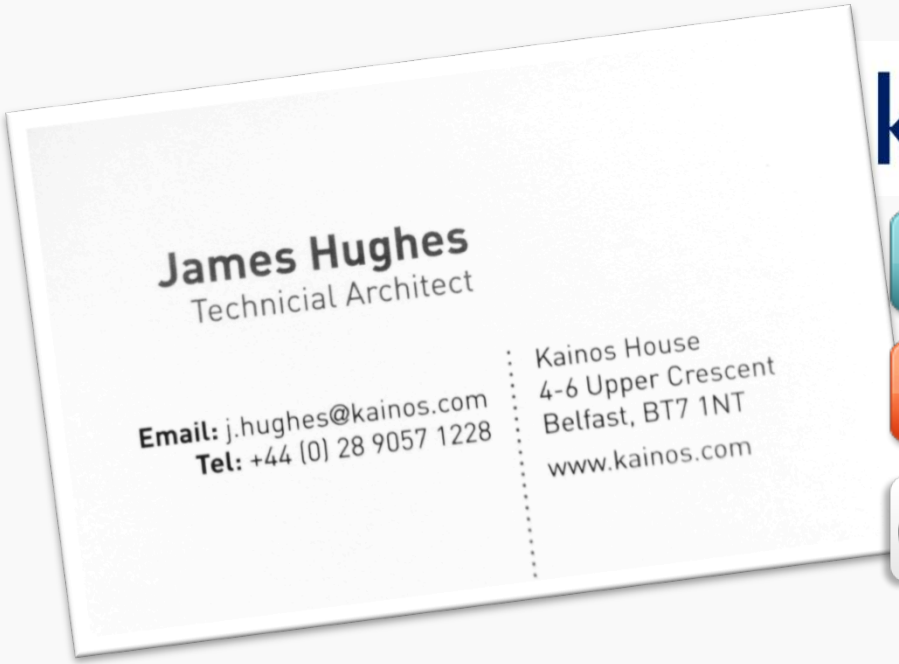
Death to ORMs in .NET

“A Propos de Moi”



kainos®

@kouphax

http://yobriefca.se

http://github.com/kouphax

Disclaimer

This presentation is full of opinion. My opinion. The fact is ORMs have been around for a long time and will be for a while yet. If things were as simple as I portray them in this presentation you wouldn't be using them today. But you probably are.

All I ask is that you never blindly use an ORM or any technology without prior **brain thinking™**

On with the show…

ORMs are for Life…

- So you've adopted an ORM, possibly
  - NHibernate
  - MyBatis
  - Entity Framework
- The samples, blogs and demos promise you a better life, your "hello world" grade apps are easy and it adds another shiny buzzword to your CV (Resume Driven Development)
- You dive right in investing a heap of development cycles, things start out well…

Suddenly and Without Warning…

The Cracks Start To Show

- Leaky Abstractions
- Bugger to debug
- Forced to work for the ORM rather than the ORM working for you
- DBAs begin to call you names
- Code generation and **heavy** use of model-polluting attributes bog you down
- Cost of change is high
- Overreliance on ORMs make you dumb


BEEEEEEE
MMMMMMINE
SLOTH
LOVE

ORMs in .NET – More Like WTFs

- NHibernate
  - HQL – **WTF?**
- MyBatis
  - XML as a medium for SQL? – **WTF?**
- Entity Framework
  - Great demoware but the reality is more like **WTF?**

Lets Be Honest

- How many features of a heavyweight ORM do you really use regularly? 80%? 20%? 10%?
- How many of these features exist purely to support the abstraction?
- How difficult is the mapping between DB's and objects really?
- Isn't SQL the best DSL for working with data in a relational database?

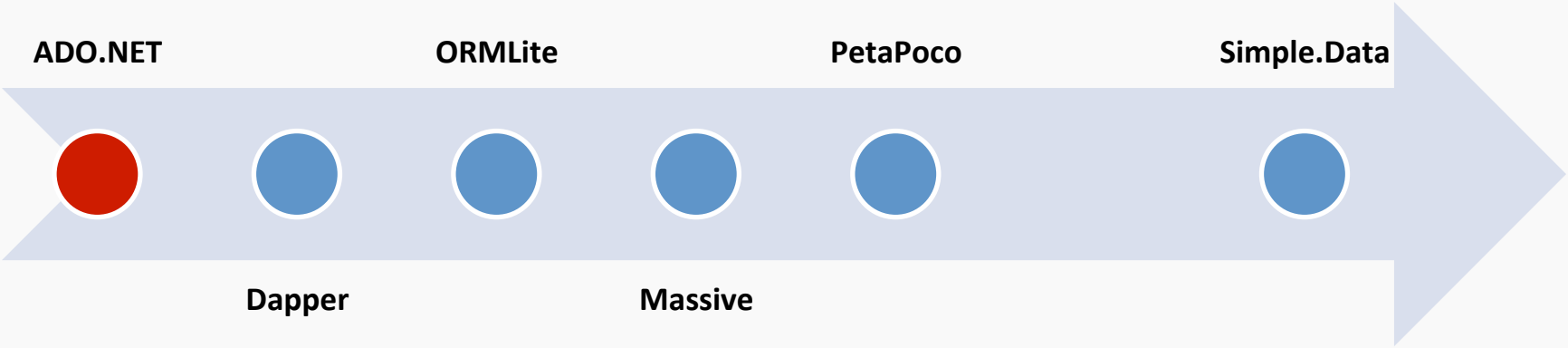
So why is this abstraction necessary?

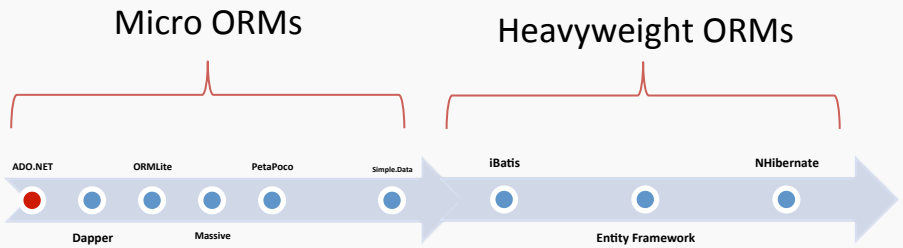
It isn’t.

- Historically speaking the alternatives were pretty slim. Most people would need to roll their own project specific data access layer
- Many ORMs grew out of that sort of thinking but from a community that thought XML was sexy
  - Some people are weird
- The world is a saner place these days.
  - Reflection is faster
  - Statically typed languages have embraced dynamics, and,
  - most people realise XML is a noisy medium for almost everything

The Advent of Micro ORMs

Level of Abstraction





Dapper

Dapper

- Created by Sam Saffron of Stackoverflow
- Single file (available on Nuget)
- Set of extension methods to IDbConnection
- Supports most .NET ADO providers
  - SQL Server, Oracle, SQLite, MySQL, SQL CE
- Works on boths POCOs and dynamic objects
- Pretty close to the metal
  - No connection handling
  - No helpers for INSERT, DELETE, UPDATE etc.
- Community extensions for CRUD operations
- .NET 3.5 support

Demo

Queries

```
using (SqlConnection conn = new SqlConnection(ConnectionString))
{
    conn.Open();
    Author a = conn.Query<Author>(
        "SELECT * FROM Authors WHERE Id = @Id",
        new { Id = 1 }).First();
}
```

```
IEnumerable<Author> dapperA = conn.Query<Author>(
    "SELECT * FROM Authors WHERE Username LIKE @PartialUsername",
    new { PartialUsername = "%example.com" });
```

```
IEnumerable<dynamic> rows = conn.Query(
    "SELECT * FROM Authors WHERE Username LIKE @PartialUsername",
    new { PartialUsername = "%example.com" });
```

```
Author author = conn.Get<Author>(1);
```

Inserts, Updates & Deletes

```
int count = conn.Execute(
    "INSERT Authors(Username, FullName, CreatedDate) " +
    "VALUES (@Username, @FullName, GETDATE())",
    new { Username = "james@dapper.net", FullName = "James Hughes" });
```

```
int count = conn.Execute(
    "UPDATE Authors SET FullName = @FullName WHERE Id = @Id",
    new { FullName = "James Hughes", Id = 1 });
```

```
int count = conn.Execute(
    "DELETE FROM Authors WHERE Id = @Id",
    new { Id = 1 });
```

Inserts, Updates & Deletes (CE)

```
long id = conn.Insert(
    new Author { Username = "james@test.net", FullName = "J Hughes" });
```

```
bool success = conn.Update(entity);
```

```
bool success = conn.Delete(entity);
```

And the rest…

- Dapper is very close to the metal
- Very little in the way of providing helpers for 1..N, N..N relationships
- Dapper Community Extensions bridge *some* of the mental gap for people not used to SQL

ORMLite

ORMLite

- Part of the ServiceStack technology suite
- Set of CRUD extensions around System.Data.* namespace
- Currently supports SQLite and SQL Server
- Maps data to and from concrete POCOs
  - Via convention
  - Via attributes on the POCO – preferred
  - Via raw SQL
- Relies heavily on POCO attribution
- Supports .NET 3.5

Demo

Configuration

```
[ServiceStack.DataAnnotations.Alias("Authors")]
```

```
OrmLiteConfig.DialectProvider = new SqlServerOrmLiteDialectProvider();
```

```
using (IDbConnection db = "...".OpenDbConnection())
using (IDbCommand cmd = db.CreateCommand())
{


}
```

```
IDbConnectionFactory dbFactory = new OrmLiteConnectionFactory("...",
        SqlServerOrmLiteDialectProvider.Instance);
```

Queries

```
Author rows = cmd.GetById<Author>(1);
```

```
List<Author> rows = cmd.Select<Author>(
    "Username LIKE {0}",
    "%example.com");
```

```
Author author = dbFactory.Exec(dbCmd => dbCmd.GetById<Author>(1));
```

```
List<Author> rows = dbFactory.Exec(dbCmd =>
        dbCmd.Select<Author>("Username LIKE {0}", "%example.com"));
```

Inserts, Updates & Deletes

```
cmd.Insert<Author>(new Author
{
    Username = "james@dapper.net",
    FullName = "James Hughes",
    CreatedDate = DateTime.Now
});
```

```
cmd.Update(new Author
{
    Id = 7,
    Username = "james@dapper.net",
    FullName = "James Hughes",
    CreatedDate = DateTime.Now
});
```

```
cmd.DeleteById<Author>(5);
```

```
cmd.Delete<Author>("Id = @0", 6);
```

```
cmd.Delete(new Author { Id = 7 });
```

And the rest…

- ORMLite also allows you to create tables from entities

```
cmd.CreateTable<Author>(true);
```

```
cmd.CreateTables(true, typeof(Author), typeof(Book));
```

- Each model must have an Id property which must be its primary key.
- Integrates into the rest of the ServiceStack technology stack

Massive

Massive

- Created by Rob Conrey
- Single file (available on Nuget)
- Makes extensive use of dynamics
- No mapping to POCOs for queries
- Prefers to wrap tables rather than entities
  - Provides a single point of configuration
  - No need for attributes
- OOTB Paging support
- .NET 4 Only

Demo

Creating the Dynamic Model

```
public class Authors : Massive.DynamicModel
{
    public Authors()
        : base("DefaultConnectionString")
    {
        PrimaryKeyField = "Id";
    }
}
```

Queries

```
Authors table = new Authors();
dynamic result = table.Single(1);
```

```
var tbl = new Authors();
IList<dynamic> dyn = tbl.Query(
    "SELECT * FROM Authors WHERE Username LIKE @0",
    "%example.com");
```

```
dynamic db = new Products();
var products = db.FindBy(
    CategoryID:8,
    UnitPrice:100
);
```

Inserts, Updates & Deletes

```
var tbl = new Authors();
var x = tbl.Insert(new
{
    Username = "james@dapper.net",
    FullName = "James Hughes",
    CreatedDate = DateTime.Now
});
```

```
var tbl = new Authors();
var x = tbl.Update(new
{
    FullName = "James Hughes"
}, 1);
```

```
var tbl = new Authors();
var x = tbl.Delete(2);
```

And the rest…

- Validations
- Events
- Also Github says…
  - Asynchronous support for queries
  - Ability to reflect Schema Metadata
  - Ability to generate a (DB) default row
  - Factory method for ad hoc/on the fly queries

PetaPoco

PetaPoco

- Yet another single file (and on Nuget)
- Inspired by Massive and Subsonic
- Supports POCOs (and optional attributes)
- Paging and transaction support OOTB
- Works with SQL Server, SQL Server CE, MySQL, PostgreSQL and Oracle.
- .NET 3.5 upwards

Demo

Creating the Database

```
PetaPoco.Database db = new PetaPoco.Database("ConnectionString")
```

Queries

```
Author a = db.Single<Author>(1);
```

```
Author b = db.Single<Author>("SELECT * FROM Authors WHERE Id = @0", 1);
```

```
Author c = db.Single<Author>(
    PetaPoco.Sql.Builder
        .Append("SELECT * FROM Authors")
        .Append("WHERE Id = @0", 1));
```

```
Author d = db.Single<Author>(
    PetaPoco.Sql.Builder
        .Select("*")
        .From("Authors")
        .Where("Id = @0", 1));
```

```
IEnumerable<Author> authors = db.Query<Author>(
    "SELECT * FROM Authors WHERE Username LIKE @0",
    "%example.com");
```

Inserts, Updates & Deletes

```
var x = db.Insert(new Author
{
    Username = "james@dapper.net",
    FullName = "James Hughes",
    CreatedDate = DateTime.Now
});
```

```
var x = db.Update("Authors", "Id", new
{
    Id = 10,
    FullName = "James Hughes"
});
```

```
db.Delete<Author>(8);
```

And the rest…

- Out of the box support for 1..N, N..N fetching of objects
  - Slightly awkward approach
  - RelationExtensions project abstracts this awkwardness for 1..N, N..1 queries
- Provides a set of T4 Templates for generating POCOs from DB schema

Simple.Data

Simple.Data

- Slightly different beast though it's principles are firmly in the MicroORM camp.
- Data Access Component
  - Not just for relational databases
  - NoSQL support
  - Extensible Adapter/Provider support
- Supports a wide range of data stores
  - SQL Server, SQL Server CE, Oracle, MySQL, SQLite, MongoDB
- .NET 4 only
- Extensive use of dynamics and meta-programming concepts.
  - TryInvokeMember, Method name reflection

Demo

Creating the Database

```
Simple.Data.Database.Open(); Simple.Data.Properties.Settings.DefaultConnectionString
```

```
Simple.Data.Database.OpenNamedConnection("DefaultConnectionString");
```

```
Simple.Data.Database.OpenConnection(Program.ConnectionString);
```

Queries

```
Author author = db.Author.FindById(1);
```

```
IEnumerable<Author> authors = db.Authors.FindByAllByName("Mr Author");
```

```
IEnumerable<Author> authors = db.Authors
    .FindAll(authors.Username.Like("%.com"))
    .Cast<Author>();
```

Inserts, Updates & Deletes

```
Author x   db.Authors.Insert(new Author
{
    Username = "james@dapper.net",
    FullName = "James Hughes",
    CreatedDate = DateTime.Now
});
```

```
db.Authors.Update(new Author
{
    Id = 5,
    Username = "james@dapper.net",
    FullName = "James Hughes",
    CreatedDate = DateTime.Now
});
```

```
db.Authors.UpdateById(
    Id: 6,
    FullName: "James Hughes");
```

```
db.Authors.DeleteById(3);
```

```
db.Authors.Delete(Id: 4);
```

And the rest…

- Inclusion of Simple.Data here may seem like a bit of contradiction. After all it is a straight up abstraction.
- It makes the 95th centile of all CRUD tasks “simple”
- Ability to compose DAL from other lightweight frameworks for the other edge case tasks
  - Additive approach rather than upfront monolithic approach


QUESTIONS?