Today!

Microsoft®

Your potential. Our passion.™

http://www.microsoft.com/practices