React実習

08. SpringとReactの連携

株式会社ジードライブ

今回学ぶこと

- Spring BootでのWeb APIの作成する方法
  - 作成したAPIは、Reactで利用することができる

Web APIとREST API

Web API

- Webの仕組みを使い(= インターネット経由で)、アプリサーバー上のプログラムを呼び出すための窓口

REST API / RESTful API

- RESTという設計モデルに沿って作られたWeb API
  - URLがリソースを表し、リソースに対する操作はHTTPメソッドで決定する

HTTPメソッド	リソースに対する操作
GET	リソースの取得
POST	リソースの追加
PUT	リソースの更新
DELETE	リソースの削除

Spring BootでのWeb API作成

- @Controllerの代わりに、@RestControllerを使用する
  - コントローラー用メソッドの戻り値がビューの名前ではなくResponse Bodyとして扱われる
  - これにより、戻り値のオブジェクトがJSONデータに変換されてクライアントに送信されるようになる

- @GetMapping, @PostMappingに加え、@PutMapping, @DeleteMappingアノテーションを使用する
  - これにより、各種HTTPメソッドに対応することができる

アノテーション

- これまでに学習したアノテーションに加え、以下のようなアノテーションを使用する

アノテーション	説明
@PutMapping	PUTリクエストに対応するメソッドに付与する
@DeleteMapping	DELETEリクエストに対応するメソッドに付与する
@ResponseBody	コントローラー内のメソッドに付与することで、ビュー(Thymeleaf)へフォワードせず、直接的にコンテンツ(HTMLやJSON)を返すことができるようになる
@RequestBody	コントローラー内メソッドの引数に付与することで、POSTメソッド等で送信されるJSONデータを受け取ることができるようになる
@RestController	@Controllerの代わりに、コントローラークラスに付与することができるアノテーション。クラス内の一つ一つのメソッドに、@ResponseBodyが付与された状態になる
@CrossOrigin	CORS(Cross-Origin Resource Sharing)の設定ができる

エンドポイントの作成

- REST APIでは、エンドポイント(URL)は、リソースを表す名詞にする

リソースが会員情報の場合の例

エンドポイント	HTTPメソッド	提供される情報や機能
/api/members	GET	会員のリスト
/api/members/{id}	GET	特定の会員情報
/api/members	POST	会員の追加
/api/members	PUT	会員情報の更新
/api/members/{id}	DELETE	会員情報の削除

会員情報の更新や削除を@PostMapping("/api/updateMember") や @GetMapping("/api/deleteMember/{id}")のようなURLによって対応するのではなく、@PutMapping, @DeleteMappingを使い、HTTPメソッドで対応する

コントローラーの例

Java

```java
@RestController
@RequestMapping("/api/members")
public class MemberController {

    @Autowired
    MemberMapper mapper;

    // 会員リストをJSONで返す
    @GetMapping
    public List<Member> getAllMembers() {
          return mapper.selectAll();
    }

    // JSONデータを受け取り、会員情報を追加する
    @PostMapping
    public void addMember(@RequestBody Member member) {
          mapper.insert(member);
    }
```

返すデータの例

```
[
  {
     "id" : 1,
     "name": "山田太郎",
     "aga": 25
   },
   {
     "id": 2,
     "name": "田中花子",
     "aga": 24
   }
]
```

JSONからMember型へ変換される
JSONデータのもつプロパティとMemberクラスのフィールドがマッチする必要がある

コントローラーの例

Java

```
    … 前頁の続き …

    // JSONデータを受け取り、会員情報を更新する
    @PutMapping
    public void updateMember(@RequestBody Member member) {
          mapper.update(member);
    }

    // 会員IDを元に、会員情報を削除する
    @DeleteMapping("/{id}")
    public void deleteMember(@PathVariable Integer id) {
          mapper.delete(id);
    }

}
```

コントローラーの例

- @RequestBodyに、@Validアノテーションを併用し、バリデーションを行うことも可能

Java

```
@PostMapping
public String addMember(
    @RequestBody @Valid Member member,
    Errors errors) {
     if(errors.hasErrors()) {
        return "入力に不備あり";
     }

     mapper.insert(member);
     return "会員を追加しました";
}
```

日時のフォーマット

- JSON ⇔ ドメインクラスの変換で、日時を扱う際は、@JsonFormatアノテーションで形式を指定することができる

Java

```java
@Data
public class Member {

   private Integer id;

   private String name;

   @JsonFormat(pattern = "yyyy-MM-dd")
   private LocalDate birthday;

}
```

ResponseEntity

- ResponseEntityクラスを利用することで、レスポンスにヘッダ情報やステータスコードを含めることができる
  - ヘッダには、認証・認可に関する情報などを含める

書式：ステータスコードを含むコンストラクタ

```
ResponseEntity(T body, HttpStatusCode status)
```

書籍情報や会員リストなどAPIを通じて提供するリソース

書式：ヘッダ情報とステータスコードを含むコンストラクタ

```
ResponseEntity(T body, MultiValueMap<String, String> headers,
               HttpStatusCode status)
```

ResponseEntityの例

- 書籍IDを受け取り、該当する書籍データを返すメソッド
  - 戻り値をResponseEntity型にすることによって、ステータスコードを含むデータをクライアントに返すことができる

提供するリソースの型

Java

```
@GetMapping("/api/books/{id}")
public ResponseEntity<Book> getBook(@PathVariable String id) {
    Book book = service.getBookById(id);
    if(book != null) {
        return new ResponseEntity<>(book, HttpStatus.OK);
    } else {
        return new ResponseEntity<>(null, HttpStatus.NOT_FOUND);
    }
}
```

提供するリソースとステータスコード

ステータスコード

- ステータスコードは、列挙型クラスのHttpStatusを使い指定することができる

列挙定数	対応するコード番号	利用場面の例
OK	**200**	**問題なし**
BAD_REQUEST	400	リクエストパラメーターに不備がある
UNAUTHORIZED	401	リソースへのアクセスを許可しない
NOT_FOUND	404	リソースが見つからない
CONFLICT	409	データベースに、同一IDのデータが存在しており、データを追加できない

その他一覧：
https://spring.pleiades.io/spring-framework/docs/current/javadoc-api/org/springframework/http/HttpStatus.html

練習

- 練習08-1

CORSの設定

- セキュリティ上の理由から、Webブラウザは「**同一オリジンポリシー**」を実装している
  - 同一オリジンポリシー：あるオリジンから送信されたリソース(JavaScript等)は、その送信元オリジンとは異なるリソースにアクセスできないというルール
- CORS(Cross-Origin Resource Sharing)は、異なるオリジンのリソースにアクセスできるようにする仕組み
  - オリジン：プロトコル(http等)、ドメイン(example.com等)、ポート番号(80等)のセット

http://localhost:3000 で動いているReactのアプリから、
http://localhost:8080 で動いているSpringが提供するリソースにアクセスできるように、CORSを設定する必要がある
⇒ リソースを提供する側(Spring Boot)で設定をする

レスポンスヘッダ

- 異なるオリジンに対してリクエストを行う前に、ブラウザは自動的にプリフライト・リクエストを送り、実際にリクエストを送信するか判断する
  - この事前リクエストに対するレスポンスヘッダには、以下のような情報が含まれている（この情報がCORS設定となる）

ヘッダ項目	説明
Access-Control-Allow-Origin	リクエストを許可するオリジンの設定
Access-Control-Allow-Credentials	Cookie等の認証情報を含むリクエストの許可設定
Access-Control-Allow-Methods	リソースにアクセスする際に使用可能なHTTPメソッド
Access-Control-Allow-Headers	リクエストを行う際に使用可能なHTTPヘッダ
Access-Control-Expose-Headers	レスポンスの一部として公開するヘッダ
Access-Control-Max-Age	プリフライト・リクエストの結果(許可されるメソッドやヘッダの情報)をキャッシュできる時間の設定

CORSの有効化（方法1）

- コントローラークラスに@CrossOriginアノテーションを付けてCORSを設定する

例：クラス全体に設定する場合

Java
```
@RestController
@CrossOrigin(origins = "http://localhost:3000", allowCredentials="true")
public class MemberController {
  ...
```

例：メソッドごとに設定する場合

Java
```
@RestController
public class MemberController {

  @CrossOrigin(origins = "http://localhost:3000", allowCredentials="true")
  @GetMapping("/api/members)
  public List<Member> members() {
    ...
```

@CrossOriginのオプション

- Webクライアントから Cookie などの認証情報を送信できるようにするには allowCredentials を true に設定する必要がある
  - また、この場合 origins は明示する必要がある

プロパティ	対応するヘッダ項目	デフォルト値
origins	**Access-Control-Allow-Origin**	全て許可
allowCredentials	**Access-Control-Allow-Credentials**	false
methods	Access-Control-Allow-Methods	コントローラーメソッドがマップされているメソッド
allowedHeaders	Access-Control-Allow-Headers	すべてのヘッダ
exposedHeaders	Access-Control-Expose-Headers	なし
maxAge	Access-Control-Max-Age	1800秒(30分)

CORSの有効化 (方法2)

- アプリケーションの設定でaddCorsMappings()メソッドをオーバーライドする

設定ファイルの記述例

Java

```java
@Configuration
public class ApplicationConfig implements WebMvcConfigurer {

    @Override
    public void addCorsMappings(CorsRegistry registry) {
        registry.addMapping("/**")
            .allowedOrigins("http://localhost:3000")
            .allowedMethods("GET", "POST", "PUT", "DELETE")
            .allowedHeaders("Authorization", "Content-Type")
            .allowCredentials(true);
    }

}
```

addCorsMappings内の設定

- まず、CorsRegistry#addMapping()メソッドで、設定の対象となるパスを指定し、以下のメソッドを繋げることでCORS設定を行う

メソッド	設定内容
allowedOrigins(String... origins)	リクエストを許可するオリジンの設定
allowCredentials(boolean allowed)	Cookie等の認証情報を含むリクエストの許可設定
allowedMethods(String... methods)	リソースにアクセスする際に使用可能なHTTPメソッド
allowedHeaders(String... headers)	リクエストを行う際に使用可能なHTTPヘッダ
exposedHeaders(String... headers)	レスポンスの一部として公開するヘッダ
maxAge(long maxAge)	プリフライト・リクエストの結果(許可されるメソッドやヘッダの情報)をキャッシュできる時間の設定

練習

- 練習08-2

- 練習08-3
- 練習08-4
- 練習08-5
- 練習08-6
- 練習08-7

React