CodeIgniter Models

lesson #basic03

James L. Parry
B.C. Institute of Technology

Agenda

1. CodeIgniter Models
2. Component Models
3. Object Relational Mapping
4. Database Utilities
5. Model Conventions
6. Database Configuration

CODEIGNITER MODELS

CodeIgniter leans more to the data access layer flavor of model, rather than the domain flavor of model.

Models in CodeIgniter are found in the `application/models` folder.

The model class name should have the first letter capitalized, and the filename should match that. You can get away with breaking this rule on Windows, but your webapp will break on Linux (the model's source file will not be found).

CI_Model

Each model should extend CI_Model, or your base model if you provide one.

Models need to be loaded before use. This can be done explicitly, using `$this->load->model('customers');` or it can be implied by configuration in `application/config/autoload.php`, with `$autoload['model'] = array('customers',...);`.

Once loaded, the model becomes a property of your controller, as in `$this->customers->...`.

Implementing Models

Some developers prefer to work closely with database drivers, writing their own SQL statements, as in

```
$results = $this->db->query(...);
```

Other developers prefer to use the Query Builder for a more O-O approach to database manipulation, as in

```
$model->where(...);
$model->limit(...);
$results = $model->get();
```

Using a Base Model

Many developers use a base model (`application/core/MY_Model`) to provide a consistent implementation of the base model methods they want for their webapp. This typically includes all of the CRUD methods, regardless of their style (driver or query builder).

Your webapp models would extend the base model, and implement usecase-specific methods.

An example using a base model:

```
class Sales extends MY_Model {
  function __construct() {
    parent::__construct();
  }

  function new_order(...) {}
  function add_item(...) {}
  function calc_tax(...) {}
  function receipt(...) {}

}
```

Misusing a Base Model?

The CodeIgniter core classes can be extended by having a `MY_classname.php` inside `application/core`. You might have noticed, with `application/core/MY_Controller.php`, that the class name inside that source file does not *have* to match the MY_... naming convention. We saw that with our original example, with the `Welcome` controller extending `Application`.

CodeIgniter instantiates a singleton when a model or other core class is loaded, which makes it awkward to deal with interfaces or abstract classes. However, we can exploit the general treatment of included source files by having more than one thing inside one.

Here is a Fancy Base Model!

Here is a "fancy" model. The source file includes an `Active_record` interface, as well as a `MY_Model` class. You could use this as a sort of template for writing adapters for other kinds of models (folders, XML, etc).

As long as one of your models, extending MY_Model, is loaded first, the `Active_record` interface will be accessible to you.

You could add additional models to the file, for instance `XML_Model` or `Folder_Model`, each implementing `Active_record`. Note that this is not necessarily a "best" practice", just a "possible" one, exploiting the current CodeIgniter. There will likely be better and more proper ways to do this with the next version!

Example Working With DB Driver

```
$query = $this->db->query('SELECT name, title, email FROM customers');

foreach ($query->result() as $row) {
  echo $row->title;
  echo $row->name;
  echo $row->email;
}

echo 'Total Results: ' . $query->num_rows();
```

Example Working With Query Builder

```
$query = $this->db->get();

foreach ($query->results() as $row) {
  echo $row->title;
  echo $row->name;
  echo $row->email;
}

echo 'Total Results: ' . $query->count_all();
```

COMPONENT MODELS

Component models encapsulate entities with classes that follow well-known conventions.

The best known component model is the JavaBean, originally from Sun Microsystems.

PHP does not have a similar convention, but many enterprise developers will assume that you know it, and possibly that you will try to follow it where practical.

JavaBeans

JavaBean rules, in a nutshell:

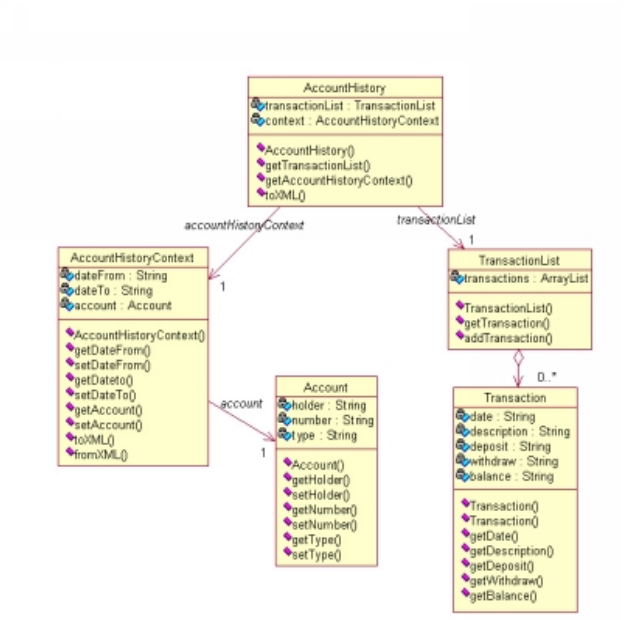
- No argument constructor
- Private fields
- Public accessor methods, a.k.a. getters
- Public mutator methods, a.k.a. setters

Common JavaBean practices:

- Convenience constructors
- Equality testing, `equals(object)`
- Text representation, `toString()`

Component Model Diagrams

Here is a typical UML class diagram showing some related JavaBeans.

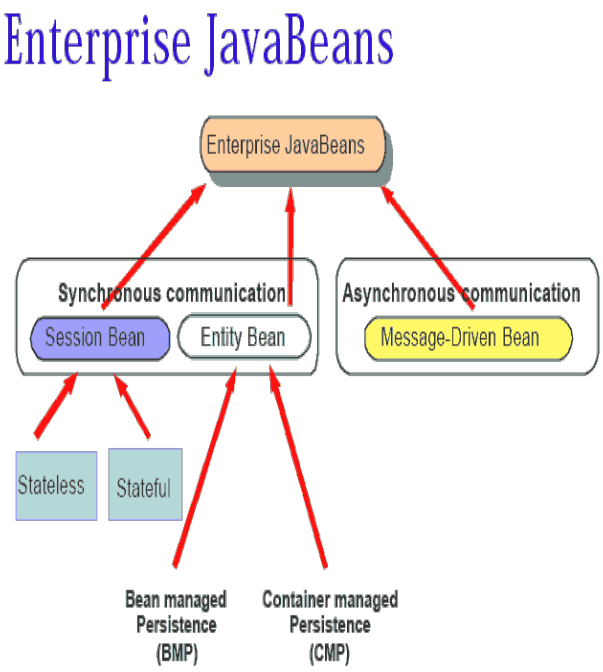


Enterprise Component Model

Building on JavaBeans, Enterprise JavaBeans provide a much more robust set of conventions, including the ability to handle persistence.

Without getting too carried away, the EJB inheritance hierarchy is shown to the right.

Why do you care? Because many enterprise systems assume a similar infrastructure in webapps they work with or interact with.

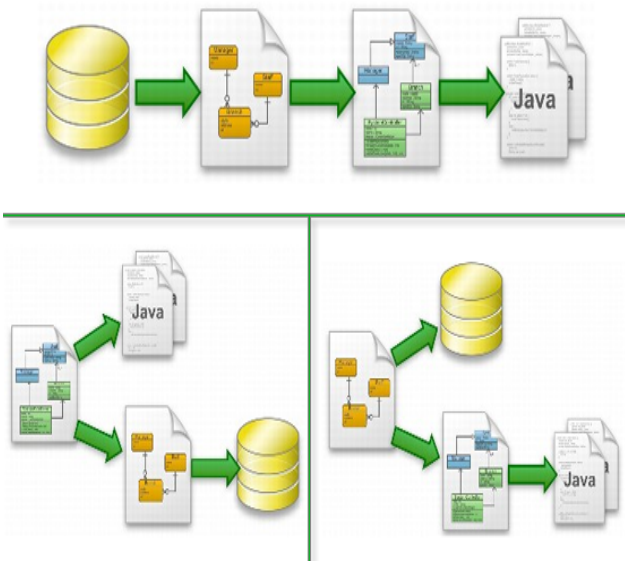


OBJECT-RELATIONAL MAPPING

Object-relational mapping is a programming technique or convention for converting data between an object model/state and a relational database model/state.

It is usually implemented by a tool or package which:

- From a database schema, generates an ERD, class diagram, and then code
- From a class diagram, generates code, an ERD and a schema
- From an ERD, generates a schema, and a class diagram and code



ORMs, Really...

In the PHP environment, object-relational mapping is typically effected by a third-party tool or plugin.

These tools attempt to automatically create the robustness of JavaBeans with the persistence of Enterprise JavaBeans.

These are notoriously awkward to implement properly.

Several such packages are available for CodeIgniter: Doctrine, Eloquent, and Propel. You're on your own using them, though - not part of this course!

Objects From Results

When PHP returns a result set from a relational database query, it usually does so as an iterable collection of row objects.

Each row object has properties corresponding to the table column names that the data came from.

This is *not* ORM, but it may suffice for many purposes.

If a row object is cast as an array, the result is an associative array with the table column names as indices.

Many PHP frameworks consider the objects and/or associative arrays returned by the database layer to be a simple form of ORM.

DATABASE UTILITIES

CodeIgniter has two classes specifically to manipulate databases.

The Database Forge class helps you work with metadata.

- Create or drop entire databases
- Create, drop or modify tables

The Database Utilities class helps you work with metadata.

- List databases
- Backup or export databases
- Repair or optimize databases
- Extract or convert records (CSV, XML)

SUGGESTED MODEL CONVENTIONS

Use entity-related naming!

Use the plural of an entity name for a table & model name. An example would be the `Posts` model for the `posts` table.

Use the singular of the entity name for a CRUD controller, for instance a RESTful one...

- `Post`
- `Post/view/x`
- `Post/add`

Avoid controllers and models with the same class name. Only one will be loaded. Namespaces (coming in CI4?) address this issue.

DATABASE CONFIGURATION

Your database access is configured by settings in `application/config/database.php`, or in environment-specific subfolders there for "development", "testing" or "production".

Read the Config class writeup carefully, as you want to use subfolders inside `application/config` to hold environment specific database configuration, for instance the username, password and database name to use when your webapp is deployed. It is a good practice to "git ignore" any such subfolders, so that you don't share confidential information on a public repository.

```
$db['default'] = array(
  'dsn' => '',
  'hostname' => 'localhost',
  'username' => 'root',
  'password' => '',
  'database' => 'database_name',
  'dbdriver' => 'mysqli',
  'dbprefix' => '',
  ...
  );
```

Database Conventions

For our purposes, I will expect your database configuration to work seemlessly when I pull your labs or assignments for marking. What this means:

- Make sure your default database configuration for "development" uses the username "root" with no password (unless I request otherwise)
- If I specify a database name, please use that, as I will have a test database already setup; such names are case-sensitive
- Use config subfolders, not commited to your repo, for any local or secure settings you need.

"Submission" Conventions

If you use a database in a project, clearly indicate so in your repo readme, and provide a SQL dump (gzipped) of your database, including dropping any tables you use before creating them, and including the insert statements to populate your database, as the file `setup.sql.gz` in either the root of your project or directly inside a `data` folder in your project. If you provide both, I will use the one in your project root.

If your webapp breaks because it uses the wrong database, you may be fired.
If you don't provide a database dump, and I cannot run your webapp at all, you're definitely fired.

Congratulations!

You have completed lesson #basic03: CodeIgniter Models

If you would take a minute to provide some feedback, we would appreciate it!

The next activity in sequence is: basic03-more More About Models (optional)

You can use your browser's back button to return to the page you were on before starting this activity, or you can jump directly to the course homepage, organizer, or reference page.