Abstracting databases access in Titanium Mobile

Xavier Lacot - September 2011
Hello

My name is Xavier Lacot

- I live in Paris
- I work at Clever Age, as director of the Expertise Center (http://www.clever-age.com/)
- Open Source convinced and contributor
- Titanium enthusiast and developer since 2009
- Web Frameworks expert
- Vice Pr. of the French PHP Users Association (afup.org)
- http://twitter.com/xavierlacot
Summary

1. Using databases in Titanium Mobile applications
2. Who said "pain"?
3. The ORM concept
4. Various js ORMs available
   - Titanium Mobile compatibility chart
5. A focus on joli.js
   - Main use
   - Joli API extension
Titanium provides a complete Database API:

- Titanium.Database
- Titanium.Database.DB
- Titanium.Database.ResultSet

Access to SQLite databases

The way to go when manipulating data in mobile applications!
Databases are very common in mobile applications

- Traveling guides (non-connected mode);
- News apps,
- Todo lists,
- Etc.
// create a connection
var db = Titanium.Database.open('database_name');

// execute a SQL query
var rows = db.execute('SELECT short_url FROM urls WHERE long_url = ?', Longurl);

// get a result
if (rows.isValidRow() && rows.fieldByName('short_url')) {
    result = rows.fieldByName('short_url');
}

// close the resultset
rows.close();

// close the database connection
db.close();
Some details to care to:

- Never forget to close() resultsets, or:
  - you will get memory leaks;
  - The app will unexpectedly close
- You will have to accept the mix of “view code” and “database code”... javascript and SQL in the same code pages...
Summary

1. Using databases in Titanium Mobile applications

2. Who said "pain"?

3. The ORM concept

4. Various js ORMs available
   - Titanium Mobile compatibility chart

5. A focus on joli.js
   - Main use
   - Joli API extension
Who said "pain"?

- Titanium.Database is ok for a few requests
- It is limited in large scale applications:
  - Imagine a 10 tables model, each with 20 fields
- Some questions:
  - Why write SQL queries yourself?
  - How to ensure data consistency / related entries retrieval?
  - How to deal with database migrations in your app?
  - How to avoid writing again and again the same queries?
A pain-in-the-ass sample

- Remove an item from an ordered list
  - Remove the item from the database
  - Update the other items positions

```javascript
// add delete event listener
tableview.addEventListener('delete', function(e) {
  var db = Titanium.Database.open('database_name');

  // delete the item
  db.execute(
    'DELETE FROM short_url WHERE id = ?',
    e.row.children[0].text
  );
```
... 

// update the other items positions  
var rows = db.execute('SELECT * FROM short_url ORDER BY position ASC');  
var position = 1;  

while (rows.isValidRow()) {
    db.execute(
        'UPDATE short_url SET position = ? WHERE id = ?',
        position,
        rows.fieldByName('id')
    );
    position++;
    rows.next();
}

// be a good boy  
rows.close();
db.close();
});
Wait oh wait

- Our business-code is cluttered with database manipulation code
- Why not simply write:

```javascript
// add delete event listener
tableView.addEventListener('delete', function(e){
  // assume short_url is an object which represents the short_url table
  short_url.get(e.row.children[0].text).remove();
});
```
Just to convince you...

- **A todo-list application**
  - Only display, count, get stats about the tasks of the currently selected category
  - Will you always write the «`WHERE category_id = '12'`» condition?
  - A better idea:

```javascript
category.get(12).listArticles();
category.get(12).countArticles();
// etc
```
Summary

1. Using databases in Titanium Mobile applications
2. Who said "pain"?
3. The ORM concept
4. Various js ORMs available
   • Titanium Mobile compatibility chart
5. A focus on joli.js
   • Main use
   • Joli API extension
What is an « ORM »?

- Object-Relational Mapper
  - Data access and manipulation abstraction
  - Classes represent tables, objects represent their content

<table>
<thead>
<tr>
<th>table Human</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>id</td>
<td>integer</td>
</tr>
<tr>
<td>lastname</td>
<td>text</td>
</tr>
<tr>
<td>firstname</td>
<td>text</td>
</tr>
<tr>
<td>city_id</td>
<td>integer</td>
</tr>
<tr>
<td>born_at</td>
<td>timestamp</td>
</tr>
<tr>
<td>is_alive</td>
<td>boolean</td>
</tr>
<tr>
<td>dead_at</td>
<td>timestamp</td>
</tr>
</tbody>
</table>

// say Human is a mapping class
var john = new Human();

john.set('lastname', 'Doe');
john.set('firstname', 'John');

// persist it
john.save();
Goals of an ORM

- **Manipulate records**
  - Never create or delete a record manually
  - Use behaviors (timestampable, taggable, etc.)
  - Clean user entries

- **Execute queries**
  - Abstract queries as objects
  - Pass it to several methods

- **Create your data model and manage it with migrations**
Summary

1. Using databases in Titanium Mobile applications
2. Who said "pain"?
3. The ORM concept
4. Various js ORMs available
   - Titanium Mobile compatibility chart
5. A focus on joli.js
   - Main use
   - Joli API extension
There are lots of javascript ORMs

- Suited for various Database access APIs
  - Browsers
  - Node
  - Titanium
  - Etc.

- Not every is convenient for Titanium
  - Leaks, incompatibility, not tested, etc.
  - Not using Titanium.database
Some of them, designed for Titanium

- ActiveJS Titanium fork - https://github.com/sr3d/activejs-1584174
- AppceleratorRecord - https://github.com/wibblz/AppceleratorRecord
- JazzRecord - http://www.jazzrecord.org/
- TiStore - https://github.com/jcfischer/TiStore
- yORM - https://github.com/segun/yORM
- Joli.js - https://github.com/xavierlacot/joli.js
- Maybe others?

... That's a nice list!
# ORMs chart

<table>
<thead>
<tr>
<th></th>
<th>Iphone</th>
<th>Android</th>
<th>Doc</th>
<th>License</th>
<th>Comments</th>
<th>Watchers (Forks)</th>
</tr>
</thead>
<tbody>
<tr>
<td>ActiveJS</td>
<td>Yes</td>
<td>No</td>
<td>Light</td>
<td>unknown</td>
<td>Migrations not working Not maintained?</td>
<td>7 (2)</td>
</tr>
<tr>
<td>Titanium fork</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Appcelerator Record</td>
<td>No</td>
<td>No</td>
<td>Light</td>
<td>unknown</td>
<td>Few functionalities Code not clean Not maintained?</td>
<td>48 (5)</td>
</tr>
<tr>
<td>JazzRecord</td>
<td>No</td>
<td>No</td>
<td>Extensive</td>
<td>MIT</td>
<td>Not only for Titanium (Air, etc.) Broken on Titanium since 2010/07 Not maintained?</td>
<td>75 (11)</td>
</tr>
<tr>
<td>TiStore</td>
<td>Yes</td>
<td>No</td>
<td>None</td>
<td>Apache</td>
<td>Not packaged Few functionalities</td>
<td>41 (5)</td>
</tr>
<tr>
<td>yORM</td>
<td>Yes</td>
<td>Yes</td>
<td>Light</td>
<td>unknown</td>
<td>Few functionalities Recent project</td>
<td>1 (1)</td>
</tr>
<tr>
<td>joli.js</td>
<td>Yes</td>
<td>Yes</td>
<td>Medium</td>
<td>MIT</td>
<td></td>
<td>99 (14)</td>
</tr>
</tbody>
</table>
Summary

1. Using databases in Titanium Mobile applications
2. Who said "pain"?
3. The ORM concept
4. Various js ORMs available
   - Titanium Mobile compatibility chart
5. A focus on joli.js
   - Main use
   - Joli API extension
Why joli.js

- I could not find what I was looking for in the other ORMs
- I wanted an abstract query API
- I wanted something short, simple and efficient

Some facts

- Much inspired by JazzRecord (js) and Doctrine (PHP)
- First release was written in 3 nights
- Models container
- Models declaration
- Abstract query language
- Record lifecycle management
- Performance analysis
- Extensible

- All this in a single ~850 lines file!

Features

- Models container
- Models
- Migrations
- Records
- Query engine

joli.js
- Easy access to the model classes
  - `get()`
  - `has()`
  - Etc.

- Able to launch the migrations
- **Models represent the tables**
  - Model declaration
  - Tables creation
  - Mass-records management
  - Fast selection methods (aka « Magic getters »)
Declaring a model

- Include joli.js
- Declare a connection to your database:

```javascript
var city = new joli.model(
    {
        table: 'city',
        columns: {
            id: 'INTEGER',
            name: 'TEXT',
            description: 'TEXT'
        }
    }
);
```

- Describe the model

```javascript
joli.connection = new joli.Connection('your_database_name');
```

- call

```javascript
joli.models.initialize();
```
Declaring a model

- Several models? Put them in a bag!

```javascript
var models = (function() {
    var m = {};

    m.human = new joli.model({
        table: 'human',
        columns: {
            id: 'INTEGER PRIMARY KEY AUTOINCREMENT',
            city_id: 'INTEGER',
            first_name: 'TEXT',
            last_name: 'TEXT'
        }
    });

    m.city = new joli.model({
        table: 'city',
        columns: {
            id: 'INTEGER PRIMARY KEY AUTOINCREMENT',
            name: 'TEXT'
        }
    });

    return m;
})();
```
var human = new joli.model({
  table: 'human',
  columns: {
    ...
  },
  methods: {
    countIn: function(cityName){
      // do something
    }
  },
  objectMethods: {
    moveTo: function(newCityName){
      // do something
    }
  }
});

// use a table-method
var habitantsCount = human.countIn('San Francisco');

// use an object-method
john.moveTo('Paris');
```javascript
var table = models.human;

// remove all humans
table.truncate();

// remove some records
table.deleteRecords([1, 7, 12]);

// test existance, based on "id"
// count entities
var allCount = table.count();
var DoesCount = table.count(
    where: {
        'last_name = ?': 'Doe',
        'age >= ?': 21
    }
);

// get all the ones matching criterions
var Does = table.all(
    where: {
        'last_name = ?': 'Doe',
        'age >= ?': 21
    },
    limit: 12
);
```
• **Goal:** Have an easy way to select the records of one table matching a given criteria.
  - `findOneById()`
  - `findOneBy()`
  - `findBy()`

```javascript
var table = models.human;

// returns all the inhabitants of the city n°12
var parisians = table.findBy('city_id', 12);

// returns one "human" record only (not sorted)
var michel = table.findOneBy('first_name', 'Michel');

// returns the human of id "118"
var human = table.findOneById(118);
```
Migrations

- Update the database layout when updating the application
- Allows to run other operations (callbacks available)
- Records are objects related to a row in the database
  - Record creation
  - Record access
  - Record update

- Records can be used even while not persisted
// first method
var john = models.human.newRecord({
    first_name: 'John',
    last_name: 'Doe'
});

// second method
var john = new joli.record(models.human);
john.fromArray({
    first_name: 'John',
    last_name: 'Doe'
});

// third method
var john = new joli.record(models.human);
john.set('first_name', 'John');
john.set('last_name', 'Doe');
Manipulate records

// persist a record
john.save();

// destroy it
john.destroy();

// get a property
var name = john.get('last_name');

// export to an array
var johnArray = john.toArray();
var json = JSON.stringify(johnArray);
// {"id":"110","lastname":"Doe","firstname":"John","company_name":"ACME"}
- Abstract the way queries are run against the database
- Stop writing SQL
- Use chained method calls « à la jQuery »
- Have hydration facilities
Querying the model

- No more SQL queries
- Let's introduce an OOP querying model
  - Queries are objects
  - They can be executed

```javascript
// create the query object
var q = new joli.query()
  .select()
  .from('human')
  .where('last_name = ?', 'Doe');

// let's execute it
var humans = q.execute();
```
A complete SQL-like vocabulary

- Several methods for building queries:
  - count()
  - destroy()
  - from()
  - groupBy()
  - insertInto()
  - join()
  - limit()
  - order()
  - set()
  - update()
  - values()
  - where()
  - wherein()
Progressive query construction

- Queries as objects are easy to handle
- No matter the order in which you call the query methods!
Let's talk about hydration

- Calling `execute()` will:
  - Build the query string;
  - Send it to `joli.Connection()` for its execution;
  - And create a bunch of record objects (one per result).
- This last step is called « hydration »
- It can cost time. A lot.

- Joli.js offers a way to hydrate plain arrays, not complete joli.js records.
Let's talk about hydratation

```javascript
var people = new joli.query()
  .select()
  .from('people')
  .execute();
// people is an array of objects

var people = new joli.query()
  .select()
  .from('people')
  .execute('array');
// people is a simple plain array
```

- An ORM as a cost, sure, but you can make it invisible to the user
- Save you app, take care to the performances
Querying useful methods

- `getSqlQuery()` returns the string that will be generated when executing the query

```javascript
var q = new joli.query()
  .select()
  .from('view_count')
  .where('nb_views between ? And ?', [1000, 2000]);

var queryString = q.getSqlQuery();
// select * from view_count where nb_views between "1000" and "2000"
```

- All the queries go through `joli.Connection.execute()`. Possibility to log things here and see what is happening.
- **Joli.js** is unit-tested using **titanium-jasmine**
  - 90+ tests and growing
  - See [https://github.com/xavierlacot/joli.js-demo](https://github.com/xavierlacot/joli.js-demo) for the test suite
Summary

1. Using databases in Titanium Mobile applications
2. Who said "pain"?
3. The ORM concept
4. Various js ORMs available
   - Titanium Mobile compatibility chart
5. A focus on joli.js
   - Main use
   - Joli API extension
Joli API extension

- We often need to synchronize data from/to the Web
- Case sample: an online address book
  - We want the contacts to be available on the phone even when not connected
  - The contacts list must also be available online
- Here comes joli.api.js, the little brother to joli.js
- joli.api.js is a wrapper to joli.js, which makes synchronization to REST web services easy

- All CRUD operations are available: GET / POST / PUT / DELETE
Let's have a small demo

- A Titanium-powered synchronized AddressBook
- Code will be available at https://github.com/xavierlacot/joli.api.js-app-demo
- Uses REST APIs built in PHP with the Symfony framework
var people = new joli.apimodel({
  table: 'people',
  columns: {
    id: 'INTEGER PRIMARY KEY AUTOINCREMENT',
    firstname: 'TEXT',
    lastname: 'TEXT',
    company_name: 'TEXT',
    email: 'TEXT',
    phone: 'TEXT',
    picture_url: 'TEXT',
  },
  updateTime: 86400,
  url: 'http://local.example.com/api/people.json'
});
Minor changes compared to joli.js

```javascript
// selects from the database
// if no result and the updateTime is gone, checks the API
var peoples = joli.models.get('people').all(
    {order: ['lastname asc', 'firstname asc']});

// creates the record and saves it to the REST endpoint
joli.models.get('people').newRecord(values, true).save();
```

Should the record be synchronized?
This is Free and Open Source Software...

- **All the code is here:**
  - joli.js - https://github.com/xavierlacot/joli.js
  - joli.api.js - https://github.com/xavierlacot/joli.api.js
  - joli.js test suite - https://github.com/xavierlacot/joli.js-demo
  - joli.api.js demo application - https://github.com/xavierlacot/joli.api.js-app-demo
Let's have a small demo

- This app was built completely while I was in the plane. Less than 4 hours coding!

```javascript
// persist the values of the form
button.addEventListener('click', function() {
    // extractValues() builds an associative array of the form values
    save(extractValues(container));
    win.close();
});

var save = function(values) {
    joli.models.get('people').newRecord(values, true).save();
};
```

INFO] POST request to url http://local.example.com/api/people.json
INFO] Received from the service:
INFO]   {"id":111,"lastname":"Lacot","firstname":"Xavier", ...}
INFO] 1 new record(s), 0 record(s) updated.
DEBUG] fire app event: joli.records.saved
Roadmap and wishlist...

- **Joli.js:**
  - Abstract the configuration
    - Logging enabled or not, default hydration model
    - Easy support for several databases
  - Improve migrations, add more unit tests

- **Joli.api.js**
  - Support for all the HTTP methods
  - Make it possible to map the Data model to different REST services formats

Keep all this fun, short and efficient