

Webデザイン実習4B

2019/10/08

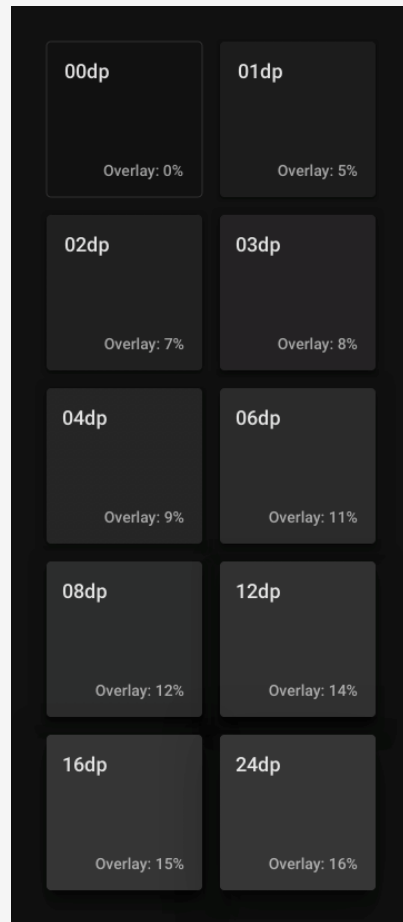
Kazuma Sekiguchi
class@cieds.jp

DarkUI

- iOS13から利用可能
 - Androidも10から利用可能
 - macOSやWindows10でも採用されてきている
- アプリのUIもDarkUIに対応する必要がある
 - 背景を黒にすれば良い、ということではない（純な黒ではないし）
- CSS Media Queryで定義が可能
 - @media (prefers-color-scheme: dark)
 - IEとEdgeは非対応
 - 上書きする形で適用する
- ドロップシャドウなどは使えない表現
 - 彩度も少し落とした方が見やすいとされる
- 液晶が明るくなってきたせい、ともされるが・・・

DarkUI

Light	Dark
<div>R 0 G 122 B 255</div>	<div>R 10 G 132 B 255</div>
<div>R 142 G 142 B 147</div>	<div>R 152 G 152 B 157</div>
<div>R 52 G 199 B 89</div>	<div>R 48 G 209 B 88</div>
<div>R 88 G 86 B 214</div>	<div>R 94 G 92 B 230</div>
<div>R 255 G 149 B 0</div>	<div>R 255 G 159 B 10</div>
<div>R 255 G 45 B 85</div>	<div>R 255 G 55 B 95</div>
<div>R 175 G 82 B 222</div>	<div>R 191 G 90 B 242</div>



Elevation overlay transparencies range from 0% for the lowest level to 16% for the highest level.

Elevation level	White overlay transparency
00dp	0%
01dp	5%
02dp	7%
03dp	8%
04dp	9%
06dp	11%
08dp	12%
12dp	14%
16dp	15%
24dp	16%

- Appleのガイドライン
 - LightとDarkで色が少し違う点に注意
 - Darkの方が少し彩度が低い
- Googleのガイドラインでは上に存在するものほど明るくする、とされる
 - dpが高さ
 - GoogleのMaterialDesignは階層を持つ

管理画面の作成

- 管理画面はセッションでログインをしているかどうかを判定する
 - ログインしていない場合はログイン画面に遷移させる
 - ログインしている場合は、必要な画面を表示させる
- セッションの管理が重要
 - ログアウト時はセッションを破壊してログアウトとする

管理画面の作成

- 通常の機能

- 登録 (Create)
- 読み出し (Read)
- 更新 (Update)
- 削除 (Delete)

の各機能が必要

1つの機能を作るために4つの機能が必要

- 実際には削除はしないことが多い (論理削除)

- 削除フラグなどを立てて、削除したと見なすようにする
- 万が一の際にDBにアクセスして復旧させることが可能

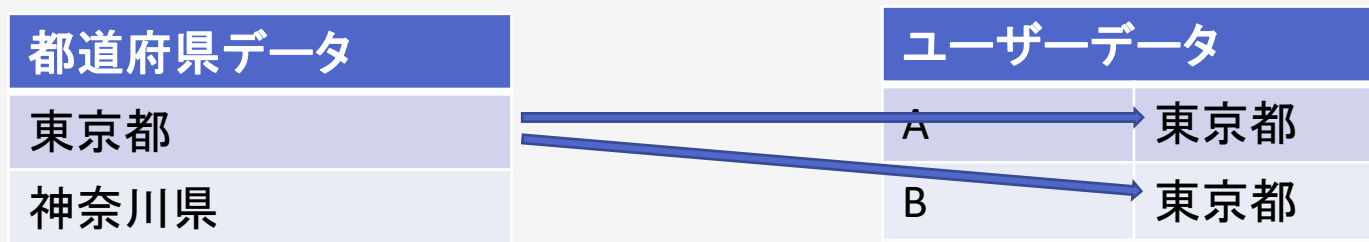
管理画面の作成

- 更新

- 作成するのが面倒なものの1つ
- 既に登録されているデータを取得して適切な場所に表示させる
- アップデート時に新規登録時と同様の値のチェックが必要
- どれを更新させるかキーとなる値を付与して、更新をするプログラムにデータを渡す
 - キーとなるもの=ほとんどはID
- キーを利用してDatabase内のデータを更新する
 - 更新時にキーを利用する点に注意

データベースの構造

- 基本的にデータベースは 1 対 1 の関係にあるデータを保存する
 - 例：Aという項目のデータがZ
- 1 対Nの関係にあるデータは工夫しないと保存できない
 - 複数のテーブルを組み合わせることで表現を行っていく



- この関連性をきちんと表現していくことが必要
 - 正規化

JSON

- JSでデータを受け取ったりする場合展開が容易なため良く使われているファイル形式
 - JSのオブジェクトと同じような形式で記述
 - jQueryで展開が容易
 - 最近はJSでも展開が容易にできるように
 - 旧来はXMLでデータを取得していたが、JSONで取得するのがほとんど
 - ex:Twitterのデータ、Amazonのデータなど

```
{  
    "name": "Dreamweaver",  
    "version": "CC",  
    "price": "4,000/month"  
}
```

```
[{  
    "name": "Dreamweaver",  
    "version": "CC",  
    "price": 4000  
},  
{  
    "name": "Photoshop",  
    "version": "CC",  
    "price": 4000  
}]
```


クロスドメインセキュリティ

- Ajaxの場合、後からデータを取得可能なため、ドメインを跨がって取得することも可能
 - 最初にデータを取得した場所のURLではなく、別のURLからなはずなのだが、ブラウザのセキュリティにより不可能にしているブラウザが多い
- ユーザに見せているHTMLと後から取得するデータのドメインが同じでないとデータを取得出来ない、ことがある
 - Firefox , GoogleChromeなど
 - ドメインを跨ぐならJSONPを利用する

JSONP

- 基本的にはJSONと同じ
 - ブラウザーのセキュリティで後からデータを落とす場合
（JSONは後からデータをJS側で取得させる）元のサーバと同じURLからしか落とせない
 - 別サービスにアクセスしてJSONデータを取得してごにょごにょするのが無理

JSONP

- <script>タグで呼び出せば、ほかのサーバからでも落とせる
- JSONではデータなので、落としても上手くいかない
 - 関数の形でJSONを呼び出す
 - 呼び出すURLの後にgeo.php?callback=?みたいにcallbackを付けてJSONを呼び出す
 - このタイプのJSONがJSONP
 - 処理は変わらない
 - jQueryなら\$.getJSONで処理できる
- Ajaxでは当たり前のもの（Pjax）
 - 末尾はXMLなのにね

\$.getJSON

- JSONを非同期的にダウンロードするjQueryの機能
 - JSONの形式を展開してくれるため、非常にあとの処理が楽
 - 現在Ajaxというと、データはJSONが多いので、良く使う機能

```
[{  
  "name": "名前1ですよ",  
  "address": "住所ですよ"  
},  
{  
  "name": "名前2ですよ",  
  "address": "住所ですよ"  
}]
```

JSON

```
$.getJSON("JSONまたはJSONPのURL",function(json){  
  for(var i in json){  
    $("#contents").append("名前:"+json.name+"住所"+json.address);  
  }  
});
```


ノンリロードコンテンツ

- 通常のウェブページの場合、ページ遷移などを行った場合は、一度ブラウザーの画面を初期化（真っ白）にしてデータを読み込み、再度レンダリングを行って表示を行う
 - 非常に手間の掛かるプロセスを経ている
 - ステートレスなため、セッションなどを利用している場合は、変数などをCookieなどで保持しておく必要がある
- Ajaxなどの技術を使うと後から必要なデータを取得することが可能
 - この場合、JSでデータだけを取得してくるため、ブラウザーが初期化されることがない＝リロードがない＝ノンリロード

JSで取得

- JSでデータを取得したら、JSでデータをパース（使えるようにデータを加工する）して、適切な位置にデータを挿入または置換して入れ込む
 - 更新、変更のあった部分だけ修正、アップデートすることが可能
 - ユーザはその間も操作することが可能
 - ユーザが操作できる＝使えてしまう、という点は注意が必要
 - 読み込み完了前にも操作できてしまうため、誤作動の原因になることも
 - 読み込み完了したら画面表示するようにするなどの工夫も必要

データの送受信

- 昔はXML形式で実行
 - 現在はJSONまたはJSONPで行うのが通常
 - JSONであれば、JSでのパースが極めて容易かつ軽量
 - バイナリデータ（画像データなど）は含められないので、画像を使う場合は、URLを指定しておき、改めてJS側で取得し直すようにする
- やりとりは全てバックエンドで行われるため、どのURLにアクセスしてどういうデータを取得しているか、は開発者ツールなどで確認する必要がある
 - 正しいJSONが返ってきているかどうかなどを確認していく

何が面倒か

- HTMLとCSSの場合、全てDOMに依存して表示を行う
 - テンプレートエンジンを用いても最終的にはDOM依存
 - クラス名やID名、タグの順番などに依存してしまうため、ちょっとした変更にも極めて弱い
 - 特にjQueryで作成した場合は、顕著
 - 各パーツをできるだけ分離することで、ほかの部分に影響を及ぼさないような設計を
 - CSSとの兼ね合いも気をつける
- 最近はReactやらVueなどを使って解決するのが一般的
 - DOM自体をJSに状態を保持させる

jQueryの\$.ajax

- 細かい制御を行うことのできるAjax通信用メソッド

```
$.ajax({  
  type: "POST",  
  url: "some.php",  
  dataType: "json", //JSONデータを受け取る場合指定  
  timeout: 10000, //タイムアウトにする秒数(ミリ秒)  
  data: { name: "John", location: "Boston" }, //送り出すデータ  
  success: function(){  
    //データを受信できた場合の処理  
  },  
  error: function(){  
    //データを受信できなかった場合の処理  
  }  
});
```

タイムアウト処理を入れた場合

```
$.ajax({  
    type:"GET",  
    url: http://wbclass.net/json/json.php,  
    cache: false,  
    timeout: 10000,  
    success: successHandler,  
    error: errorHandler  
});
```

```
function successHandler(data)  
{  
    alert("通信成功");  
}  
  
function errorHandler(XMLHttpRequest, textStatus,  
    errorThrown)  
{  
    alert("通信失敗");  
}
```

- 10秒経ってもデータが来ない場合、エラーになる
 - 成功したときは、successHandler関数が呼ばれる
 - 失敗したときは、errorHandler関数が呼ばれる

PHPからJSONを吐き出す

- 結構単純
- PHP上で吐き出したいデータを変数などに格納しておく（配列に格納しておく）
 - 配列に格納するデータは連想配列の形にしておく
 - 連想配列：`$data["name"] = "Kazuma" ;`
- JSONを吐き出す前にJSONを出力する旨のヘッダーを出力する
 - `header('Content-type: application/json; charset=utf-8');`
- JSONのデータをエンコードして吐き出す
 - `echo json_encode($data);`