Introduction to Reactor

An ORM for ColdFusion

alagad

120 Nelson Lane , Clayton, NC 27527 | p 1 (888) ALAGAD4 | f 1 (888) 248.7836 | www.alagad.com | info@alagad.com

Doug Hughes, President

What to Expect...

- A quick review of ORM techniques
- What Reactor is and isn't
- Supported database servers
- The Reactor API (briefly)
- Some examples (in code)

alagad

Traditional Data Access

```
<cfquery name="getUsers" datasource="myDSN">
   SELECT *
   FROM Users
</cfquery>

<cfoutput query="getUsers">
   <p>#firstName# #lastName#</p>
</cfoutput>
```

What problems can this code cause?

alagad

Traditional Data Access

```
<cfquery name="getUsers" datasource="myDSN">
   SELECT *
   FROM Users
</cfquery>

<cfoutput query="getUsers">
   <p>#firstName# #lastName#</p>
</cfoutput>
```

A hard coded DSN makes it difficult to change data source names in the future.

alagad

Traditional Data Access

```
<cfquery name="getUsers" datasource="myDSN">
   SELECT *
   FROM Users
</cfquery>

<cfoutput query="getUsers">
   <p>#firstName# #lastName#</p>
</cfoutput>
```

The code is not cohesive. Requires knowledge of data source, SQL, and display logic.

Traditional Data Access

```
<cfquery name="getUsers" datasource="myDSN">
   SELECT *
   FROM Users
</cfquery>

<cfoutput query="getUsers">
   <p>#firstName# #lastName#</p>
</cfoutput>
```

Data access is not encapsulated. Maintenance and reuse become more challenging.

Restate the Problem

- There are many places data comes from
- Access techniques vary depending on data source (using cfquery, cffile, cfldap, etc)
- Traditional techniques...
  - are difficult to maintain
  - are poorly encapsulated

Design Patterns to the Rescue

- Data Access Objects
- Gateways
- Active Records

Some Drawbacks to the New Way

- It's time consuming
  - Writing a one-off query only takes a minute
  - Writing a series of CFCs (and testing them) takes a lot longer.
- It's verbose
  - A typical query is only a few lines of code.
  - Adding CFCs into the mix adds a lot of extra CFML
- It's repetitive
  - Most CFCs end up looking almost identical.
  - Leads to a copy-paste-and-edit mentality.
  - Leads to subtle bugs, especially in rarely used code.

Database Abstraction Generators to the Rescue!

- Many developers end up writing programs to generate this repetitive code.
  - May automatically inspect the database (or not)
  - Tend to create static files which are manually updated as needed.
- This technique has it's own problems:
  - What if you customize an object but later add a new field to your database?

Enter Reactor...

- Reactor is code generation API.
  - Reactor generates objects as needed
  - The Reactor API is used in your code.
  - Objects are generated only as needed (and configured).
- Reactor instantiates objects and returns them.

A Simple Example

- This shows creation of the ReactorFactory and a records for the Address table in a database.

```
<cfset Reactor = CreateObject("Component",
    "reactor.reactorFactory").init(expandPath("reactor.xml")) />

<cfset Address = reactor.createRecord("Address") />
```

What Reactor Does

- Reactor is used to generate ColdFusion objects to access data in your database
- Reactor automates much of the repetitive, tedious and error-prone work involved in creating an Object Oriented database abstraction layer.

What Reactor Isn't...

- Reactor is Not Ruby (or ColdFusion) on Rails
  - Does not generate application controllers
  - No Scaffolding
  - Relies on XML, not conventions
- Reactor is Not a Panacea
  - It does not do everything you want it to!
  - You will need to customize some reactor generated objects.

Supported DBMS

- Microsoft SQL Server 2000 and 2005
- MySQL 4
- MySQL 5 and later
- PostgreSQL
- Oracle 9i and 10g
- DB2
- Others, too!

Getting Reactor

- Download from Alagad.com or ReactorFramework.org
- Get from Subversion (Preferred)
- You have a copy on CD in your binder.

Reactor Install Options

- Create a /reactor ColdFusion mapping pointing to the /reactor folder.
- Place /reactor folder a custom tag path
- Place /reactor folder in your application root (Preferred!)
  - Can be committed to source control
  - Specific version does not impact other applications

Reactor Configuration Bean

- reactor.config.Config
  - A bean used to configure reactor
  - Can easily be setup in ColdSpring
  - Passed into ReactorFactory's init method.

reactor.config.Config configuration options

pathToConfigXml
- This is path to the Reactor.xml file

project
- A unique name for this application
- Must follow ColdFusion variable naming rules

dsn
- The ColdFusion data source name to use

type
- The database type being used.
- Options are: db2, informix, mssql, mysql, mysql4, oracle, oraclerdb, postgresql, sqlanywhere

mapping
- This is a relative path to a directory where reactor will generate files

mode
- This controls how Reactor will regenerate files
- Options are: always, development, production.
- Use production for production. It's much faster.

username *(optional)*
- The username for the DSN

password *(optional)*
- The password for the DSN

The Reactor Factory

- Used to create a variety of objects
  - **Records**
  - **Gateways**
  - **Iterators**
  - Data Access Objects
  - Transfer Objects
  - Metadata Objects
  - Validator Objects

Creating the ReactorFactory

- You must pass a Config object to the ReactorFactory's init method.

```
<!--- setup the config bean --->
<cfset Config = CreateObject("Component",
   "reactor.config.config").init("Reactor.xml") />
<cfset Config.setProject("mg4dayExample1") />
<cfset Config.setDsn("Fortune") />
<cfset Config.setType("mssql") />
<cfset Config.setMapping("data") />
<cfset Config.setMode("always") />

<!--- create the reactor factory --->
<cfset Reactor = CreateObject("Component",
   "reactor.ReactorFactory").init(Config) />
```


alagad

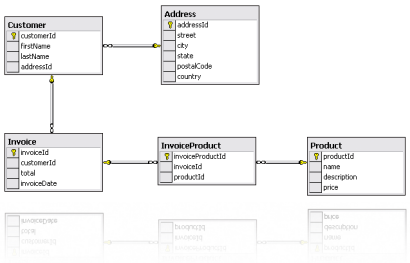
ReactorFactory Configuration in ColdSpring

- Configure the Reactor Config bean in ColdSpring
- Configure the ReactorFactory in ColdSpring
- Ask ColdSpring for the ReactorFactory

```
<cfset ColdSpring = CreateObject("component",
   "coldspring.beans.DefaultXmlBeanFactory").init() />
<cfset ColdSpring.loadBeans("ColdSpring.xml") />

<!--- create the reactor factory --->
<cfset Reactor =
   ColdSpring.getBean("ReactorFactory") />
```


alagad



The Example Database

alagad

The ReactorFactory API

(abridged)

createRecord(string)

- Creates and returns a Reactor Record

createGateway(string)

- Creates and returns a Reactor Gateway

createIterator(string)

- Creates and returns a Reactor Iterator

alagad

Reactor Records

- An implementation of the Active Record design pattern
- Generated based on database metadata and Reactor configuration settings
- Getters and setters for each field in the database
- Getters and setters for related objects and collections

Record CRUD API

load()
- Loads a record using the value set into the primary key property or a set of provided arguments and values corresponding to fields in the table.

save(useTransaction)
- Saves a record. UseTransaction defaults to true.

delete(useTransaction)
- Deletes a record. UseTransaction defaults to true.

getXyz() and setXyz(value)
- Gets and Sets values or Records from the database.
- Generated based on database metadata.

getXyzIterator()
- Returns an Iterator of data from the database
- Generated based on relationship configuration

Record Utilities API

populate(Bean)
- Populates a record based on a bean of data.
- Matches setters on the Record to getters on the Bean

isDirty()
- Indicates if the record has been modified since it was last loaded or saved

exists()
- Indicates if the Record exists in the database.

_getReactorFactory()
- Returns the ReactorFactory

Record Validation

- Reactor has a basic validation structure built in.
- Validates Records using Reactor Validators that are generated using database metadata and configuration
- You can customize Validators

Record Validation API

validate()
- Validates the Record using a Reactor Validator component

validated()
- Indicates if the record has been validated at all

hasErrors()
- Indicates if the record has errors after validation.

_getErrorCollection()
- Returns a collection of errors on the Record

Reactor Gateways

- An Implementation of a traditional ColdFusion Gateway Pattern.
- Provides some methods for getting queries of data

Gateway API

getAll()
- Gets a query of all records in the database for the gateway's table

getByFields(name1=value1, etc...)
- Accepts a collection of name value pairs indicating records to return in a query.

deleteAll()
- Deletes all records in the database for the gateway's table.

deleteByFields(name1=value1, etc...)
- Accepts a collection of name value pairs indicating records to delete in the gateway's table.

Gateway Utilities API

_getReactorFactory()
- Returns the ReactorFactory

createQuery()
- Returns a query object for Object Oriented Queries

getByQuery(OOQuery)
- Returns a query based on the provided OO Query

deleteByQuery(OOQuery)
- Deletes records based on the provided OO Query

The Basic Reactor.xml Structure

<reactor> (only one)
<objects> (only one)
<object> (one or more)
<hasOne> (zero or more)
<relate> (one or more)
<hasMany> (zero or more)
<relate> (one or more, exclusive to link tag)
<link> (only one, exclusive to relate tag)

Relationships In Reactor

- Configured in Reactor.xml
- HasOne
  - Indicates a record has only one of another record.
- HasMany
  - Indicates a record has one or more of another record.

HasOne Relationships

- Indicates that a table has a column referencing another table's primary key.
- Results in a Record having a getter and setter for the related Record.

Saves Cascade

- When you save a Record with relationships relates Records are also saved
  - Only if changed
  - Does not result in infinite loops

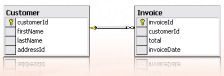
Configuring HasOne

```
<object name="Customer">
  <hasOne name="Address">
    <relate from="addressId" to="addressId" />
  </hasOne>
</object>
```


alagad

Standard HasMany Relationship

- Indicates that a table's primary key is referred to by a column in another table.
- Results in a Record having a method to get an Iterator of the related elements.



alagad

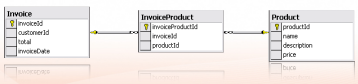
Configuring Standard HasMany

```
<object name="Address">
  <hasMany name="Customer">
    <relate from="addressId" to="addressId" />
  </hasMany>
</object>
```


alagad

Linked HasMany Relationship

- Indicates that two tables are related by a linking table with columns referencing the primary key in both tables.
- Results in a Record having a method to get an Iterator of the linked elements.



alagad

Configuring Linking HasMany

```
<object name="Invoice">
<hasMany name="Product">
   <link name="InvoiceProduct" />
</hasMany>
</object>

<object name="InvoiceProduct">
   <hasOne name="Product">
      <relate from="productId" to="productId" />
   </hasOne>
   <hasOne name="Invoice">
      <relate from="invoiceID" to="invoiceID" />
   </hasOne>
</object>

<object name="Product" />
```


Reactor Iterators

- An Iterator is a component used to collect Records
- Most often retrieved from a Record
  - getXyzIterator()
- Iterators work based on relationships configured in Reactor.xml

Iterator API

(abridged)

delete()
- Deletes a record by index, reference, or name value pair.

deleteAll()
- Deletes all elements in the Iterator

get()
- Gets an array of matching Records by index or name value pair

getAt()
- Gets a Record by index

getQuery()
- Gets a query of data in the Iterator

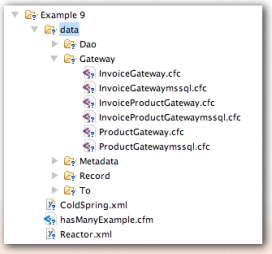
getArray()
- Gets an Array of Records in the Iterator

Reactor Common API

- All Reactor-generated objects have these functions (underscore does not imply private):
- _getReactorFactory()
- _getBean(string)

Reactor Object Customization

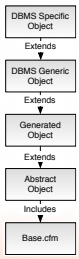
- All Reactor generated objects can be customized.
- In the configured mapping directory objects are grouped by type.
- DBMS specific and DBMS agnostic versions.



alagad

Reactor Object Hierarchy

- DBMS Specific Object
  - Where you put DBMS specific customizations
- DBMS Agnostic Object
  - Where you put DBMS agnostic customizations
- All other objects
  - Don't change!



alagad

Reactor and Model-Glue

- Model-Glue provides generic hooks into Reactor
- Model-Glue can use Reactor metadata to create UI scaffolds

alagad

Reactor ColdSpring Configuration for Model-Glue

```
    <bean id="ormService" class="reactor.reactorFactory">
    <constructor-arg name="configuration"><ref bean="reactorConfiguration" /></constructor-arg>
</bean>

<alias alias="ormAdapter" name="ormAdapter.Reactor" />

<bean id="reactorConfiguration" class="reactor.config.config">
    <constructor-arg name="pathToConfigXml"><value>/Fortune - Reactor/config/Reactor.xml</value></constructor-arg>
    <property name="project"><value>ReactorFortune</value></property>
    <property name="dsn"><value>Fortune</value></property>
    <property name="type"><value>mssql</value></property>
    <property name="mapping"><value>/Fortune - Reactor/model/data</value></property>
    <property name="mode"><value>development</value></property>
</bean>
```


alagad

What You Learned

- What an ORM framework is
- How Reactor works
- Some of the Reactor APIs
- How Reactor relates to Model-Glue and ColdSpring

Questions and Answers

Discussion

- What do you think about generating your database abstraction?
- Do you think an ORM framework like Reactor could be useful?
- Did you find your unit tests useful when updating your application to use Reactor?