来源：

　https://qiita.com/TEBASAKI/items/267c261db17f178e33eb

本家：

　https://spring.pleiades.io/spring-boot/docs/current/reference/html/getting-started.html#getting-started.first-application

**Spring Boot解説第18回(基本編：Controllerとは)**

今回からいよいよ基本編に入ります。  
初回ということで、今更ですが Controller について解説していきます。

**MVCとは**

Controller の前に、まずは MVC について解説します。  
MVC とはアプリケーションフレームワークのひとつです。  
これは**Model(モデル)**、**View(ビュー)**、**Controller(コントローラ)**の頭文字を取ったもので、全体の処理をこの3つに分割します。

続いて、それぞれの役割について解説していきます。

**Model(モデル)**

Model は処理の中核を担います。  
アプリケーションが使用するデータや、DBに対する処理などは Model が担当することになります。

**View(ビュー)**

View はクライアントからの入力やクライアントへの出力を担います。  
第15回から第17回まで解説した Thymeleaf などは View が担当することになります。

**Controller(コントローラ)**

Controller は上記2つの橋渡しを担います。  
クライアントからの入力を View から受け取り、それをもとに Model に指示を伝えます。  
その後、Model から受け取った値を View に伝え、見た目に反映させます。

以上のように、MVC ではプログラム中の処理を分担し管理します。

**Spring Boot と MVC の関係**

上記の MVC を基に Spring MVC というフレームワークが作られました。  
しかし、 Spring MVC はライブラリが膨大であることや、設定が複雑であることなどから使いこなすのが難しいという問題がありました。  
そこで、この Spring MVC をより簡単に使えるように作られたフレームワークが Spring Boot です。

**Controller の記述方法**

Spring Boot において Controller はどのように記述するのか、解説していきます。

**Controller クラス**

クラスに対して**@Controller**あるいは**@RestController**というアノテーションを記述することで、Spring Boot はそのクラスを Controller として認識します。

Controller

@Controller

public class Controller {

...

}

RestController

@RestController

public class RestController {

...

}

@RestControllerでは View に遷移せず、メソッドの戻り値がそのままレスポンスのコンテンツになります。  
@Controllerでは後述する@ResponseBodyというアノテーションを使用することで、同様の処理ができますが、@RestControllerは@ResponseBodyを記述する必要はありません。  
通常は@Controllerを使いますが、JSON や XML などを返す WebAPI 用の Controller などには@RestControllerを使います。

**@RequestMapping**

@RequestMappingというアノテーションを記述することで、クライアントからのリクエストに対してメソッドやハンドラをマッピングします。  
一般的には特定のURLリクエストに対してマッピングを行いますが、 後述する属性を使えばGET や POST といったメソッドなどで条件付けすることもできます。

RequestMapping

@RequestMapping("/test")

public string test() {

return "test";

}

@RequestMappingはクラスとメソッドどちらにも使用できます。  
クラスに対して使用した場合、設定された値は親パスとして機能します。  
その場合、メソッドはデフォルトで親パスへのリクエストにマッピングされます。  
メソッドに使用した場合は親パスからの相対パスと、絶対パスの両方が設定できます。

**属性について**

@RequestMappingは以下の属性を使用することで、マッピングの条件を指定することができます。

RequestMapping

@RequestMapping(value="/test", method=RequestMethod.GET, headers="Accept=application/\*", params="id=002")

・**value**  
URL のパスを記述します。  
この URL にマッチする場合、メソッドが呼ばれます。  
value のみを指定する場合は「value=」の部分を省略して URL のみを記述することも可能です。  
ワイルドカード (「\*」 や 「\*\*」) も使用できます。  
上記の例では {サーバー}/{コンテキストパス}/test にアクセスした場合にマッチします。

・**method**  
GET や POST などのメソッドを指定できます。  
URLに加えて、メソッドによって表示する内容を変更する場合に使用します。  
上記の例ではGETによるアクセス時にマッチします。

・**headers**  
HTTP のヘッダを指定できます。  
ヘッダ名のみ指定されている場合は、そのヘッダが存在するかどうかでマッチの判定をします。  
ヘッダ名と値が指定されている場合は、ヘッダにその値が設定されているかどうかでマッチの判定をします。  
上記の例では Accept ヘッダに「application/\*」にマッチする値が設定されている場合にマッチします。

・**params**  
リクエストパラメーターを指定できます。  
headers と同じ記述方法で指定します。  
上記の例では id に「002」が設定されている場合にマッチします。

**戻り値**

Controller は様々なクラスを戻り値として返すことができます。  
それぞれについて解説していきます。

**String**

View 名を返します。  
この View 名は ViewResolver によって実際のファイル名に変換され、クライアントに返却されます。  
また、「forward:」「redirect:」といったプレフィックスを使用すると、それぞれフォワード処理、リダイレクト処理ができます。

戻り値：String

@RequestMapping("/test")

public string test() {

return "test";

}

**void**

値を返しません。  
メソッド内で直接クライアントに処理を返す場合などに使用します。

戻り値：void

@RequestMapping("/test")

public void test(@RequestBody String body, Writer writer) throws IOException {

writer.write(body);

}

上記の例の@RequestBodyについては後述しますが、リクエストのボディを受け取ることができます。

**@ResponseBody**

クラスではありませんが、@ResponseBodyというアノテーションをメソッドに記述すると、戻り値がそのままレスポンスのボディの内容になります。  
[第2回](http://qiita.com/TEBASAKI/items/a30a7daf11cdb51272fb)で作成した Hello World でも使用していました。

@ResponseBody

@RequestMapping("/test")

@ResponseBody

public string test() {

return "Hello World";

}

上記の例ではただ文字列を設定しているだけですが、@ResponseBodyは戻り値について細かく設定できます。  
この設定方法などの詳細については次回に説明します。

**ModelAndView**

Model と View 名を持つクラスです。  
View に対して渡す Model も同時に返すことができます。

戻り値：ModelAndView

@RequestMapping("/test")

public ModelAndView test(ModelAndView mav) {

mav.addObject("model", "model"));

mav.setViewName("test");

return mav;

}

以降本ブログでは、Controller の戻り値は **ModelAndView** を使用します。  
これはメソッドの戻り値を統一することで、ソースコードをわかりやすくするためです。

**Controller で使用できるアノテーション**

**メソッドに対するアノテーション**

先述した@RequestMappingや@ResponseBodyのほかにも、メソッドに対して使用できるアノテーションが存在します。

**@ModelAttribute**

自作したモデルクラスを引数として受け取る場合に使用します。  
@ModelAttributeはリクエストのたびに@RequestMappingの前に呼ばれ、対応するモデルとオブジェクトがバインドされます。  
このアノテーションについての詳細は次回に説明します。

**引数に対するアノテーション**

メソッドは View から様々な値を受け取ることができます。  
引数に対してアノテーションを記述することで受け取れます。  
使用できるアノテーションは以下の通りです。

**@PathVariable**

リクエストパラメーターを受け取ります。  
下記のようにURL にユーザーIDが含まれていて、その値を受け取りたい場合などに使用します。

@PathVariable

@RequestMapping("/test/{userId}")

public ModelAndView test(@PathVariable("userId") String userId, ModelAndView mav) {

...

}

**@RequestHeader**

リクエストヘッダを受け取ります。  
required 属性は必須か否かを指定します。  
required はデフォルトで true となっているため、リクエストヘッダ が存在しない場合にはエラーとなってしまいます。  
それを回避するには required 属性に false を設定する必要があります。

@RequestHeader

@RequestMapping("/test")

public ModelAndView test(@RequestHeader(value="User-Agent", required=false) String agent, ModelAndView mav) {

...

}

**@CookieValue**

クッキーの値を受け取ります。  
下記の例では、@RequestHeader同様に required 属性に false を設定しています。

@CookieValue

@RequestMapping("/test")

public ModelAndView test(@CookieValue(value="SESSIONID", required=false) String sid, ModelAndView mav) {

...

}

**@RequestBody**

リクエストボディの内容を受け取ります。

@RequestBody

@RequestMapping("/test")

public ModelAndView test(@RequestBody String body, ModelAndView mav) {

...

}

今回はここまでです。  
次回は引き続き、Controller について解説していきます。  
それではみなさん、またお会いしましょう！

目標：

1. アノテーションを利用しマッピングの理解できる
2. 画面側からのリクエストを受ける
3. データを画面へ戻す事ができる