

Loopback Developer workshop

- i4.0 IT Team
- 2020/9/14



Agenda

- Tools
- Project Architecture
- Launch Server
- Data Source
- Models
- Model Relation
- Custom API
- Access Controls List (ACL)



Tools



Postman

測試RESTful API的
工具

門檻低、上手快且跨平台、有免付費版本且功能強大、可保存歷史記錄，多終端同步。



heidisql

連接資料庫的GUI
工具

跨平台、可連結多種資料庫服務、管理用戶權限、有免付費版本、並能自動生成SQL、。



Project Architecture

- **common/**
內主要擺放 model、definition 和 ACL roles 的程式碼。
- **server/**
在這個資料夾擺放的是設定 Loopback 用的各種 JSON 設定檔，包括 datasource、middleware 等。
- **server/server.js**
Loopback 程式主體。同樣基於express.js，可視情況自行加入middleware
- **server/boot**
server/boot 資料夾則是擺放給 loopback-boot 使用的 bootstrapping scripts。



Project Architecture

- 建立專案【[使用Loopback CLI](#)】

```
> lb
> 您的應用程式名稱？ loopback-practice
> 輸入用來包含專案的目錄名稱：(loopback-practice)
> 您想要使用哪個LoopBack版本？(Use arrow keys)
  2.x (長期支援)
> 3.X (現行)
> 您想要何種應用程式？(Use arrow keys)
> api-server (採用...)
```

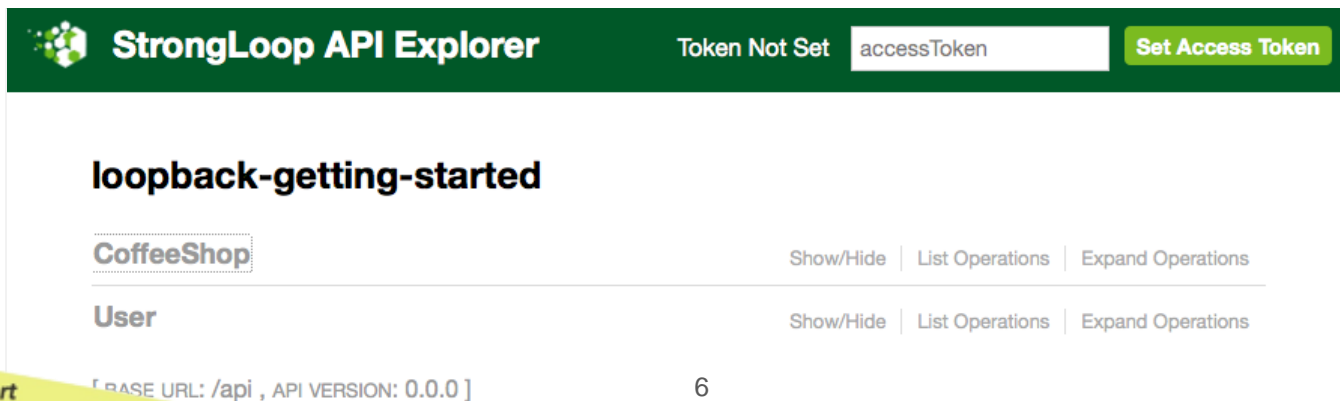


Launch Server

- 執行loopback

```
> node .
```

- 開啟API Explorer <http://localhost:3000/explorer> (預設)



The screenshot shows the StrongLoop API Explorer interface. The header is dark green with the StrongLoop logo, the title "StrongLoop API Explorer", and a "Token Not Set" status with an input field for "accessToken" and a "Set Access Token" button. The main content area displays a list of API endpoints. The first endpoint is "loopback-getting-started", which is expanded to show two sub-entries: "CoffeeShop" and "User". Each sub-entry has "Show/Hide", "List Operations", and "Expand Operations" links. At the bottom, the status bar shows "[BASE URL: /api , API VERSION: 0.0.0]".

Endpoint	Show/Hide	List Operations	Expand Operations
loopback-getting-started			
CoffeeShop			
User			



Data Source

- server/datasources.json 【以MariaDB為例】
- 使用Loopback CLI的作法

> lb datasource

> 輸入來源名稱：mariaDB

> 選取mariaDB的連接器：(Use arrow keys)

...

> MySQL (由StrongLoop支援)

> Connection String url to override other settings ?

> mysql://user:pass@host/db 填入資料庫的連線資訊

> 安裝loopback-connector-mysql@^5.3.0(Y/n) Y



Data Source

- server/datasources.json 【手動建立】

```
> npm i loopback-connector-mysql --save
```

```
1 {
2   "db": {
3     "name": "db",
4     "connector": "memory"
5   },
6   "mysqlDs": {
7     "host": "127.0.0.1",
8     "port": 3306,
9     "url": "",
10    "database": "practice_db",
11    "password": "L00pBack",
12    "name": "root",
13    "user": "root",
14    "connector": "mysql"
15  }
```




Models

- Model模型基礎

1. ApplicationModel

包含metadata、configure等對loopback server相關的描述

2. PersistedModel

連接一貫的資料來源，提供基本的CRUD

3. User

保護資料來源，須獲得系統允許才可存取資料

4. Access Control Models

包含ACL、Access Token、Scope、Role及RoleMapping，負責控制存取資源的模型



Models

- 使用Loopback CLI的作法建立Models

```
> lb model
> 輸入模型名稱：member
> 選取可連接member的資料來源 (Use arrow keys)
> mariaDB (mysql)
> 選取模型的基礎
> PersistedModel
> 透過REST API來公開member嗎？ (Y/n) Y
> 自訂複數形 (用來建置REST API)：
> 一般模型或僅限伺服器 (Use arrow keys)
> 一般
> 完成時，輸入空白內容名稱。
```



Models



- 手動建立Model【方法一】
- [common/models/member.json](#)

```
1 {
2   "name": "member",
3   "base": "PersistedModel",
4   "idInjection": false,
5   "options": {
6     "validateUpsert": true
7   },
8   "properties": {
9     "mid": {
10      "type": "number",
11      "id": true,
12      "required": true,
13      "default": 0
14    },
15    "username": {
```



Models

- 手動建立Model【方法二】
- 建立Models前，先到MariaDB建一個Table

鍵值	名稱	資料型態	長度	負數	NULL	預設
 PK	id	INT	10			AUTO_INCREMENT
 UK	username	VARCHAR	20			無
	password	VARCHAR	20			無
	email	TEXT				無

- 執行[server/bin/member.js](#)，直接產出Table的schema JSON



Model Relation

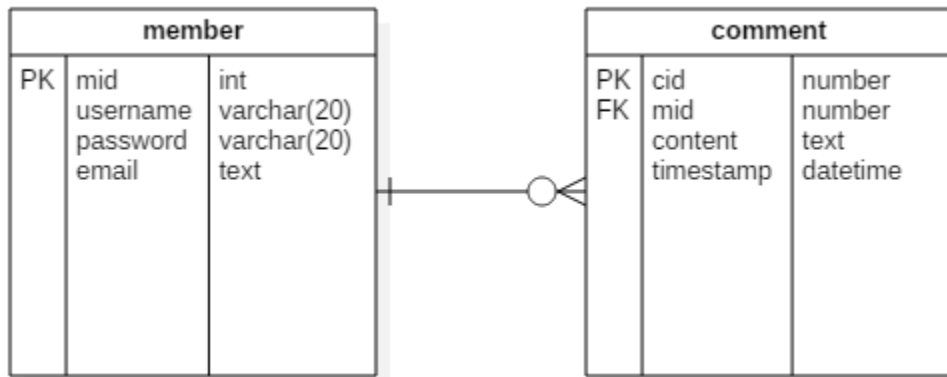
- HasMany
- BelongsTo
- HasOne
- HasManyThrough



Model Relation

- HasMany

建立在與其他模型「一對多」的連結，此關係表示模型的每個實例都具有零個或多個對應的模型實例。Ex. 每位會員可有多筆留言。



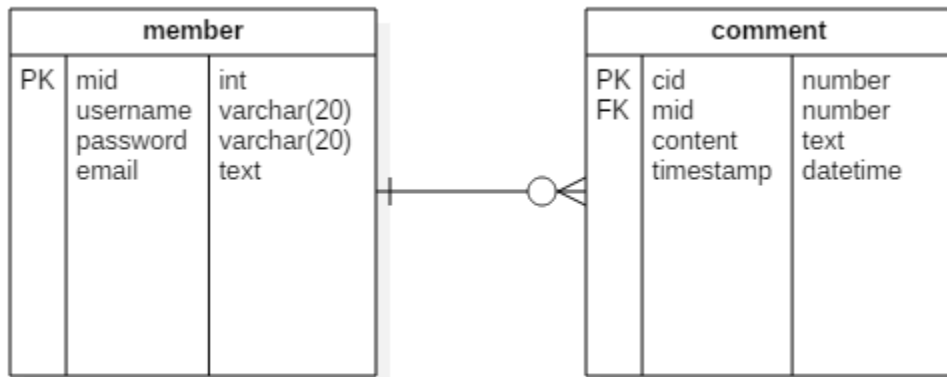


Model Relation

- BelongsTo

建立在與其他模型「一對多」的連結，在此種關係的其中一側可以發現「BelongsTo」的關係

Ex. 多筆留言都來自同一個會員。



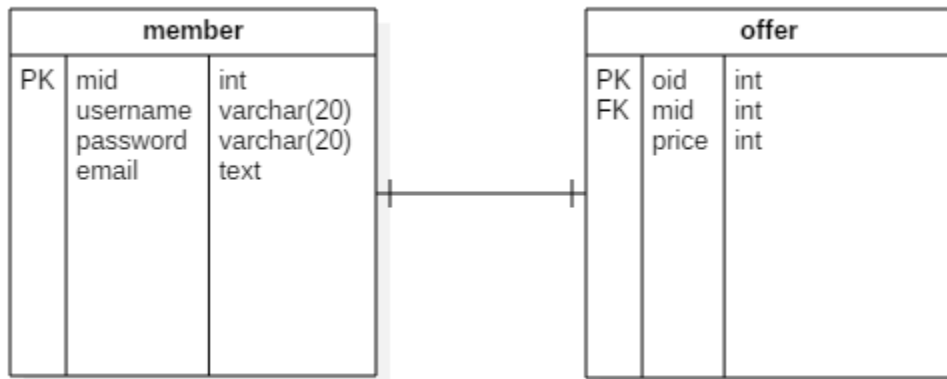


Model Relation

- HasOne

建立在與其他模型「一對一」的連結。

Ex. 每位會員只能有一筆優惠。



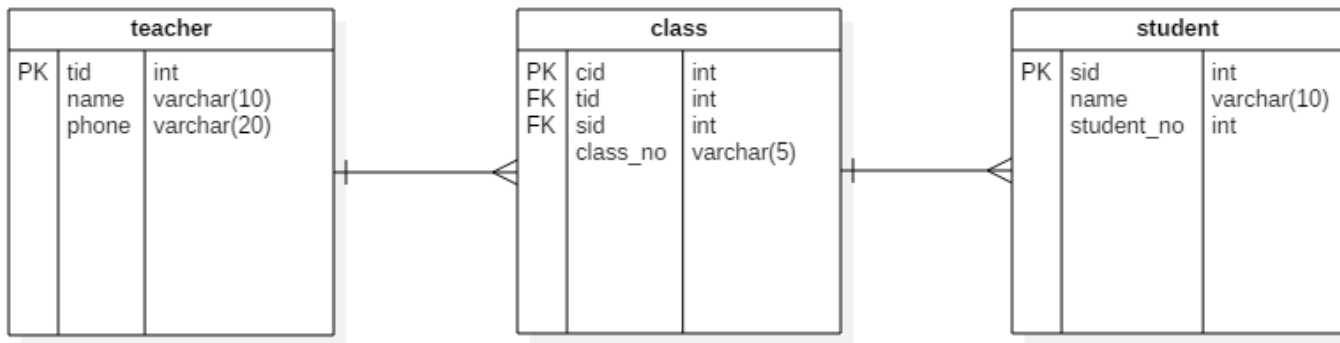


Model Relation

- HasManyThrough

同樣建立在其他模型「一對多」的連結情況，但須透過一個中繼模型。

Ex. 每科老師帶多個班級，每個學生屬於某一個班級。





Model Relation

- 使用Loopback CLI的作法建立Model Member Relation

Member has many Messages

- > lb relation
- > 選取要建立的關係模型：member
- > 關係類型：has many
- > 選擇要與其建立關係的模型：message
- > 輸入關係的內容名稱：messages
- > 選擇性地輸入自訂外部索引鍵：mid
- > 需要一個通過模型？No
- > 容許在REST API中形成巢狀關係？No
- > 停用包含關係？No



Model Relation

- 使用Loopback CLI的作法建立Model Message Relation

Message belong to Member

- > lb relation
- > 選取要建立的關係模型：message
- > 關係類型：belongs to
- > 選擇要與其建立關係的模型：member
- > 輸入關係的內容名稱：member
- > 選擇性地輸入自訂外部索引鍵：mid
- > 容許在REST API中形成巢狀關係？No
- > 停用包含關係？No



Models Relation

- 手動建立Model Member Relation 【HasMany】
- common/models/member.json

```
1 {  
2   ...  
3   "relations": {  
4     "messages": {  
5       "type": "hasMany",  
6       "model": "message",  
7       "foreignKey": "mid"  
8     }  
9   },  
10  ...  
11 }
```



Models Relation

- 手動建立Model Message Relation 【BelongsTo】
- common/models/message.json

```
1 {  
2   ...  
3   "relations": {  
4     "member": {  
5       "type": "belongsTo",  
6       "model": "member",  
7       "foreignKey": "mid"  
8     }  
9   },  
10  ...  
11 }
```



Models Relation

- API Explorer 結果 【Member】

LoopBack API Explorer

Token Set. 3BCllmDBcE05TpGG2NdFSet Access Token

GET /members/{id}/existsCheck whether a model instance exists in the data source.

GET /members/{id}/messages查詢 messages 個 (共 member 個)。

Response Class (Status 200)
Request was successful

Model Example Value

```
[
  {
    "id": 0,
    "mid": 0,
    "timestamp": "2019-01-03T08:33:37.489Z",
    "comment": "string"
  }
]
```

Response Content Type application/json

Parameters

Parameter	Value	Description	Parameter Type	Data Type
id	1	member id	path	string
filter			query	string

Try it out!

[Hide Response](#)

Curl



Models Relation

- API Explorer 結果 【Message】

LoopBack API ExplorerToken Set: I7IvtznkeJD2A84BJ4EQnI7Set Access Token

GET /comments/{id}/member

提取 belongsTo 關係 member ◦

Response Class (Status 200)
Request was successful

Model | Example Value

```
{
  "id": 0,
  "username": "string",
  "password": "string",
  "email": "string"
}
```

Response Content Type: application/json

Parameters

Parameter	Value	Description	Parameter Type	Data Type
id	5	comment id	path	string
refresh			query	boolean

Try it out!

Hide Response

Curl

```
curl -X GET --header 'Accept: application/json' 'http://localhost:3000/api/comments/5/member?access_token=I7IvtznkeJD2A84BJ4EQnI7'
```



Model Relation

- 使用Loopback CLI的作法建立Model Teacher Relation

Teacher has many through Class

- > lb relation
- > 選取要建立的關係模型：teacher
- > 關係類型：has many
- > 選擇要與其建立關係的模型：student
- > 輸入關係的內容名稱：students
- > 選擇性地輸入自訂外部索引鍵：tid
- > 需要一個通過模型？Yes
- > 選擇一個通過模型：class
- > 容許在REST API中形成巢狀關係？No
- > 停用包含關係？No



Model Relation

- 使用Loopback CLI的作法建立Model Student Relation

Student has many through Class

- > lb relation
- > 選取要建立的關係模型：student
- > 關係類型：has many
- > 選擇要與其建立關係的模型：teacher
- > 輸入關係的內容名稱：teachers
- > 選擇性地輸入自訂外部索引鍵：sid
- > 需要一個通過模型？Yes
- > 選擇一個通過模型：class
- > 容許在REST API中形成巢狀關係？No
- > 停用包含關係？No



Model Relation

- 使用Loopback CLI的作法建立Model Class Relation

Class belong to Teacher

- > lb relation
- > 選取要建立的關係模型：class
- > 關係類型：belongs to
- > 選擇要與其建立關係的模型：teacher
- > 輸入關係的內容名稱：teacher
- > 選擇性地輸入自訂外部索引鍵：tid
- > 容許在REST API中形成巢狀關係？No
- > 停用包含關係？No



Model Relation

- 使用Loopback CLI的作法建立Model Class Relation

Class belong to Student

- > lb relation
- > 選取要建立的關係模型：class
- > 關係類型：belongs to
- > 選擇要與其建立關係的模型：student
- > 輸入關係的內容名稱：student
- > 選擇性地輸入自訂外部索引鍵：sid
- > 容許在REST API中形成巢狀關係？No
- > 停用包含關係？No



Models Relation

- 手動建立Model Teacher Relation 【HasManyThrough】
- common/models/teacher.json

```
1 {
2   ...
3   "relations": {
4     "students": {
5       "type": "hasMany",
6       "model": "student",
7       "foreignKey": "tid",
8       "through": "class"
9     }
10  },
11  ...
12 }
```



Models Relation

- 手動建立Model Student Relation 【HasManyThrough】
- common/models/student.json

```
1 {  
2   ...  
3   "relations": {  
4     "teachers": {  
5       "type": "hasMany",  
6       "model": "teacher",  
7       "foreignKey": "sid",  
8       "through": "class"  
9     }  
10  },  
11  ...  
12 }
```



Models Relation

- 手動建立Model Class Relation 【BelongsTo】
- common/models/class.json

```
1 {
2   ...
3   "relations": {
4     "teacher": {
5       "type": "belongsToMany",
6       "model": "teacher",
7       "foreignKey": "tid"
8     },
9     "student": {
10      "type": "belongsToMany",
11      "model": "student",
12      "foreignKey": "sid"
13    }
14  },
```



Models Relation

- API Explorer 結果 【Teacher】

LoopBack API ExplorerToken Not Set [Set Access Token](#)

GET /teachers/{id}/students

查詢 students 個 (共 teacher 個)。

Response Class (Status 200)

Request was successful

Model

Example Value

```
[
  {
    "sid": 0,
    "name": "string",
    "student_no": 0
  }
]
```

Response Content Type

Parameters

Parameter	Value	Description	Parameter Type	Data Type
id	<input type="text" value="1"/>	teacher id	path	string
filter	<input type="text"/>		query	string

Try it out!

[Hide Response](#)

Curl



Models Relation

- API Explorer 結果 【Student】

LoopBack API ExplorerToken Not SetSet Access Token

GET /students/{id}/teachers

查詢 teachers 個 (共 student 個)。

Response Class (Status 200)
Request was successful

Model | Example Value

```
[
  {
    "tid": 0,
    "name": "string",
    "phone": "string"
  }
]
```

Response Content Type

Parameters

Parameter	Value	Description	Parameter Type	Data Type
id	<input type="text" value="1"/>	student id	path	string
filter	<input type="text"/>		query	string

Try it out! [Hide Response](#)

Curl

32



Custom API

- 使用Remote Method

1. create()
2. find()
3. findById()
4. replaceById()
5. destroyById()
6. destroyAll()
7. [More...](#)

```
module.exports = function(Member) {  
  
  Member.username = (id, callback) => {  
    Member.findById(id, (error, members) => {  
      members = members.username;  
      callback(null, members);  
    });  
  };  
  
  Member.remoteMethod('username', {  
    description: 'Get username by member id',  
    accepts: [  
      { arg: 'id', type: 'number' }  
    ],  
    http: { path: '/username', verb: 'get' },  
    returns: { arg: 'username', type: 'string' }  
  });  
  
};
```



Custom API

- Filter 用法一

REST API 【取得特定username的member】

```
> http://localhost:3000/api/members?filter[where][username]=steve
```

大於(gt)、小於(lt)、AND及OR，[更多查詢類別](#)

```
> http://localhost:3000/api/members?filter[where][id][gt]=1
```



Custom API

- Filter 用法二

Node API 【取得特定username的member】

```
> http://localhost:3000/api/members?filter={ "where": { "username": "steve" } }
```

大於(gt)、小於(lt)、AND及OR，[更多查詢類別](#)

```
> http://localhost:3000/api/members?filter={  
  "where": {  
    "or": [ { "id": 1 }, { "id": 2 } ]  
  }  
}
```



Custom API

- Express.js RESTful API 用法 【寫入server.js】



```
const mariaDB = app.dataSources.mariaDB;

app.get('/api/member/username', (req, res) => {
  res.setHeader('Access-Control-Allow-Origin', '*');

  let sql = `SELECT username FROM member
    WHERE id = ${req.query.id}`;

  mariaDB.connector.execute(sql, (error, result) => {
    if (error) res.send(error);
    res.send(result);
  })
})
```



Access Controls List (ACL)

- 存取順序

拒絕所有人存取 → 特定用戶存取 → Guest存取

- 使用CLI建立ACL

```
> lb acl
> 選取要套用ACL項目的模型：message
> 選取ACL的範圍：所有方法和內容
> 選取存取類型：全部(符合全部類型)
> 選取角色：所有使用者
> 選取要套用的許可權：明確拒絕存取
```



Access Controls List (ACL)

- 存取順序

拒絕所有人存取 → 特定用戶存取 → Guest存取

- 使用CLI建立ACL

> lb acl

> 選取要套用ACL項目的模型：message

> 選取ACL的範圍：所有方法和內容

> 選取存取類型：全部(符合全部類型)

> 選取角色：任何已鑑別的使用者

> 選取要套用的許可權：明確拒絕存取



Access Controls List (ACL)

- 存取順序

拒絕所有人存取 → 特定用戶存取 → **Guest存取**

- 使用CLI建立ACL

> **lb acl**

> 選取要套用ACL項目的模型：**message**

> 選取ACL的範圍：**所有方法和內容**

> 選取存取類型：**讀取**

> 選取角色：**所有使用者**

> 選取要套用的許可權：**明確授予存取權**



Models Relation

```
{
  ...
  "acls": [
    {
      "accessType": "*",
      "principalType": "ROLE",
      "principalId": "$everyone",
      "permission": "DENY"
    },
    {
      "accessType": "*",
      "principalType": "ROLE",
      "principalId": "$owner",
      "permission": "ALLOW"
    },
    {
      "accessType": "READ",
      "principalType": "ROLE",
      "principalId": "$everyone",
      "permission": "ALLOW"
    }
  ]
}
```




Thank you 😊