\



THE PROJECT DIPES

MÔ HÌNH MVC

CẤU TRÚC VÀ DIỄN GIẢI

# MỞ ĐẦU

# CHƯƠNG 1: MÔ HÌNH MVC

## MVC

Những concept liên quan đến MVC đều đã quá quen thuộc và chúng cũng quen thuộc với DIPES.

Mô hình MVC của DIPES cũng chia thành bốn thành phần tương tự những mô hình MVC khác, bao gồm: Models, Views, Controller và các config liên quan đến môi trường, cơ sở dữ liệu

Với DIPES, Views của nó không phải các template và static files thông thường, DIPES sử dựng Reactjs để renders giao diện và giao tiếp với Views thông qua API từ routing với các Controllers.

## CẤU TRÚC THƯ MỤC

Cây thư mục của DIPES MVC có cấu trúc như sau:

A picture containing diagram, text, parallel, technical drawing

Description automatically generated

### config

Thư mục config chứa hầu hết các cấu hình của mô hình MVC bao gồm các đối tượng và thiết đặt cơ sở dữ liệu.

1. controllers
2. database
3. connector.js

Thiết đặt kết nối cơ sở dữ liệu và export một đối tượng database connector tuỳ thuộc vào loại cơ sở dữ liệu và các thuộc tính được cấp trong .env

1. database.js
2. models
3. Thư mục fields

Chứa các tệp định nghĩa đối tượng kiểu dữ liệu cho model, mỗi đối tượng có một nhóm thuộc tính và phương thức đặt trưng.

Tất cả chúng đều kế thừa một lớp gọi là fields

Chương sau sẽ giải thích rỏ hơn vấn đề này

1. index.js

Tệp tổng hợp module và export lại chúng dưới dạng một đối tượng với tên của thư mục models

1. database.js

Tệp này định nghĩa những đối tượng cơ sở dữ liệu dựa vào cấu hình được cấp bởi /config/database/connector.js và dùng nó để thực thi các lệnh truy vấn.

1. table.js

Tệp này định nghĩa đối tượng table, đối tượng này dùng đối tượng từ database.js ở trên để thực thi truy vấn ở phạm vi bảng.

1. model.js

Tệp model định nghĩa một model chuẩn nhất và sẽ được kế thừa bởi các model do người dùng định nghĩa.

### controllers

Định nghĩa những controllers, thứ sẽ được dùng để giao tiếp với frontend dưới dạng các api.

### Models

Định nghĩa các models, những models này sẽ được controllers sử dụng để thực hiện tính toán và trả về kết quả.

Ngoài ra, dựa vào việc các models được định nghĩa thế nào mà /manage.js sẽ thực thi thay đổi cấu trúc trên cơ sở dữ liệu y như vậy.

### routes

Các routes là những tệp định nghĩa đường dẫn cho các api, route định nghĩa các phương thức và dùng controllers để phản hồi kết quả.

### server.js

Tệp chạy chính của chương trình, cấu hình toàn bộ middleware, routing và security.

### .env

Định nghĩa các biến môi trường và sử dụng chúng cho các cấu hình liên quan đến cơ sở dữ liệu và networking

### manage.js

Tệp quản lý cơ sở dữ liệu và tự động tạo các models, controller với cấu hình mặc định

## CHƯƠNG 2: CÁC ĐỐI TƯỢNG

## TRƯỚC KHI BẮT ĐẦU

Trước khi bắt đầu, chúng ta cần thống nhất với nhau một số luật về đặt tên và phạm vi sử dụng của các phương thức cũng như thuộc tính của các đối tượng.

Các phương thức và thuộc tính với tên bắt đầu bằng # được xem là các phương thức private. Nodejs không có khái niệm về protected nên các phương thức hoặc thuộc tính không có # ở đầu tên đồng nghĩa với khả năng truy cập là public.

* + - 1. Phương Thức

Tên của các phương thức với tên bắt đầu và kết thúc bằng cặp dấu *underscore [ \_\_ ]*  là những phương thức nguy hiểm có thể gây ảnh hưởng lên dữ liệu, hoặc thậm chí là cả bảng. Những phương thức này bị hạn chế truy cập và thường sẽ không được truy cập trực tiếp ở các lớp kế thừa cuối cùng.

Các phương thức với tên bình thường có thể được phép truy cập tự do

Các phương thức có cờ async là những phương thức bất đồng bộ và phải dùng await hoặc bất kỳ cách thức nào khác có thể phân giải promise để có thể sử dụng được.

* + - 1. Thuộc tính

Tên của các thuộc tính đều bắt đầu bằng cặp dấu *underscore [ \_\_ ]* và chúng đều không được phép truy cập bên ngoài lớp chứa chúng bất kể việc chúng là public hay private. Giá trị của các thuộc tính đều phải thông qua một phương thức get/set tương ứng để thực thi, nếu không có các phương thức này thì kể như thuộc tính đó là bất khả xâm phạm.

* + - 1. Biến

Tên các biến phải được đặt theo nguyên tắt camelCaseSyntax, tuy nhiên một số biến được băm ra khỏi một đối tượng thì vẫn có thể không áp dụng nguyên tắt này.

## NHỮNG ĐỐI TƯỢNG CỐT LÕI

### FIELD ( Trường )

* 1. **Vị trí**

Đối tượng Field có thể được tìm thấy tại đường dẫn ~/config/models/fields/field.js

* 1. **Mô tả**

Field là đối tượng được dùng để định nghĩa lớp trừu tượng phục vụ cho việc kế thừa của các đối tượng định nghĩa kiểu dữ liệu

* 1. **Kế thừa**

Field không kế thừa bất kỳ lớp nào.

* 1. **Các đối số khởi tạo**

Field có 3 đối số khởi tạo

|  |  |  |
| --- | --- | --- |
| **Tên đối số** | **Kiểu dữ liệu** | **Mô tả** |
| name | <String> | Tên trường |
| datatype | <String> | Kiểu dữ liệu của trường |
| value | <Any> | Giá trị của trường |

* 1. **Các thuộc tính**

|  |  |  |  |
| --- | --- | --- | --- |
| **Tên thuộc tính** | **Kiểu** | **Phạm vi** | **Mô tả** |
| \_\_fieldName | <String> | public | Tên trường |
| \_\_datatype | <String> | public | Kiểu dữ liệu của trường |
| \_\_value | <Any> | public | Giá trị hiện tại của trường |

Với value không được truyền vào, \_\_value mặc nhiên sẽ mang giá trị *undefined*

* 1. **Các phương thức**
     1. **\_\_alterDataType\_\_(type)**

**Phạm vi:** public

**Đối số:**

* type <String>

**Mô tả:** Thay đổi kiểu dữ liệu của trường

* + 1. **value( val =** *undefined* **)**

**Phạm vi:** public

**Đối số:**

* val <Any>

**Mô tả:** Nếu đối số val được truyền vào và mang một giá trị cụ thể ( kể cả chuỗi rỗng hoặc false, null ), phương thức này sẽ đặt lại giá trị của thuộc tính \_\_value thành val và trả về val. Nếu không thì trả về giá trị hiện tại của \_\_value.

* + 1. **serializingValue()**

**Phạm vi:** public

**Mô tả:** Trả về một đối tượng với một thuộc tính duy nhất là giá trị của \_\_fieldName và mang giá trị là \_\_value;

### TABLE ( Bảng )

* 1. **Vị trí**

Đối tượng Table có thể được tìm thấy tại đường dẫn ~/config/models/table.js

* 1. **Mô tả**

Đối tượng bảng định nghĩa toàn bộ những thuộc tính và phương thức cần có của một bảng trong cơ sở dữ liệu bao gồm tên gọi, các trường, khoá và cách thức truy vấn dữ liệu

* 1. **Kế thừa**

Bảng không kế thừa từ bất kỳ lớp nào

* 1. **Các đối số khởi tạo**

|  |  |  |
| --- | --- | --- |
| **Tên đối số** | **Kiểu** | **Mô tả** |
| name | <String> | Tên bảng |

* 1. **Các thuộc tính**

|  |  |  |  |
| --- | --- | --- | --- |
| **Tên thuộc tính** | **Kiểu** | **Phạm vi** | **Mô tả** |
| \_\_tableName | <String> | private | Tên bảng |
| \_\_fields | <Obj>[] | private | Danh sách các trường, mỗi trường là một obj với 2 thuộc tính là \_\_field\_name và \_\_fieldObject lần lượt mang kiểu <String> và Field |
| \_\_foreignKeys | <Obj>[] | private | Danh sách các trường khoá ngoại, mỗi trường là một object với 2 thuộc tính là \_\_fieldName và \_\_tableName và đều mang kiểu <String>. |
| \_\_primaryKey | <String>[] | private | Danh sách tên các trường được chỉ định làm khoá chính cho bảng |
| dbo | Database | private | Đối tượng truy vấn trực tiếp với cơ sở dữ liệu |

* 1. **Các phương thức**
     1. **async \_\_dbInit\_\_**

**Phạm vi:** public

**Mô tả:** Khởi tạo đối tượng dbo.

Đây là phương thức khởi tạo nhưng phải chạy bất đồng bộ nên nó sẽ được gọi lại mỗi khi có một phương thức thực thi truy vấn đến cơ sở dữ liệu

* + 1. **\_\_getTableName\_\_**

**Phạm vi:** Public

**Mô tả:** Trả về giá trị của thuộc tính \_\_tableName, cũng là tên của bảng hiện tại.

* + 1. **\_\_getFields\_\_**

**Phạm vi:** public

**Mô tả:** Trả về danh sách các trường hiện có của bảng.

* + 1. **\_\_getPrimaryKey\_\_**

**Phạm vi:** public

**Mô tả:** Trả về danh sách các trường thuộc khóa chính

* + 1. **\_\_getForeignKeys\_\_**

**Phạm vi:** public

**Mô tả:** Trả về danh sách các khóa ngoại hiện có.

* + 1. **\_\_setFields\_\_( fields )**

**Phạm vi:** public

**Đối số:**

* fields <Obj>[]

**Mô tả:** Đặt lại danh sách các trường

* + 1. **async \_\_getNewId\_\_**

**Phạm vi:** public

**Mô tả:** Trả về một ID mới và duy nhất thuộc bảng này, ID này dùng cho việc lưu trữ dữ liệu và truy vấn bất kể việc khóa chính có được thiết lập hay không.

* + 1. **\_\_isFieldExisted\_\_( fieldName )**

**Phạm vi:** public

**Đối số:**

* fieldName <String>

**Mô tả:** Kiểm tra xem một field có tồn tại trong danh sách \_\_fields hay không

* + 1. **\_\_addField\_\_( fieldName, fieldObject, fieldProps =** *undefined* **)**

**Phạm vi:** public

**Đối số:**

* fieldName <String>
* fieldObject <Field>
* fieldProps <Obj>

**Mô tả:** Thêm một trường mới vào danh sách đồng thời tạo một thuộc tính mới với tên trường và giá trị là một đối tượng kế thừa Field tuỳ thuộc vào kiểu dữ liệu và danh sách các thuộc tính được truyền vào.

* + 1. **\_\_addForeignKey\_\_** **( fieldName, referencesOn )**

**Phạm vi:** public

**Đối số:**

* fieldName <String>
* referencesOn <Model>

**Mô tả:** Thêm một trường khoá ngoại và ghi đè một thuộc tính với tên là bảng chứa khoá chính

* + 1. **\_\_addPrimaryKey\_\_( fields )**

**Phạm vi:** public

**Đối số:**

* fields <String>[]

**Mô tả:** Thêm (các) trường vào danh sách khoá chính

* + 1. **\_\_serializePrimaryData\_\_( serializedData )**

**Phạm vi:** public

**Đối số:**

* serializedData <Obj>

**Mô tả:** Dựa vào danh sách khóa chính để chọn ra những trường phù hợp, lưu chúng vào một object và trả về

* + 1. **\_\_primaryKeyCheck\_\_( data )**

**Phạm vi:** public

**Đối số:**

* data <Obj>

**Mô tả:** Bằng cách sử dụng kết quả từ **\_\_serializePrimaryData\_\_** với đối số là data như điều kiện truy vấn, thực thi lệnh select với đối tượng dbo để gọi dữ liệu, nếu kết quả trả về là null thì khóa chính đó hợp lệ vì chưa tồn tại, ngược lại thì kể như khóa chính không hợp lệ.

* + 1. **\_\_foreignKeyCheck\_\_( data )**

**Phạm vi:** public

**Đối số:**

* data <Obj>

**Mô tả:** Kiểm tra (các) ràng buộc khóa ngoại, nếu dữ liệu được ánh xạ không tồn tại thì kể như dữ liệu không hợp lệ.

* + 1. **async \_\_insertRecord\_\_(data)**

**Phạm vi:** public

**Đối số:**

* data <Any>[]

**Mô tả:** Chèn dữ liệu theo dạng một bảng ghi với thông tin được truyền vào tuần tự theo thứ tự của các trường, không nên dùng phương thức này vì có thể làm sai lệch dữ liệu giữa các trường

* + 1. **async \_\_insertObject\_\_(serializedData)**

**Phạm vi:** public

**Đối số:**

* serializedData <Obj>
  + 1. **Mô tả:** Lưu dữ liệu vào cơ sở dữ liệu sau khi kiểm tra các ràng buộc về khóa bằng các phương thức **\_\_primaryKeyCheck\_\_(** serializedData **)** và **\_\_foreignKeyCheck\_\_(** serializedData **).**
    2. **async \_\_find\_\_( amount =** *undefined* **)**

**Phạm vi:** public

**Đối số:**

* amount <Int>

**Mô tả:** Truy vấn một số lượng dữ liệu nếu amount có giá trị, nếu không thì truy vấn toàn bộ dữ liệu.

* + 1. **async \_\_findCriteria\_\_( criteria )**

**Phạm vi:** public

**Đối số:**

* criteria <Obj>

**Mô tả:** Truy vấn dữ liệu bằng điều kiện là một đối tượng

* + 1. **async \_\_updateObject\_\_( serializedData )**

**Phạm vi:** public

**Đối số:**

* serializedData <Obj>

**Mô tả:**

Cập nhật dữ liệu

- Cập nhật dữ liệu chỉ áp dụng với các trường không phải khóa chính

- Nếu trường được cập nhật là khóa ngoại mà vi phạm ràng buộc thì cũng sẽ không được cập nhật

* + 1. **async \_\_deleteObject\_\_( criteria =** *undefined* **)**

**Phạm vi:** public

**Đối số:**

* criteria <Obj>

**Mô tả:** Xóa một (hoặc nhiều) bảng ghi khỏi bảng

* + 1. **async \_\_deleteAll\_\_**

**Phạm vi:** public

**Mô tả:** Xóa toàn bộ dữ liệu hiện có của bảng

### MODEL

1. **Vị trí**

Đối tượng Model có thể được tìm thấy tại đường dẫn ~/config/models/model.js

1. **Mô tả**

Model là đối tượng chung nhất, thể tất cả những hành vi của một model thường có, những model trong MVC thực thi các lệnh truy vấn thông qua các phương thức có được từ kế thừa Model

1. **Kế thừa**

Model không kế thừa trực tiếp một đối tượng nào nhưng nó có một thuộc tính là model có kiểu dữ liệu là một đối tượng Table.

Đối tượng Table này sẽ mặc nhiên thêm một trường với tên **id** kiểu Int để làm chỉ số cho các bảng ghi mỗi khi một bảng ghi được thêm mới vào cơ sở dữ liệu. **Id** đóng vai trò như mã định danh cho bảng ghi bất kể việc bảng có được đặt khóa chính hay không, các phương thức truy vấn vẫn sẽ biết được dữ liệu nào sẽ cần được truy vấn đến.

1. **Các đối số khởi tạo**

|  |  |  |
| --- | --- | --- |
| **Tên đối số** | **Kiểu** | **Mô tả** |
| modelName | <String> | Tên của Model và cũng là tên của bảng sẽ được dùng cho việc lưu trữ dữ liệu. |

1. **Các thuộc tính**

|  |  |  |  |
| --- | --- | --- | --- |
| **Tên thuộc tính** | **Kiểu** | **Phạm vi** | **Mô tả** |
| types | <Obj> | static | Các kiểu dữ liệu sẽ được dùng cho trường |
| model | <Table> | private | Đối tượng bảng sẽ được dùng cho truy vấn dữ liệu |
| id | <Int> | public | Mã định danh cho bảng ghi, trường này thường sẽ mang giá trị là *undefined* trừ khi được một ModelRecord khởi tạo ( Chương 3 ) |

1. **Các phương thức**
   * 1. **getModel**

**Phạm vi:** public

**Mô tả:** Trả về thuộc tính model

* + 1. **setDefaultValue( serializedData )**

**Phạm vi:** public

**Đối số:**

* serializedData <Obj>

**Mô tả:** Thiết đặt giá trị cho các trường của model, phương thức này chỉ dùng cho đối tượng kế thừa cuối cùng và phạm vi ở mức độ một bảng ghi.

* + 1. **\_\_addField\_\_( fieldName, fieldObject, fieldProps =** *undefined* **)**

**Phạm vi:** public

**Đối số:**

* fieldName <String>
* fieldObject <Field>
* fieldProps <Obj>

**Mô tả:** Truyền các đối số trên vào phương thức **\_\_addField\_\_** của thuộc tính model

* + 1. **\_\_addForeignKey\_\_** **( fieldName, referencesOn )**

**Phạm vi:** public

**Đối số:**

* fieldName <String>
* referencesOn <Model>

**Mô tả:** Truyền các đối số trên vào phương thức **\_\_addForeignKey\_\_** của thuộc tính model

* + 1. **\_\_addPrimaryKey\_\_( fields )**

**Phạm vi:** public

**Đối số:**

* fields <String>[]

**Mô tả:** Truyền các đối số trên vào phương thức **\_\_addPrimaryKey\_\_** của thuộc tính model

* + 1. **find( query =** *undefined* **)**

**Phạm vi:** public

**Đối số:**

* query <Int> || <Obj>

**Mô tả:**

Nếu query là *undefined* hoặc một số nguyên thì sẽ gọi phương thức **this**.#model\_\_find\_\_

Nếu query là một đối tượng thì sẽ gọi phương thức **this**.#model.\_\_findCriteria\_\_

* + 1. **insert( data )**

**Phạm vi:** public

**Đối số:**

* data <Any>[]

**Mô tả:** Chèn thêm một bảng ghi vào bảng, dữ liệu trong Array phải đúng thứ tự với thứ tự tương ứng của các trường ở phần định nghĩa của model.

* + 1. **async save**

**Phạm vi:** public

**Đối số:** Phương thức này không có đối số, nhưng dữ liệu được phương thức này dùng được lấy từ phương thức **setDefaultValue** bên trên, nên khi sử dụng phương thức này, cần đảm bảo phương thức **setDefaultValue** đã được gọi và các trường mang ràng buộc khóa đều đã có dữ liệu.

**Mô tả:**

* Nếu trường id chưa có dữ liệu, phương thức này sẽ lưu một bảng ghi mới.
* Nếu trường id có dữ liệu và nó tồn tại trong bảng thì sẽ cập nhật lại các trường không phải khóa chính
  + 1. **async remove**

**Phạm vi:** public

**Đối số:** Phương thức này không có đối số, nó dùng ID có được từ **this**.id để xóa chính xác bảng ghi với ID của đối tượng.

**Mô tả:** Xóa bảng ghi với ID là **this**.id.value()

### Controller

1. **Vị trí**

Đối tượng Controller có thể được tìm thấy tại đường dẫn ~/config/controllers/controller.js

1. **Mô tả**

Controller cung cấp những phương thức chung nhất khi xây dụng các controller bao gồm xác thực token, xác thực các quyền và mã hóa

1. **Kế thừa**

Controller không kế thừa một đối tượng nào cả.

1. **Các đối số khởi tạo**

Controller không có đối số khởi tạo

1. **Các thuộc tính**

|  |  |  |  |
| --- | --- | --- | --- |
| **Tên thuộc tính** | **Kiểu** | **Phạm vi** | **Mô tả** |
| tokenKey | <String> | public | Khóa dùng để biên dịch token |

1. **Các phương thức**
   * 1. **writeReq(request)**

**Phạm vi:** public

**Đối số:**

* request: Đối số này là đối tượng request mỗi khi một url được gọi, bản thân request mang rất nhiều thông tin về giao thức http, tuy nhiên chúng ta chỉ dùng 2 thuộc tính của nó là url và method

**Mô tả:** in ra màng hình, và có thể sẽ ghi log, mỗi khi có request được gửi đến

* + 1. **writeRes(response)**

**Phạm vi:** public

**Đối số**:

* response: Đối số này chứa 2 thuộc tính
  + status: trạng thái trả về
  + message: mô tả nội dung trả về

**Mô tả:** in ra màng hình, và có thể sẽ ghi log, mỗi khi phản hồi một yêu cầu.

* + 1. **notNullCheck(data, fields)**

**Phạm vi:** public

**Đối số:**

* data <Obj>
* field <String>[]

**Mô tả:** Dựa vào danh sách cái trường (fields) để kiểm tra xem trong số chúng, thì có thuộc tính tương ứng nào của data bị *undefined* không. Nếu toàn bộ đều có dữ liệu thì kể như đạt. Phương thức này trả về một objects với 2 thuộc tính tùy thuộc vào việc dữ liệu có hợp lệ hay không

{

valid: true | false,

nullFields: <String>[]

}

* + 1. **makeToken(data)**

**Phạm vi:** public

**Đối số:**

* data <Obj>

**Mô tả:** Tạo một token mới có hiệu lực trong 1h với dữ liệu từ data

* + 1. **decodeToken(token)**

**Phạm vi:** public

**Đối số:**

* token <JWT>

**Mô tả:** Dịch token thành <Obj> dựa vào dữ liệu được lưu trong token

* + 1. **verifyToken(req)**

**Phạm vi:** public

**Đối số:**

* req <Request>

**Mô tả:** Gọi token từ req.headers, nếu token tồn tại và còn hiệu lực thì trả về true, không thì trả về false

### Database

1. **Vị trí**

Đối tượng Database có thể được tìm thấy tại đường dẫn ~/config/models/database.js

1. **Mô tả**

Database không phải một đối tượng cụ thể mà tùy vào các thiết đặt ở ~/.env mà Database sẽ có cách triển khai riêng, nhưng về cơ bản thì những đối tượng Database sẽ cung cấp một bộ phương thức với tên y hệt nhau.

1. **Kế thừa**

Database không kế thừa một đối tượng nào cả.

1. **Các đối số khởi tạo**

Database không có đối số khởi tạo

1. **Các thuộc tính**

|  |  |  |
| --- | --- | --- |
| **Tên đối số** | **Kiểu** | **Mô tả** |
| dbo | <ConnectorObj> | Connector được export từ ~/config/database/connector.js |

1. **Các phương thức**
   * 1. **async init**

**Phạm vi:** public

**Mô tả:** Phương thức khởi tạo thuộc tính dbo

* + 1. **async getAutoIncrementId( table )**

**Phạm vi:** public

**Đối số:**

* table <String>

**Mô tả:** Lấy một ID mới, duy nhất của bảng với tên *table*

* + 1. **async select( table, criteria =** *undefined* **)**

**Phạm vi:** public

**Đối số:**

* table <String>
* criteria <Obj>

**Mô tả:** chọn ra tất cả dữ liệu tương ứng với điều kiện được truyền vào bởi *criteria* từ bảng với tên *table*

* + 1. **async insert( table, value )**

**Phạm vi:** public

**Đối số:**

* table <String>
* value <Obj>

**Mô tả:** Lưu đối tượng vào bảng

* + 1. **async update ( table, criteria, newValue )**

**Phạm vi:** public

**Đối số:**

* table <String>
* criteria <Obj>
* newValue <Obj>

**Mô tả:** cập nhật một hoặc nhiều bảng ghi với giá trị mới

* + 1. **async delete( table, criteria )**

**Phạm vi:** public

**Đối số:**

* table <String>
* criteria <Obj>

**Mô tả:** Xóa những bảng ghi dữ liệu tương ứng với những giá trị có trong *criteria*

## NHỮNG ĐỐI TƯỢNG MỞ RỘNG

### NUMBER

1. **Vị trí**

Number có thể được tìm thấy ở ~/config/models/fields/field.js

1. **Mô tả**

Number thể hiện kiểu số nói chung trong cơ sở dữ liệu bao gồm số nguyên và số thực

1. **Kế thừa**

Number kế thừa Field

1. **Các đối số khởi tạo**

|  |  |  |
| --- | --- | --- |
| **Tên đối số** | **Kiểu** | **Mô tả** |
| name | <String> | Tên trường |
| value | <Any> | Giá trị của trường, mặc nhiên là *undefined* |
| props | <Obj> | Các thuộc tính khác của trường |

1. **Các thuộc tính**

Number kế thừa toàn bộ các thuộc tính của Field và có riêng cho nó một bộ thuộc tính mới bao gồm

|  |  |  |  |
| --- | --- | --- | --- |
| **Tên thuộc tính** | **Kiểu** | **Phạm vi** | **Mô tả** |
| \_\_required | <Bool> | public | Thể hiện tính bắt buộc của dữ liệu khi khởi tạo đối tượng |

1. **Các phương thức**

Number kế thừa toàn bộ phương thức từ Field

* + 1. **\_\_initializeProperties\_\_(props)**

**Phạm vi:** private

**Đối số:**

Props <Obj {

required <Bool>

}>

**Mô tả:** Thiết đặt các thuộc tính nội tại của kiểu Number

* + 1. **selfValidate()**

**Phạm vi:** public

**Mô tả:** Xác thực giá trị hiện tại của trường có đúng với kiểu dữ liệu hay không.

### INT

1. **Vị trí**

Int có thể được tìm thấy ở ~/config/models/fields/int.js

1. **Mô tả**

Int biểu diễn kiểu số nguyên

1. **Kết thừa**

Int kế thừa Number

1. **Các đối số khởi tạo**

|  |  |  |
| --- | --- | --- |
| **Tên đối số** | **Kiểu** | **Mô tả** |
| name | <String> | Tên trường |
| value | <Any> | Giá trị của trường, mặc nhiên là *undefined* |
| props | <Obj> | Các thuộc tính khác của trường |

1. **Các thuộc tính**

Int kế thừa toàn bộ các thuộc tính của cả Field và Number.

1. **Các phương thức**
   * 1. **selfValidate**

**Phạm vi:** public

**Ghi đè:** Number.selfValidate()

**Mô tả:** Xác thực giá trị hiện tại có phải kiểu nguyên hay không và đặt lại giá trị sau khi đã ép kiểu vào **this**.\_\_value

* + 1. **value( val =** *undefined* **)**

**Phạm vi:** public

**Ghi đè:** Field.value( val = undefined )

**Đối số:**

* val <Any>

**Mô tả:** Tương tự như Field.value() nhưng có thêm một bước xác thực kiểu dữ liệu ở trường hợp có đối số truyền vào.

### BOOL

* 1. **Vị trí**

Bool có thể được tìm thấy ở ~/config/models/fields/Bool.js

* 1. **Mô tả**

Kiểu Bool dùng để thể hiện giá trị đúng hoặc sai

* 1. **Kế thừa**

Bool kế thừa Field

* 1. **Các đối số khởi tạo**

|  |  |  |
| --- | --- | --- |
| **Tên đối số** | **Kiểu** | **Mô tả** |
| name | <String> | Tên trường |
| value | <Any> | Giá trị của trường, mặc nhiên là *undefined* |
| props | <Obj> | Các thuộc tính khác của trường |

* 1. **Các thuộc tính**

Bool kết thừa toàn bộ thuộc tính của Field, và:

|  |  |  |  |
| --- | --- | --- | --- |
| **Tên thuộc tính** | **Kiểu** | **Phạm vi** | **Mô tả** |
| \_\_default | <Bool> | public | Giá trị mặc định |

* 1. **Các phương thức**
     1. **\_\_initializeProperties\_\_( props )**

**Phạm vi:** private

**Đối số:**

* props <Obj {

default <Bool>

}>

**Mô tả:** Thiết đặt các thuộc tính nội tại của kiểu Bool

* + 1. **selfValidate**

**Phạm vi:** public

**Mô tả:** Xác thực kiểu dữ liệu của giá trị hiện tại

### STRING

1. **Vị trí**

String có thể được tìm thấy ở ~/config/models/fields/string.js

1. **Mô tả**

Kiểu String được dùng để thể hiện dữ liệu dạng chuỗi ký tự hoặc đoạn văn bản.

1. **Kế thừa**

String kế thừa Field

1. **Các đối số khởi tạo**

|  |  |  |
| --- | --- | --- |
| **Tên đối số** | **Kiểu** | **Mô tả** |
| name | <String> | Tên trường |
| value | <Any> | Giá trị của trường, mặc nhiên là *undefined* |
| props | <Obj> | Các thuộc tính khác của trường |

1. **Các thuộc tính**

Bool kế thừa toàn bộ thuộc tính từ Fields, và:

|  |  |  |  |
| --- | --- | --- | --- |
| **Tên thuộc tính** | **Kiểu** | **Phạm vi** | **Mô tả** |
| \_\_default | <String> | public | Giá trị mặc định |
| \_\_maxLength | <Int> | public | Độ dài tối đa |
| \_\_required | <Bool> | public | Thể hiện tính bắt buộc của dữ liệu khi khởi tạo đối tượng |

1. **Các phương thức**
   * 1. **\_\_initializeProperties( props )**

**Phạm vi:** private

**Đối số:**

* props <Obj {

default <String>

maxLength <Int>

required <Bool>

}>

**Mô tả:** Thiết đặt giá trị cho các thuộc tính của kiểu

* + 1. **setMaxLength( length )**

**Phạm vi:** public

**Đối số:**

* length <Int>

**Mô tả:** Thiết đặt giá trị cho độ dài tối đa

* + 1. **value ( val =** *undefined* **)**

Tương tự Field.value() nhưng có thêm xác thực độ dài trước khi đặt lại giá trị cho **this.\_\_**value

### DATETIME

1. **Vị trí**

Datetime có thể được tìm thấy ở ~/config/models/fields/datetime.js

1. **Mô tả**

Datetime thể hiện kiểu dữ liệu ngày, ngày giờ

1. **Kế thừa**

Datetime kế thừa Field

1. **Các đối số khởi tại**

|  |  |  |
| --- | --- | --- |
| **Tên đối số** | **Kiểu** | **Mô tả** |
| name | <String> | Tên trường |
| value | <Any> | Giá trị của trường, mặc nhiên là *undefined* |
| props | <Obj> | Các thuộc tính khác của trường |

1. **Các thuộc tính**

Datetime kế thừa toàn bộ thuộc tính từ Fields, và:

|  |  |  |  |
| --- | --- | --- | --- |
| **Tên thuộc tính** | **Kiểu** | **Phạm vi** | **Mô tả** |
| \_\_format | <String> | public | Định dạng khi xuất dữ liệu |
| \_\_required | <Bool> | public | Thể hiện tính bắt buộc của dữ liệu khi khởi tạo đối tượng |

1. **Các phương thức**
   * 1. **\_\_initializeProperties\_\_(props)**

**Phạm vi:** private

**Đối số:**

* props <Obj {

format <String>

required <Bool>

}>

**Mô tả:** Thiết đặt giá trị cho các thuộc tính của kiểu

* + 1. **\_\_format\_date\_\_**

**Phạm vi:** private

**Mô tả:** Dựa vào **this.\_\_**format để tạo ra một chuỗi kết quả từ dữ liệu của **this.**\_\_value và trả về

* + 1. **getFormatedValue**

**Phạm vi:** public

**Mô tả:** Trả về kết quả có được từ phương thức \_\_format\_date\_

# CHƯƠNG 3: CẤU TRÚC TIÊU CHUẨN

## Models

Các models đều được tạo bằng một CLI với tên *manage.js*, cách dùng và mô tả chi tiết đã được đề cập đến ở *THE DIPES PROJECT - MVC CLI USAGE*

### Tổng quan

Sau khi được tạo bởi CLI, một model với tên *MyModel* sẽ có sẳn cấu trúc như sau:



1. **MyModel**

MyModel là Model được tạo bởi CLI và kế thừa trực tiếp Model, tên của nó được được dùng làm đối số kế thừa, Model sẽ dùng đối số này để đặt tên cho bảng sẽ được dùng để lưu trữ dữ liệu, đồng thời làm tên của thuộc tính đại diện cho bản thân MyModel nếu nó được dùng như một khóa ngoại ở một model khác.

MyModel có phạm vi hoạt động ở mức độ một bảng bao gồm các phương thức truy vấn và thay đổi hàng loạt dữ liệu.

MyModel không cần đối số khởi tạo.

Phương thức khởi tạo ngoài dùng super để truyền đối số kế thừa, còn dùng một vài phương thức khác để thiết đặt cấu trúc cho mô hình dữ liệu

\_\_addField\_\_, \_\_addPrimaryKey và \_\_addForeignKey\_\_ được kế thừa từ Model ( đã được giải thích ở *CHƯƠNG 2: CÁC ĐỐI TƯỢNG* ).

Khi MyModel được tạo, nó có sẳn 1 trường là MyModel\_id và mang kiểu Int.

Những kiểu dữ liệu có thể truy cập thông qua Model.types bao gồm:

|  |  |  |
| --- | --- | --- |
| **STT** | **Kiểu dữ liệu** | **Cú pháp** |
| 1 | INT | Model.types.int |
| 2 | Number | Model.types.number |
| 3 | Bool | Model.types.bool |
| 4 | Datetime | Model.types.datetime |
| 5 | String | Model.types.string |

Ghi chú: Xem chi tiết tại [Những đối tượng mở rộng](#_NHỮNG_ĐỐI_TƯỢNG)

Và trường MyModel\_id cũng được thiết đặt là khóa chính thông qua phương thức \_\_addPrimaryKey\_\_, khóa chính có thể thay đổi tùy ý, tuy nhiên sau khi thay đổi cấu trúc bảng, phải thực hiện đúng các thao tác như hướng dẫn sử dụng của *manage.js*

1. **MyModelRecord**

MyModelRecord là lớp định nghĩa một bảng ghi, nó kế thừa MyModel.

MyModelRecord có tầm hoạt động ở mức độ bảng ghi và có dữ liệu trên các trường, khác với MyModel chỉ định nghĩa các trường mà không hề lưu dữ liệu.

MyModelRecord có đối số khởi tạo, đó là một đối tượng có các thuộc tính tương ứng với các trường được định nghĩa ở MyModel và một trường **id** đã được giải thích ở chương trước. Những đối số này được truyền trực tiếp vào phương thức **this**.setDefaultValue để lưu dữ liệu vào các trường tương ứng

Các trường có thể được truy cập trực tiếp như một thuộc tính, tuy nhiên mọi thao tác liên quan đến truy xuất dữ liệu đều phải thực hiện thông qua phương thức **this**.value

### Các phương thức

MyModel và MyModelRecord kế thừa hầu hết các thuộc tính và phương thức của Model, bao gồm:

|  |  |  |
| --- | --- | --- |
| **STT** | **Tên phương thức** | **Mô tả** |
| 1 | getModel | Trả về đối tượng truy cập bảng trong cơ sở dữ liệu |
| 2 | \_\_addField\_\_ | Thêm một trường |
| 3 | \_\_addForeignKey\_\_ | Thêm khóa ngoại |
| 4 | \_\_addPrimaryKey\_\_ | Thêm khóa chính |
| 5 | find | Gọi dữ liệu |
| 6 | Insert | Thêm bảng ghi |
| 7 | save | Lưu bảng ghi |

*(Các phương thức và đối số của chúng đều đã được diễn giải cụ thể ở Chương 2 nên sẽ không nhắc lại ở đây!)*

Các phương thức trên phục vụ cho việc tạo ra các phương thức nội tại trong các model, ngoài ra, việc thêm một thuộc tính nào đó là điều hết sức bình thường hễ nó không ảnh hưởng đến cấu trúc bảng. Mọi thao tác làm thay đổi cấu trúc bảng như thêm, bớt, đặt khóa chính/ngoại đều phải thực hiện theo hướng dẫn của *manage.js*

### Các dùng

Sau khi được định nghĩa và đảm bảo về tính hợp lệ, các model đã có thể được sử dụng. Thông thường các model sẽ được import vào các controller để thực thi các lệnh truy vấn dữ liệu, hoặc được dùng trong các hàm do người dùng định nghĩa

Cách dùng chi tiết sẽ được đề cập cụ thể ở các chương sau.

## Controllers

Các controller đều được tạo bởi CLI và có sẳn một cấu trúc như sau:



Một controller sẽ kế thừa lớp Controller.

Khi được tạo bởi CLI, controller sẽ có sẳn một phương thức mẫu với cấu trúc như phương thức **get** bên trên

Phương thức này nhận 2 tham số req và res, chúng là những tham số đặt thù khi dùng để định nghĩa một *route* trong express.

Phương thức này còn dùng 2 phương thức khác là *writeReq* & *writeRes* để in ra màng hình debug lịch sử truy cập, việc gọi 2 phương thức này là optional và có thể bỏ qua.

**get** kết thúc bằng cách trả về một response dưới dạng một đối tượng, cấu trúc trả về này là tùy biến, tuy nhiên vẫn khuyến nghị sử dụng cấu trúc như trên để tiện cho việc debug và consume các api.