Ajax

Ajaxを利用するアプリケーションの実装方法について説明する。
以下、Ajaxリクエストの各送信方法について、ControllerとJavaScriptの実装例を示す。

実装方針

- ajaxメソッドは$.ajaxではなくAP基盤提供のatc.ajaxを使用すること
  - エラーハンドリング(failコールバック)などの共通化を行っている
  - atc.ajaxは$.ajaxに対する薄いラッパーであり、使い方は$.ajaxと同じ (但し、統制のため $.ajax({url: '....', data: '...'}) 形式の呼び出し方は未サポート)
- エラーハンドリングは特別な要件が無い限り、個別に行わないこと
  - 各画面用JSではエラーメッセージ表示要素などを宣言的に指定するだけで、failコールバックは実装しない

GETリクエスト

JS実装例 (エラーハンドリングは省略)

```
$(function() {
  'use stri';

  var contextPath $('meta[name="contextPath"]').attr('content'); // <1>

    atc.ajax(contextPath'/foo/DBAP0010/search', // <2>
    type: 'GET',
    data: // <3>
      q1: '...',
      q2: '...'
        }
    }).dofunction(data) // <4>

  });

});
```

- <1> ベースのJSPで出力しているcontextPath用のmetaタグからcontextPathを取得する
- <2> Ajaxリクエストの送信にはatc.ajaxメソッドを使用する
- <3> クエリパラメータの指定 (このオブジェクトからクエリ文字列が生成される)
- <4>
  正常系のレスポンスはdoneハンドラで受け取る。第一引数のdataはJSON文字列をパースした結果であるJSオブジェクトになる。

Controller実装例

```
@Controller
@RequestMapping("foo/DBAP0010")
public class DBAP0010PController {

    @GetMapping("search")
    @ResponseBody // <1>
    public AjaxSampleResponse search(AjaxSampleForm form) {

        final AjaxSampleResponse resnew AjaxSampleResponse();
        // ...
        return res;
      }

}
```

- <1> JSON文字列を返すために@ResponseBodyアノテーションを付ける

POSTリクエスト(フォームデータ)

JS実装例 (エラーハンドリングは省略)

```
$(function() {
  'use stri';

  var contextPath $('meta[name="contextPath"]').attr('content');

    atc.ajax(contextPath'/foo/DBAP0010/register', {
    type: 'POST',
    data: $('#foo-form').serialize() // <1>
    }).dofunction(data) {

    });

});
```

- <1> formデータをapplication/x-www-form-urlencodedで送信する場合はserializeメソッドを使用する
    - 送信するデータを加工する場合は代わりにserializeArrayメソッドを使用する

Controller実装例

```
@Controller
@RequestMapping("foo/DBAP0010")
public class DBAP0010PController {

    @PostMapping("register")
    @ResponseBody
    public AjaxSampleResponse regist@Validated AjaxSampleForm form) {

        final AjaxSampleResponse resnew AjaxSampleResponse();
        // ...
        return res;
          }

}
```

POSTリクエスト(JSON)

JS実装例 (エラーハンドリングは省略)

```
$(function() {
  'use stri';

  var contextPath $('meta[name="contextPath"]').attr('content');

    atc.ajax(contextPath'/foo/DBAP0011/register', {
    type: 'POST',
    contentType: 'application/json; charset=UTF-', // <1>
    data: JSON.stringify(// <2>
      d1: '...',
      d2: '...'
        })
    }).dofunction(data) {

    });

});
```

- <1> JSON文字列を送信するので、contentTypeをapplication/jsonにして送信する

- <2> JSON.stringifyによりJSON文字列を生成する

Controller実装例

```
@Controller
@RequestMapping("foo/DBAP0011")
public class DBAP0011PController {

    @PostMapping("register")
    @ResponseBody
    public AjaxSampleResponse regist@Validated @RequestBody AjaxSampleForm form)// <1>

        final AjaxSampleResponse resnew AjaxSampleResponse();
        // ...
        return res;
          }

}
```

- <1> リクエストボディに含まれたJSONを取得するにはマッピングする引数に@RequestBodyアノテーションを付ける

エラーハンドリング

概要

- エラー発生時の挙動観点でのエラー分類
  - 画面上部に表示するエラー(バリデーションエラー、RecoverableBusinessException)
  - 共通エラー画面へ遷移して表示するエラー (システムエラー、UnRecoverableBusinessException)
- サーバ側(Controller)での留意事項
  - 入力チェックの結果をControllerでハンドリングせず、FWに任せる
    (つまり、Controllerの引数にBindingResultを定義しない)
  - RecoverableBusinessExceptionもControllerでcatchせず、FWに任せる
- クライアント側(JS)での留意事項
  - 画面上部に表示するエラーが発生するAjaxリクエストを送信する箇所では、以下に示す設定を必ず追加する必要がある
  - 共通エラー画面へ遷移して表示するエラーに関しては個別画面のJSで意識する必要は無い

JSP実装例

```
<div id="error-message-area" class="asw-notice-massage" style="display: none"></div> <!--1-->

<form id="foo-form">
    <!-- ... -->
</form>
```

- <1> エラー表示エリアを用意しておき、style="display: none"で非表示にしておく

JS実装例

```
$(function() {
  'use stri';

  var contextPath $('meta[name="contextPath"]').attr('content');
  var errorMessageTemplate = Handlebars.com'{{#each this}}<p>{{{this}}}</p>{{/each');

  var $form =$('#foo-form');

  $form.on('submit', function() {

          atc.ajax(contextP/'/foo/DBAP0010/register', {
      type: 'POST',
      data: $form.serialize(),
      context: {
        errorMessageArea: '#error-message-area', // <1>
        errorMessageTemplate: errorMessageTemplate,
```

```
//  <2>  ※  このオプションはエラーメッセージ表示エリアのDOM構造が標準と異なる場合のみ指定すること  ※
        form: $form //  <3>
                }
            }).function(data)  {

            });
    //  <4>

    return false;
     });

});
```

- <1> エラーメッセージ表示エリアを指定する
  - 画面上部に表示するエラーが発生するAjaxリクエストの場合は指定必須
  - 指定方法は jQuery or Element or Selector
    - 上記の例ではSelectorで指定している
- <2> エラーメッセージ表示エリアのコンテンツ(=エラーメッセージ)用のJSテンプレート関数を指定する
  - 基本的に指定しない (オプションパラメータ)
    - デフォルト値は以下の通り
    - atd-csm-web: atd.errorMessageDefaults.standardErrorMessageTemplate ({{#each this}}
{{{this}}}
{{/each}})
    - ati-csm-web: ati.errorMessageDefaults.standardErrorMessageTemplate ({{#each this}}
{{{this}}}
{{/each}})
  - エラーメッセージ表示エリアのDOM構造が異なる場合のみ指定すること
- <3> リクエストパラメータに渡すデータ取得元となるform要素を指定する
  - バリデーションエラーメッセージの表示順をformのフィールドの出現順でソートしたい場合に指定する
  - バリデーションエラーが発生するAjaxリクエストの場合は基本的に指定することになる
    (指定しない場合、表示順はランダムになる)
  - また、二重送信防止対応を適用するformに対するAjaxリクエストの場合は指定必須 (ref. 二重送信防止)
  - 指定方法は jQuery or Element or Selector
    - 上記の例ではjQueryオブジェクトで指定している
- <4> failコールバックは実装しない (共通で差し込まれているfailコールバックに任せる)

Controller実装例

```
@Controller
@RequestMapping("foo/DBAP0010")
public class DBAP0010PController  {

    @Autowired
    private  SampleService  sampleService;

    @PostMapping("register")
    @ResponseBody
    public  AjaxSampleResponse  regist@Validated  AjaxSampleForm  form)//  <1>

                    SampleDto  result  =  sampleService.regis//  <2>ple(....);

        final  AjaxSampleResponse  resnew  AjaxSampleResponse();
        //  ...
        return  res;
          }

}
```

- <1> Controllerの引数にBindingResultを定義しない
  - そうすることでSpring
    MVCが例外を送出することになり、FWはその例外をハンドリングして共通のJSONレスポンスをクライアントに返す
  - 入力データがFormデータではなくJSONの場合(@RequestBodyを付けるケース)でも同様
- <2> ServiceがRecoverableBusinessExceptionを送出する場合でもControllerでcatchしない
  - UnRecoverableBusinessException及びシステムエラーもControllerでcatchしない (これは非Ajaxの場合と同じ)

バリデーションエラーが発生した場合の動作をカスタマイズする方法

以下の様に、atc.ajaxのcontext.appValidationErrorCallbackオプションを使うことで対応可能。
Note: この機能は atd-csm-web 及び ati-csm-web
アプリでのみ使用可能です。他のアプリで使用したい場合はAP基盤までご連絡を。

```
var $form = $('#sample-form');

$form.on('submit', function() {

    atc.ajax(contextPath + '/reservation/sample/authenticate', {
  type: 'POST',
  data: $form.serialize(),
  context: {
    errorMessageArea: '#sample-error-area',
    form: $form,
    appValidationErrorCallback: function(data) {
      // エラーが発生した入力要素に error クラスを付与する
      $.each(data.errors, function(index, error) {
        $form.find('[name="' + error.target + '"]').addClass('error');
                });
            }
        }
    }).done(function(data) {
// ...
    });

  return false;
});
```

- <1> appValidationErrorCallbackを利用することでバリデーションエラー発生時の画面固有動作を実装する事が出来る
  - 引数に渡されるdataの構造はデータ構造は次の通り

```
{
  "errors": [
        {
      "message": "バリデーションエラーメッセージ",
      "target": "バリデーションエラーが発生したフィールド名orオブジェクト名 (e.g. users[0].user" + ")
        },
    // ..省略..
    ]
}
```

  - appValidationErrorCallbackでfalseを返すことで、標準のバリデーションエラー処理をキャンセルすることが可能

参考

- jQuery.ajax() | jQuery API Documentation
- 4.13. Ajax — TERASOLUNA Server Framework for Java (5.x) Development Guideline 5.3.0.RELEASE documentation