1. MVC ?
- M-V-C ?
- M-V-P ?
- M-V-* ?
- M-V-Whatever

2. Feature
- Jeremy Ashkenas (Coffee Script, Underscore.js)
- RESTful Design 에 특화된 Model / Collection Structure
- View 의 Event Bind 와 Delegation 이 용이
- Model-View 가 철저히 분리되며, 이렇게 구성된 각 App 은 이벤트 기반 또는 상호참조 기반 연결 가능
- View 표현이 자유로우며, 높은 유연성과 확장성
- Based on JSON response
- _.template(Underscore.js) as client-template

3. Underscore.js
- Backbone.js 를 개발한 Jeremy Ashkenas 가 만든 JS Utility Library
- jQuery 와 유사해보이나, 전혀 다른 Library
- jQuery 가 주로 DOM manipulation (조정)에 초점을 두는 반면,
  Underscore 는 Array(models) 연산에 중점.
  (SQL Query 유사점)
- jQuery : $.method() = Underscore:_.method()

4. Backbone.js 구조
- Backbone.Model
- Backbone.Collection
- Backbone.View
- Backbone.Router
- Backbone.Sync
- 기타 등등

5. Backbone.Model
- 개별 단위의 Data 집합체로서 가장 중요한 역할
- Collection 에 의한 생성된 한개의 Model 은 View 와 서로 참조 연결하여 사용하면 효과적
  (Model -> View || View -> Model)
- 각 Model 은 CRUD 를 위한 고유의 URL 을 가진다.
  (기본값 : collection.url/model.id)
  ex) collection 의 url 이 "/notes" 이고, 이에 속하며, id(pk value)가 10dls model.url "/notes/10"

정리
- **엔터티 모델 + DAO**
- **엔터티 속성은 해시 객체**
- **모델 변경에 따른 이벤트 발생**
- **저장소와 동기화 (with Backbone.sync)**
- **기본 동기화 방식 : RESTful HTTP(with jQuery.ajax)**

6. Backbone.Model 의 OOP 적인 적근

- 개별 노트의 정보에 해당

```
{
    note_idx: 1,
    note_content: "메모를 입력해봅시다!",
    note_width: 300,
    note_height: 300,
    note_x: 100,
    note_y: 120,
    note_zIndex: 1,
    note_color: "#FFF259",
    note_regidate: "0000-00-00 00:00:00",
    note_modidate: "0000-00-00 00:00:00"
}
```

- Code

```
var Memo = Backbone.Model.extend({
    "url" : "",      // 따로 지정하지 않는 경우, collection.url/model.id 로 기본 지정됨
    "idAttribute": "note_idx",      // PK 역할을 하는 attribute (model.id 로 가져올 수 있다)
    "default" : {     // 새로운 Model 생성시, 초기 값을 지정, fetch 후 server 측 data 로 override.
        "note_content": "",
        "note_x": 0,
        "note_y": 0,
        "note_width": 250,
        "note_height": 250,
        "note_color": "#EEEE8C",
        "note_regidate": "0000-00-00 00:00:0)"
    }
});
```

7. Method of Backbone.Model

1) model.**fetch()** **[GET]**
- 개별 model 을 url 을 호출하며, response 를 attribute 에 저장
- "Change" event 를 발생

2) model.**save(attr)** **[POST / UPDATE or PATCH]**
- 전달된 Attributes 들을 JSON Format 으로 model.url 에 Ruquest
- "Change" event 를 발생
  이미 create 된 model 을 save 로 수정하면, 기본적으로 모든 Attribute 를 정리해서 UPDATE Request 함
  하지만 options 에 patch:true 를 주게되면 새로 지정된 값만 정리해서 PATCH Method 로 보냄

3) model.**destroy()** **[DELETE]**
- 스스로를 삭제하며 model.url 로 delete method 를 전달
- "destory" event 를 발생시킴

4) model.**get("attr")**
- 특정값을 선택하여 반환

5) model.**set("attr")** / model.**unset("attr")**
- 특정값을 지정/추가하거나 속성 자체를 제거

6) model.**has("attr")**
- 특정 값을 가지고 있는지에 대해 Boolean 반환

7) model.**clear()**
- 해당 모델이 가진 모든 속성과 idAttribute 를 제거

8) model.**toJSON()**
- 속성의 key-value Data 를 JSON String Format 으로 반환

9) API 정리

- extend
- constructor / initialize
- get / set
- escape
- has
- unset
- clear
- id
- idAttribute
- cid
- attribute
- changed
- defualts
- toJSON
- sync
- fetch
- save
- destroy
- validate
- validationError
- url
- urlRoot
- parse
- clone
- isNew
- hasChanged
- changed Attributes
- previous
- previous Attributes

8. Backbone.Collecton

- 서버에 요청하여 받은 JSON array 의 개별 원소를 각각 Model instance 로 생성하며, collection.models 에 push 함
- default 상태에서 개별 Model 들은 **{collection.url} / {model.id}**라는 model.url 을 가짐
- 개별 Model 을 add, create, remove 할 수 있다.
  이미 만들어진 Model 은 스스로의 URL 을 통해서 서버와 통신하며, 스스로도 destroy 가능
- 각 Model 로부터 발생하는 Event 는 Collection 에도 전달

정리

- "Model"의 집합 + DAO
- 목록 변경에 따른 이벤트 발생
- 풍부한 순회함수 with Underscore.js
- 저장소와 동기화 with Backbone.sync
- 기본 동기화방식은 RESTFul HTTP (with jQuery.ajax)

9. Backbone.Collection 의 OOP 적인 적근

- 개별 노트들의 집합

```
[
    {
        note_idx: 1,
        note_content: "메모를 입력해봅시다!",
        ...
    },{
        note_idx: 2,
        note_content: "다음 모바일 첫화면",
        ...
    },{
        note_idx: 3,
        note_content: "키키키카~ Devon Forever!",
        ...
    }
]
```

- structure

```
var Collection = Backbone.Collection.extend({
    "url": "/notes",
    "model": Memo, // 앞에서 지정한 "Memo" 구조를 개별 Model 의 기본으로 가짐
    "initialize": function(){}
})
```

10. Methods of Backbone.Collection

1) collection.**fetch()** **[GET]**
- 자신의 url 을 호출하며, response 를 각각 새로운 Model Instance 로 변환하여 collection.models 에 Push
- reset:true 옵션을 주면 "reset" event 를 발생함

2) collection.**add(attr)** / **create(attr)** **[POST]**
- 전달된 argument 로 구성된 새로운 Model 생성
- "add" / "request", "sync" event 발생함
  add : 새로운 Model 을 Collection 에 추가만 함
  create : 추가함과 동시에 Backbone.sync 를 통하여 Post Request

3) collection.**remove(model)** **[DELETE]**
- 하나 또는 array 단위의 Model 을 Collection 에서 삭제
- "remove" event 를 발생시킴

4) collection.**push / pop / unshift / shift (model)**
- 전달된 Model 에 대하여 Collection 에 추가

5) collection.**sort()**
- collection.comparator 로 지정된 Attribute 의 Value 순서로 Sorting

6) collection.**where / findWhere (option)**
- 특정 값을 가진 Model 들만 골라서 반환 (findWhere 는 limit 1 역할)

7) collection.**clone()**
- 해당 collection 를 새로운 instance 로 반환

8) collection.**parse(respone)**
- API 로부터 응답받은 Raw Response 에 대하여 어떤 부분을 가져올지 지정하여 반환함
  기존 REST API 가 이미 있으나, Response 를 변경하기 어려운 상황에서 유용함

9) API 정리
- extend
- model
- constructor / initialize
- models

- toJSON
- sync
- add
- remove
- reset
- update
- get
- at
- push
- pop
- unshift
- shift
- slice
- length
- comparator
- sort
- pluck
- where
- url
- parse
- clone
- fetch
- create
- 28 Underscore.js Methods Support

```
var methods = ['forEach', 'each', 'map', 'collect', 'reduce', 'foldl',
'inject', 'reduceRight', 'foldr', 'find', 'detect', 'filter', 'select',
'reject', 'every', 'all', 'some', 'any', 'include', 'contains', 'invoke',
'max', 'min', 'sortedIndex', 'toArray', 'size', 'first', 'head', 'take',
'initial', 'rest', 'tail', 'last', 'without', 'indexOf', 'shuffle',
'lastIndexOf', 'isEmpty']
```

11. Backbone.View

- 순차 상 Backbone Application 실행의 첫 시작
  View(parent) -> Collection -> Model -> View (child)
- view.el : 해당 View 가 바라보게 되는 요소 선택
  따로 지정하지 않을 경우 plain DIV 가 생성됨
- view.events : $el 안의 요소의 event 와 handler 를 key-value 형식으로 delegation 적용
- 기본적으로는 HTML/CSS 에 대한 어떠한 실력도 행사하지 않으며, 지정된 View 를 논리적으로 관리
  underscore.js 의 _.template 을 이용하며, DOM Fragment 를 줄이는데 효과적임
- Server 와의 통신은 담당하지 않음

정리

- "Presenter" in MVP
- 모델 / 컬렉션 조작 & 이벤터 처리
- DOM 조작 & 이벤트 처리
- 템플릿 처리 (선택 사항)
- 완전 수동 ~ 헉~

API 정리

- extend
- constructor / initialize
- el
- $el
- setElement
- attribute
- render
- remove
- delegateEvents

- undelegateEvents
- $

12. Model + View Render

1) model

```
{
    "name": "이효리",
    "birth_date": "1979년 5월 10일 (만 33세)",
    "birth_place": "충북 청원군",
    "group": "핑클",
    "family": "언니 이애리, 언니 이유리",
    "debut": "1998년 핑클 1집 앨범 'Blur Rain'",
    "img_url": "http://i2.daumcdn.net/thumb/S160x160/2224.jpg",
    "school": [
                "경희대학교 언론정보대학원",
                "국민대학교 연극영화학",
                "서문여자고등학교",
                "서문여자중학교",
                "동작초등학교"
                ]
    ...
}
```

2) Template

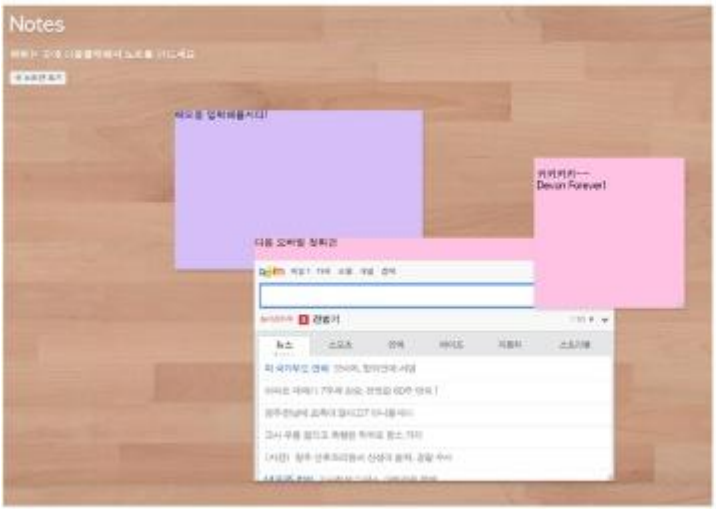
```
<script type="text/template" id="profileTemplate">
    <div class="profile_img">
        <img src="<%= img_url %>" alt="<%= name %>">
    </div>
    <div class="profile_desc">
        <a href="<%= music_link %>"><%= name %></a> <%= category %>
        <dl>
            <dt>출생</dt>
            <dd><%= birth_date %>, <%= birth_place %></dd>
            <% if(group !== undefined){ %>
            <dt>그룹</dt>
            <dd><%= group %></dd>
            <% } %>
            <dt>학력</dt>
            <dd>
            <% for(var i,len=school.length; i<len; i++){ %>
                <%= school[i] %>
            <% } %>
            </dd>
            ...
        </dl>
    </div>
</script>
```

3) View

```
var View = Backbone.View.extend({
    "initialize": function(){
        //..생략
        this.template = _.template($("#profileTemplate").html());
        this.render();
        //..생략
    },
    "render": function(){
        var content = this.template( this.model.toJSON() );
        this.$el.html(content);
    }
})
```

4) View 구조

13. Backbone.View Code 예제



Model * n : 개별 노트 Data

Collection : 노트 Data 집합

View (Child) * n : 개별 노트 View

View (Parent) : 전체를 감싸는 View

1) 개별 노트 View

```
var NoteView = Backbone.View.extend({
    "tagName": "div", //개별 View를 wrapping할 DOM tagName 지정
    "events": { //.content element에 대해 contentClickHandler 실행
        "mousedown .content": "contentClickHandler"
    },
    "initialize": function(options){ // new로 생성되는 시점에 실행할 부분
        _.extend(this, options);
        this.template = _.template($("#noteViewTemplate").html()); //개별노트의 템플릿 지정
        this.model.view = this; //model -> view로도 접근할 수 있도록 상호참조
        this.model.on({ //model에 대한 이벤트 등록
            "destroy": this.destroyHandler,
        });
        this.render();
    },
    "render": function(){
        this.$el.html(this.template(this.model.toJSON())); //템플릿 출력
    },
    "contentClickHandler": function(){ }
});
```

□□

2) Structure (Parent View)

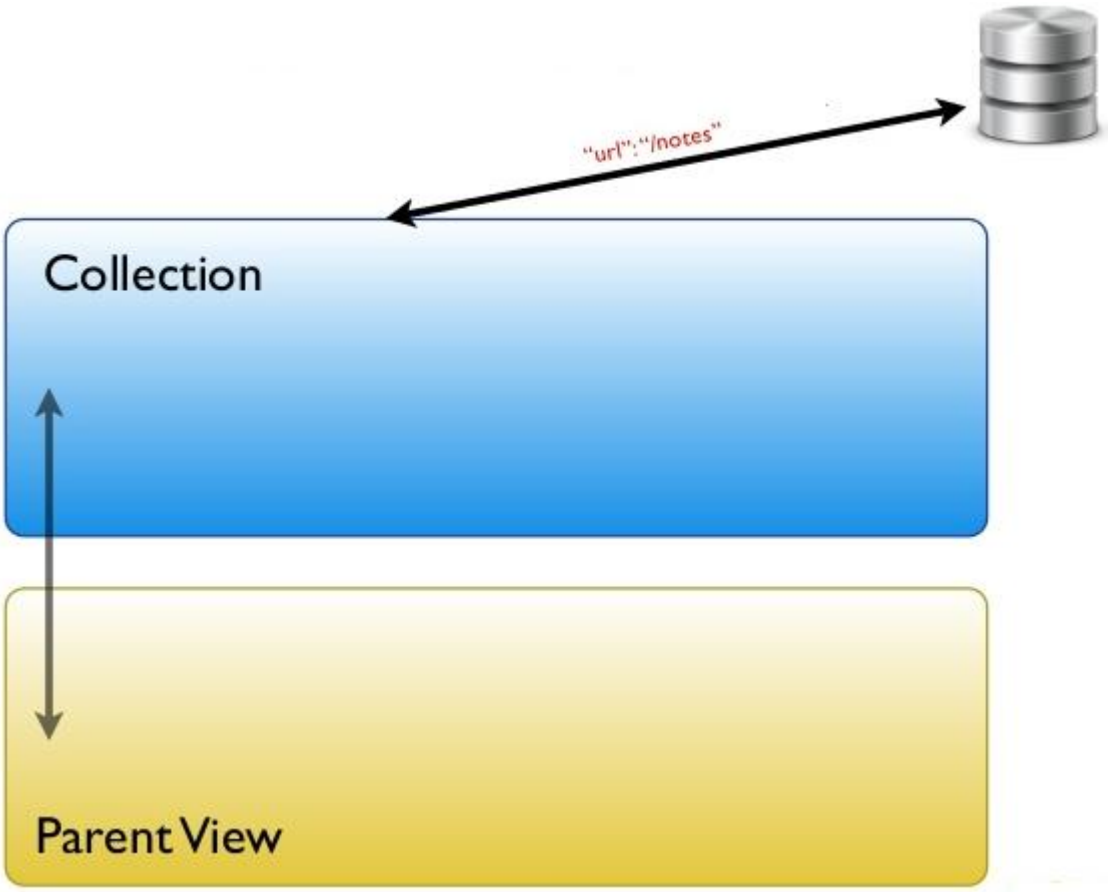
```
var NoteApp = Backbone.View.extend({
    "el": $("body"), //기존에 존재하는 element에 적용시킬 경우 'el'로 지정함
    "events": {
        "dblclick": "createNote"
    },
    "initialize": function(options){
        _.extend(this, options);
        _.bindAll(this, "render", "append", "createNote");

        this.collection = new Collection(); //새로운 콜렉션 생성
        this.collection.on({
            "reset": this.render, //콜렉션이 새로운 데이터들로 갱신될 때(reset)
            "add": this.append, //콜렉션에 새로운 데이터가 추가되었을 때
            "remove": this.collectionRemoveHandler //콜렉션에 개별노트 삭제가 발생될 때
        });

        this.collection.fetch({ //콜렉션에 지정된 url로 새로운 데이터를 받아와 reset.
            reset: true
        });
    },
```

```
    "render": function(){
        this.$el.find(".desktop").html("");
        this.collection.each(this.append); //collection의 each method : model을 인자로 넣어 loop.
    },
    "append": function(model){
        var note = new NoteView({ //개별 입력받은 model을 넣어 새로운 node object 생성
            "model": model
        });
        this.$el.find(".desktop").append(note.$el); //note의 $el을 parent view의 영역에 append
    },
    "createNote": function(e){
        var newModel = {
            "note_x": e.clientX,
            "note_y": e.clientY,
            "note_zIndex": this.collection.length
        };
        this.collection.create(newModel,{ //콜렉션에 새로운 Model을 추가함과 동시에 POST request.
            success: function(model, response){
                model.view.contentClickHandler();
            },
            error: function(){}
        })
    },
    "collectionRemoveHandler": function(){}
});
```

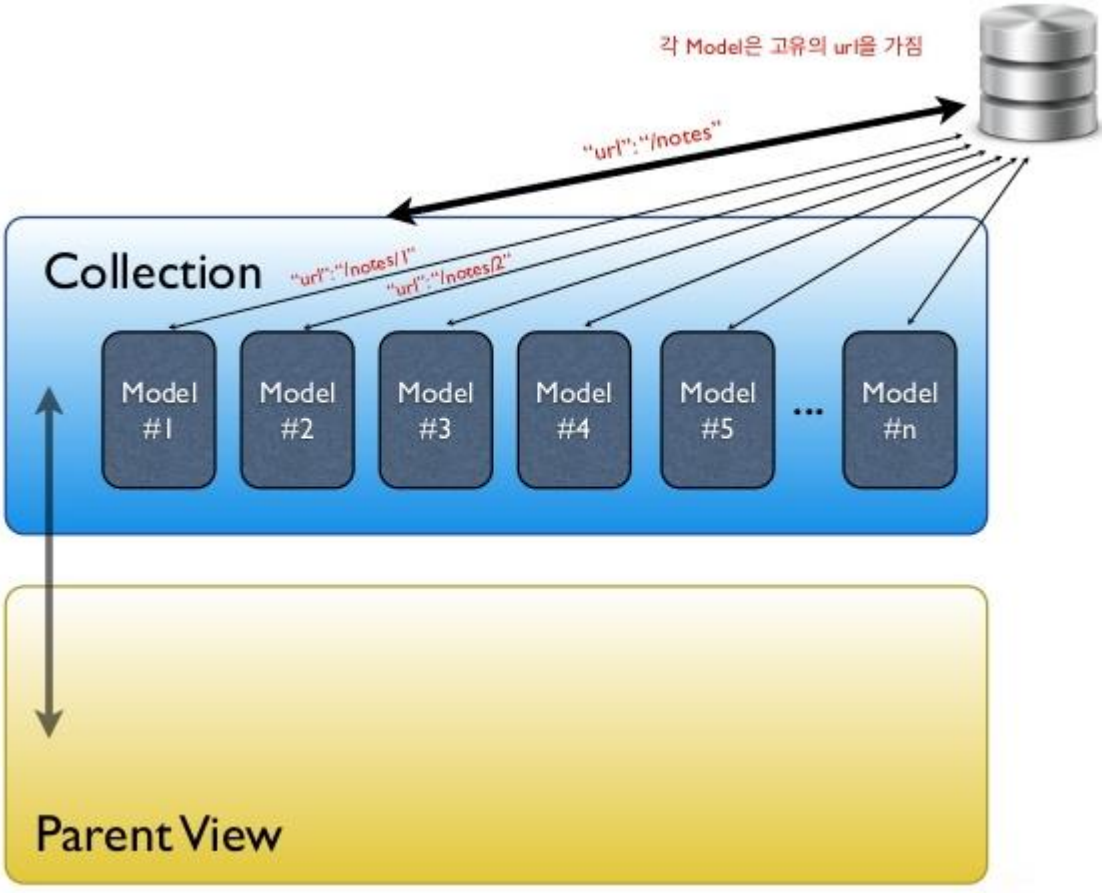
14. 실행 순서 **(Sequence of Execution)**

1) Container 역할을 하는 하나의 **View (parent)** 생성

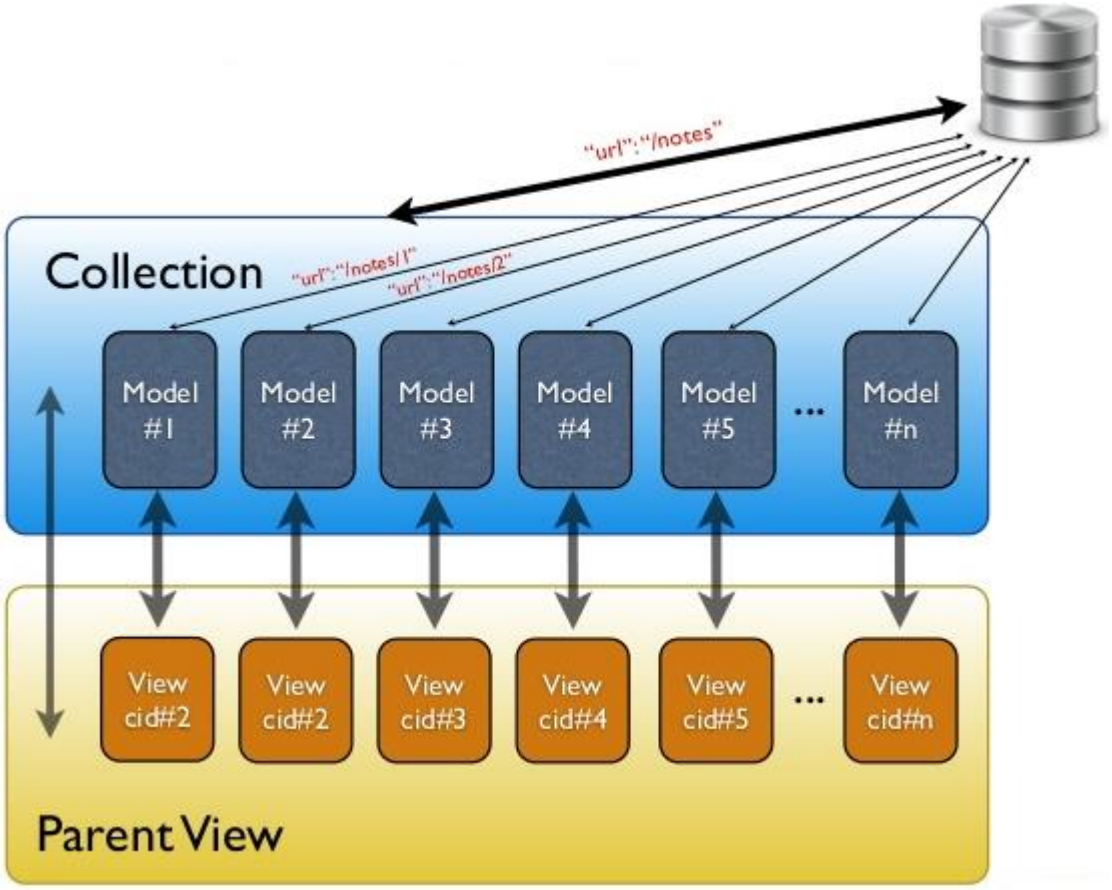
2) View (Parents)에서 **Collection** 을 생성하고 **Fetch** 시켜 **JSON Data** 를 요청함



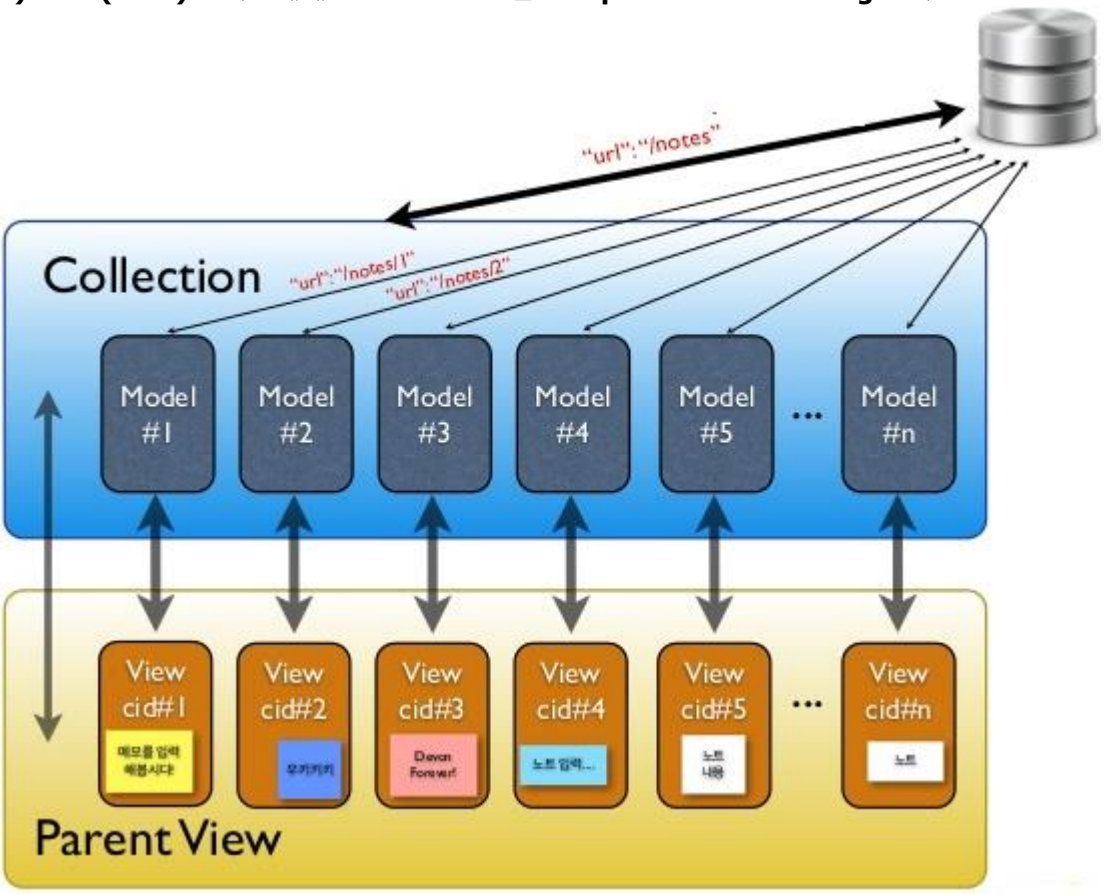
3) Collection 은 받아온 **JSON Data** 를 각각 개별 **Model** 로 생성



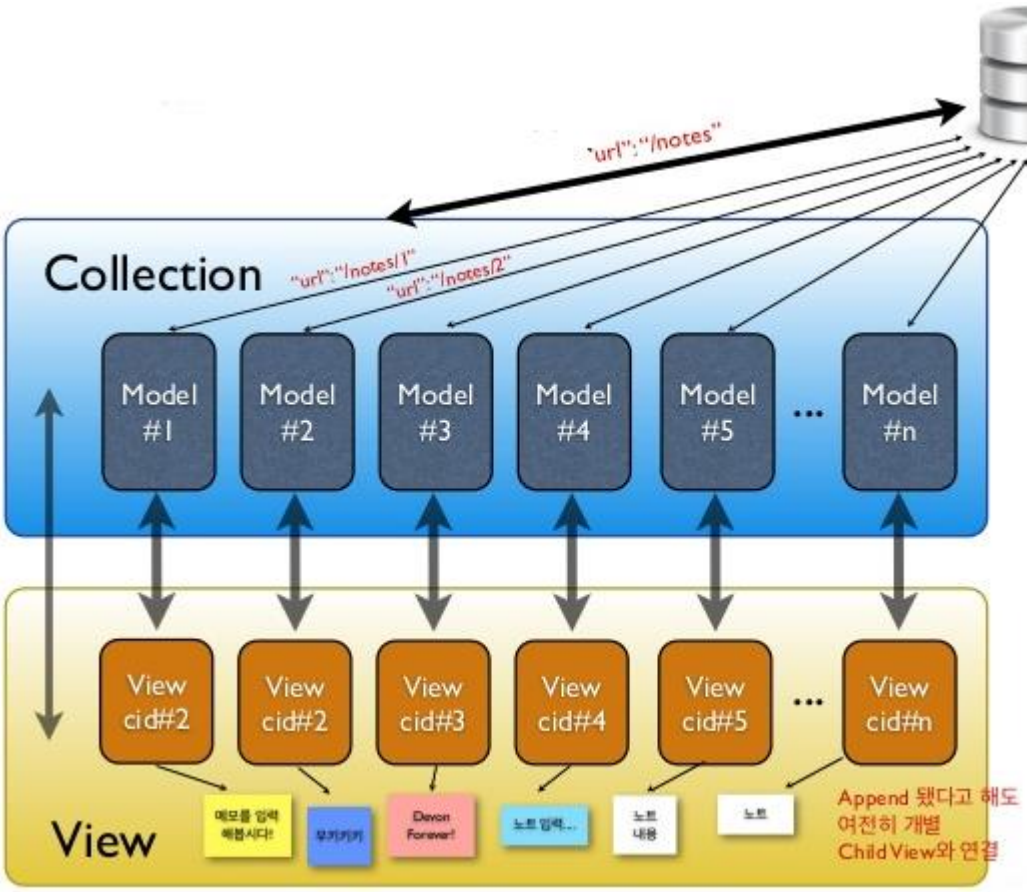
4) View(parent)는 **Collection.length** 만큼 **View(child)** 생성**.** 이 때 각 개별 **Model** 을 참조연결

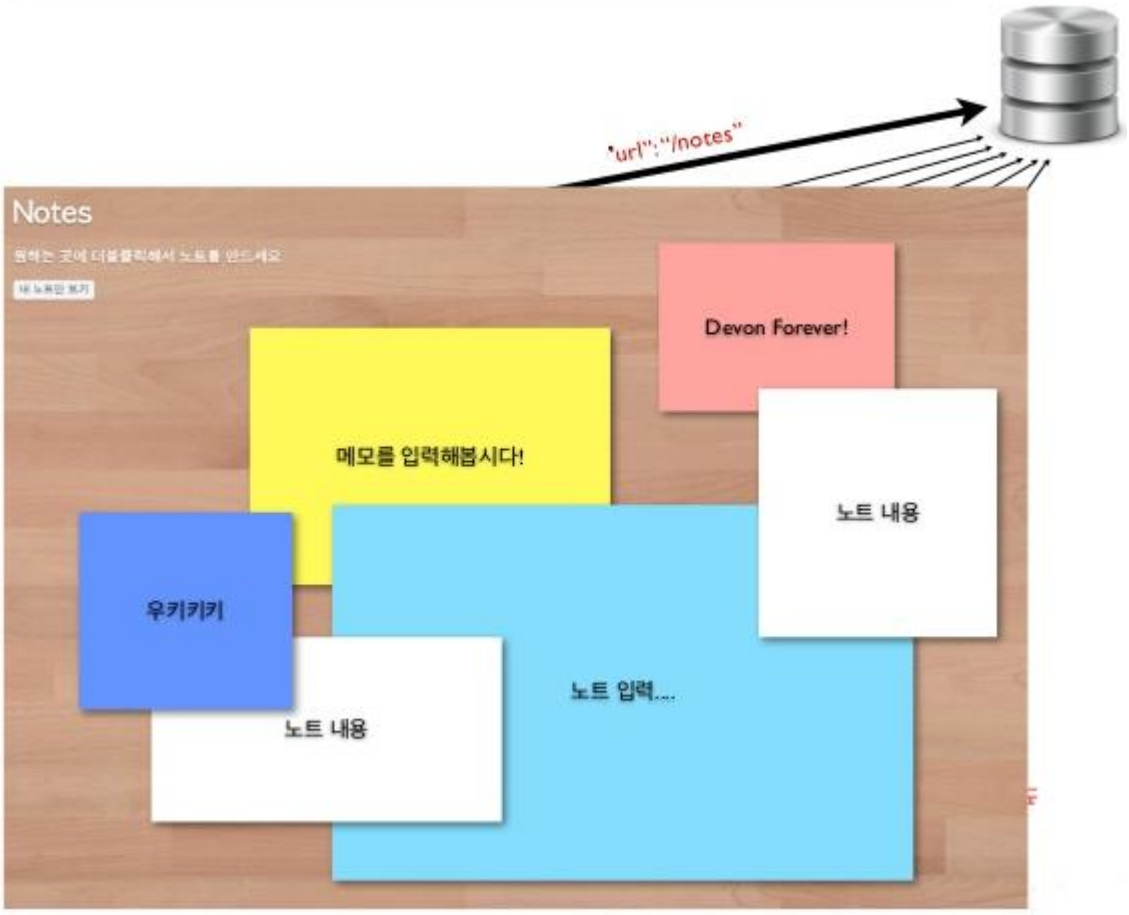


5) View(child)는 자신에게 참조된 **Model** 을 **Template** 으로 **Rendering** 한다**.**

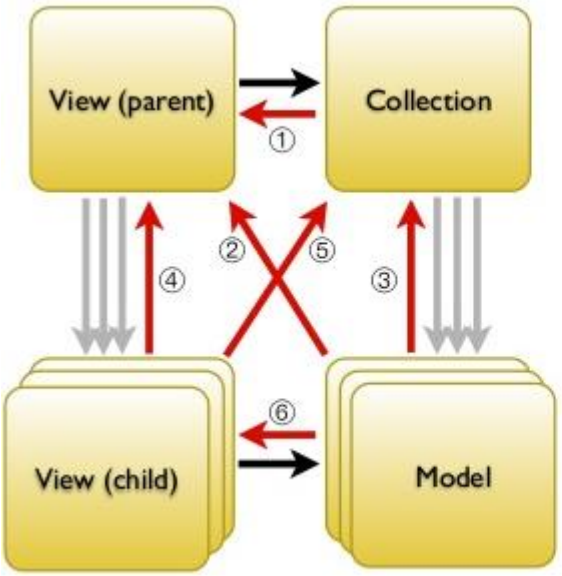


6) Parent View 는 **Child View** 의 **$el** 을 자신의 **Element** 원하는 곳에 **Append**





15. Cross Reference By "extend()"

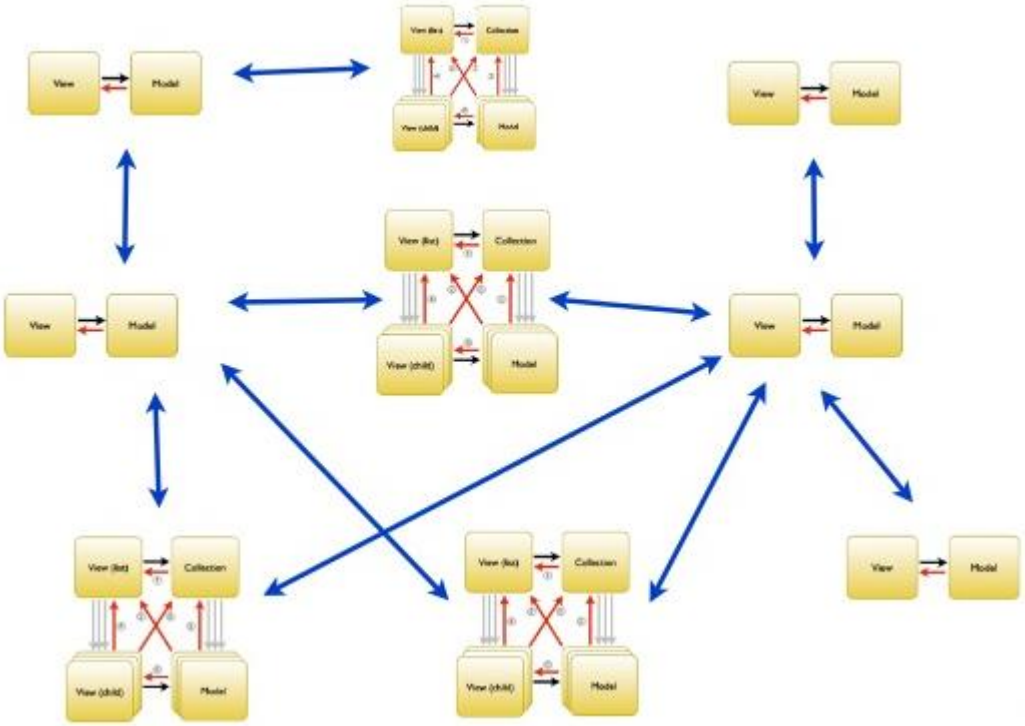


View (parent)에서...

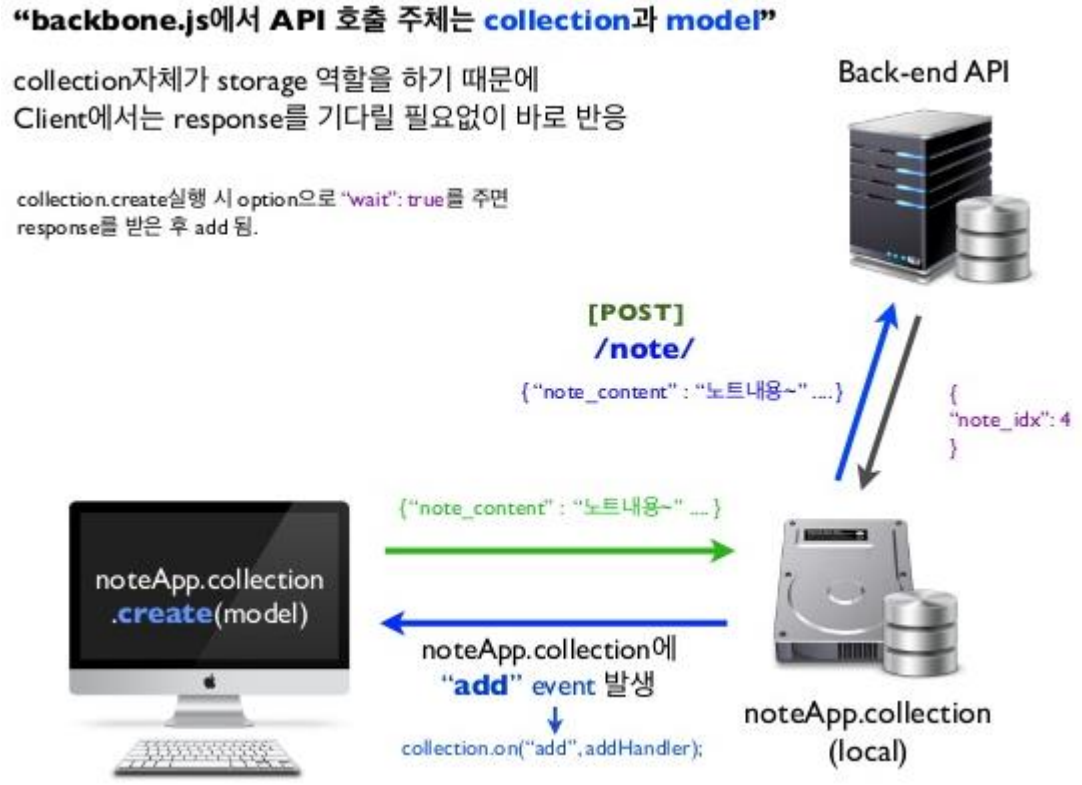
```
this.collection = new Collection({
    "list": this ①
});

//Collection을 loop돌며 개별 Child View 생성 시
model.list = this ②
model.collection = this.collection; ③

var view = new View({
    "model": model,
    "list": this, ④
    "collection": this.collection ⑤
});
```

View (child)에서...

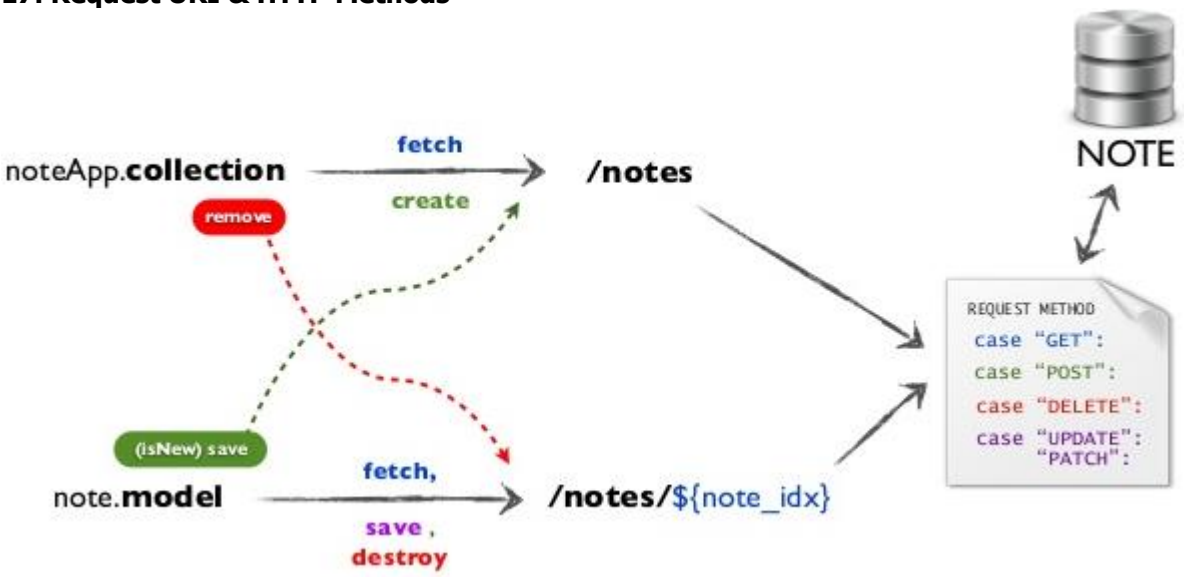
```
this.model.view = this; ⑥
```



16. Backbone.js Sync with RESTful API



17. Request URI & HTTP Methods



18. Collection, Model 의 Built-in Event

- "add" : collection.add(model) or collection.create(model) **[POST]**
- "remove" : collection.remove(model) **[DELETE]**
- "rest" : collection.fetch(model, {reset:true}) or model.fetch(model, {reset:true}) **[GET]**
- "change" : collection 에 모든 종류의 CRUD 발생 시 / model 에 attribute 변경 시 **[UPDATE/PATCH]**
- "change:[attribute]" : model 의 특정 attribute 변경 시
- "destory" : model.destroy() **[DELETE]**

19. Backbone.Event Callback 처리

Case) jQuery Draggable로 노트의 위치를 변경했을 때,
개별 노트의 x/y 정보를 곧바로 비동기로 DB에 적용해야하며,
response된 DB측 변경일시(note_modidate)를 Model에 반영
해야 하는 경우.

```
var NoteView = Backbone.View.extend({
    "initialize": function(options){
        _.extend(this, options);
        // ..생략..
        this.model.on({
            "change:note_modidate": this.modidateChangeHandler
        });
        // ..생략..
    },
    "render": function(){
        this.$el.html(this.template(this.model.toJSON()));
        this.initUI();
    },
    "initUI": function(){
        var note = this;
        this.$el.draggable({
            //..생략
            stop: function(event, ui){
                note.model.save({
                    "note_x": event.position.left,
                    "note_y": event.position.top
                },{
                    patch: true,
                    success: function(model, response){
                        model.set({
                            "note_modidate": response.note_modidate
                        })
                    }
                })
            }
            //..생략..
        })
    },
    "modidateChangeHandler": function(){
        console.log(this.model.get("note_modidate"));
    }
});
```

① ② ③

model에 "change"와 "change:note_modidate" Event를 trigger

20. Backbone.js 의 이점 및 결론

1) 회고

- 겉껍데기(Mark-Up)를 만들어 낸 뒤 영혼(JSON Object or XML)은 어디론가 가버린다.
- 껍데기 어딘가에 늘 이름표를 달아두어야 하며, 개발자는 "이럴 땐 이거하고, 저럴 땐 저거해"를 긴밀하게 늘어놓는다.
- 특질있는 영혼 (Model)이 연결된 육체(View) 그리고 그것들이 모인 리얼월드!
- 애플리케이션 개발자는 신이 되어 Model/Collection 의 변화에 대해 View 가 어떻게 반응할지 느슨하게 설정한다.
- 그 후의 일들은 Instance 들이 알아서 반응한다.

2) Javascript OOP

- Backbone.js 는 사용자가 Javascript OOP 를 완전히 이해하고 있다는 것을 전제로 구현되었음

- 매뉴얼도 매우 불친절하고 부실한 것 같음
  (Jeremy Ashkenas 는 심지어 method 별 options 을 만들어 놓고도 이를 매뉴얼에 기재하지 않음 것이 많음)
- 따라서 이를 이해하고 학습하는 과정 자체가 JS OOP 를 이해하는 과정이 됨

3) Model-View 간섭
- Model 과 View 가 서로 참조하며, Event 에 따라 각자의 Handler 가 실행될 뿐 완벽히 구분지어지며,
  Memory Address 에 의한 Reference 참조일 뿐이므로 Performance 관리에 효과적임
- 또한 Backbone.js 는 여타 JS MVC Framework 과는 달리 View 단을 간섭하지 않음

4) 엄청난 개발 Performance
- Model 과 Collection, View 가 유기체처럼 행동하며, 개발자는 이 행동을 제어만 하면 됨.
- 그리고 Application 사전 설정단계에서 각 소속 Object 들의 추상화와 Module 화를 적당히 신경써서 구현할 경우,
  개별 객체는 서로간 느슨하게 연결되고 자신만의 기능에만 초점을 둠
- 따라서 유지관리 시 어마어마한 편의성과 그에 따른 개발 Perfomance 향상이라는 장점이 있음
- 애자일 환경에서 매우 큰 효과가 있음

5) 아름다운 소스 코드
- Backbone.js 를 의도와 목적에 맞게 잘 작성한다면, Source Code 는 그 자체로서 매우 아름다운 개발 문서가 된다.

21. Backbone.js 사용시 고려할 점
- Backbone.js 의 기능을 몇 퍼센트나 사용할 것인가?
  : project 전체에 사용할 필요는 없지만, 사용하고자 하는 부분에 data 를 fetch 해 와서 template 만 뿌려주는 식으로 Backbone.js
  기능의 반만 사용할 것이면 장점은 장점대로 사라지고 Code 가 혼잡해 질 것이다.

- 자유롭다, 하지만 손이 많이 갈 것이다.
  : View 단을 자유롭게 꾸밀 수 있는 큰 장점이 있지만, 이는 결국 CSS/MarkUp 전문가가 필요

- Server 측 API 가 진정한 RESTful 한가?
  : defualt 상태에서 backbone.js 의 자동화된 비동기 호출은 Addressable URI 를 기반으로 Request Method 를
  GET/POST/DELETE/UPDATE/PATCH/PUT 등을 달리하여 실행된다.
  GET/POST 만 사용하는 환경에서는 효율적인 사용이 어렵다.

- Logical Flow 정리 필요
  : 각 기능단위 내에서 Backbone.js 코드는 그 자체로 아름다운 문서이지만, 이러한 Objects 또는 File 간의 관계를
  따로 정리해 놓지 않으면 차후 유지보수가 어려울 수 있다.

- Template 보안 이슈
  : JS Template 은 기본적으로 화면에 Template 을 뿌려놓은 상태에서 시작한다. 때문에 Template 내에서는 가급적 연산을
  지양하며, 노출되지 민감한 부분은 비동기 호출하거나 애초에 서버에서 parse 시킨 후 include 하는 방식으로 우회 필요

22. 참고

- URL : http://backbonejs.org
- URL : http://nodeqa.com/nodejs_ref/53#QmFja2JvbmUuanMgI+ynhOynnOuniOyngOuniSDstJ3soJXrpqw=
- URL : http://www.slideshare.net/gyutaejo/backbonejs-m-v
- URL : https://egghead.io/tags/AngularJS
- URL : http://nodeqa.com/nodejs_ref/49