컨트롤러

https://www.baeldung.com/

https://docs.spring.io/spring-framework/docs/3.2.x/spring-framework-reference/html/mvc.html#mvc-localeresolver

컨트롤러

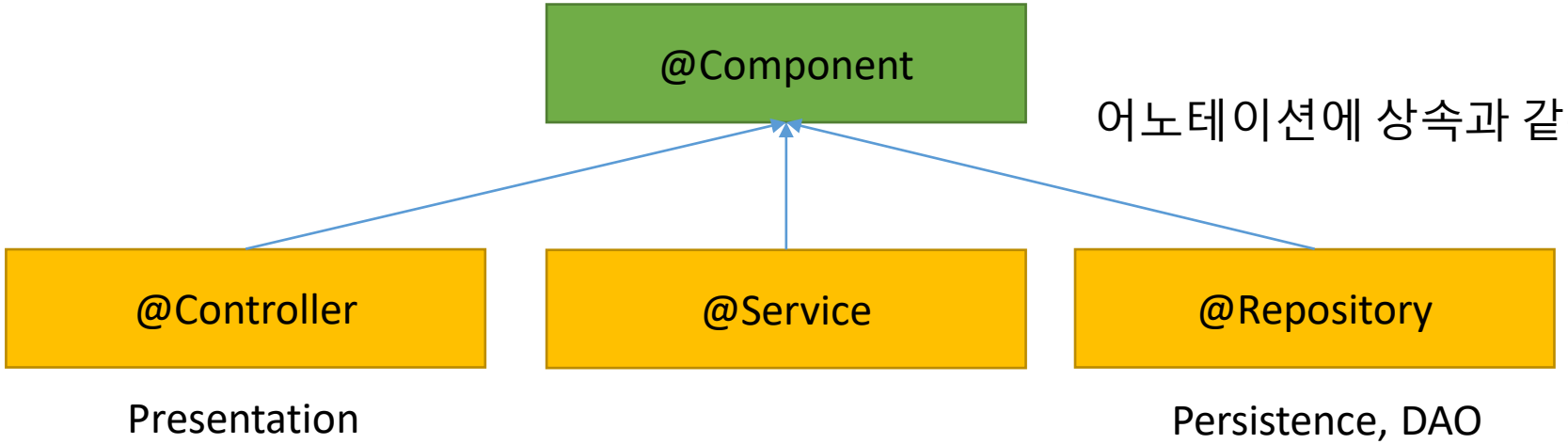
- **소개**
  - 사용자 요청(input)을 적절히 처리하여 그 결과를 Model(ModelAndView)에 담는 역할
  - 컨트롤러를 정의하는 다양한 방법이 존재
  - Spring 2.5에서 어노테이션 기반 컨트롤러 등장
    - @RequestMapping, @RequestParam, @ModelAttribute, and so on
  - 어노테이션 기반 컨트롤러의 경우 부모 클래스를 상속받거나 인터페이스를 구현할 필요 없이 단순히 하나의 클래스를 정의하면 됨

```
@Controller
public class HelloWorldController {

    @RequestMapping("/helloWorld")
    public String helloWorld(Model model) {
        model.addAttribute("message", "Hello World!");
        return "helloWorld";
    }
}
```

정의

- **@Controller**
  - 컴포넌트 스캔 대상이 되는 빈 중, Controller의 역할을 담당한다는 것을 명시



어노테이션에 상속과 같은 개념이 있는 것은 아님

  - @Component : 컴포넌트 스캔에서 사용
  - @Controller : 스프링 MVC 컨트롤러
  - @Repository : 스프링 데이터 접근 계층
  - @Service : 스프링 비즈니스 로직(특별한 기능이 있는 것은 아니고 빈의 역할을 분명하게 나타내기 위해)

예시

```
@Controller
@RequestMapping("/appointments") //클래스 레벨
public class AppointmentsController {

  private final AppointmentBook appointmentBook;

  @Autowired
  public AppointmentsController(AppointmentBook appointmentBook) {
    this.appointmentBook = appointmentBook;
  }

  @RequestMapping(method = RequestMethod.GET) //get 매핑, value가 없음: localhost:8080/appointments
  public Map<String, Appointment> get() {
    return appointmentBook.getAppointmentsForToday();
  }
  …

}
```

예시

```
@Controller
@RequestMapping("/appointments")
public class AppointmentsController {
    //get 매핑 : localhost:8080/appointments/13
   @RequestMapping(value="/{day}", method = RequestMethod.GET)
   public Map<String, Appointment> getForDay(@PathVariable @DateTimeFormat(iso=ISO.DATE) Date day, Model model) {
      return appointmentBook.getAppointmentsForDay(day);
   }
    //get 매핑 : localhost:8080/appointments/new
   @RequestMapping(value="/new", method = RequestMethod.GET)
   public AppointmentForm getNewForm() {
      return new AppointmentForm();
   }

   @RequestMapping(method = RequestMethod.POST) //Post 매핑
   public String add(@Valid AppointmentForm appointment, BindingResult result) {
      if (result.hasErrors()) {
         return "appointments/new";
      }
      appointmentBook.addAppointment(appointment);
      return "redirect:/appointments";
   }
}
```

RequestMapping

- **@RequestMapping 대체: HTTP 종류에 따른 구체적인 명시**
  - GetMapping
  - PostMapping
  - PutMapping
  - DeleteMapping
  - PatchMapping

- **하나의 RequestMapping으로 다수의 URL을 매핑**
  - 사용자 입력의 편이성 제공

```
@RequestMapping({"/welcome","/welcome2"})
```

Path Variable

- **@PathVariable**
  - "localhost:8080/appointments/13"와 같이 요청 URL에 13이라는 자원을 지정할 수 있음
  - REST API가 점점 중요해짐 → PathVariable 사용 빈도 증가

```
@RequestMapping(value="/owners/{ownerId}/pets/{petId}", method=RequestMethod.GET)
public String findPet(@PathVariable String ownerId, @PathVariable String petId, Model model) {
  Owner owner = ownerService.findOwner(ownerId);
  Pet pet = owner.getPet(petId);
  model.addAttribute("pet", pet);
  return "displayPet";
}
```

  - PathVariable이 다수일 경우, Map<String, String>으로 한 번에 받을 수 있음

```
@Controller
@RequestMapping("/owners/{ownerId}")
public class RelativePathUriTemplateController {

  @RequestMapping("/pets/{petId}")  // /owners/42/pets/21
  public void findPet(@PathVariable String ownerId, @PathVariable String petId, Model model) {
    // implementation omitted
  }
}
```

Supported method argument types

- **다양한 메소드 argument 사용**
  - 컨트롤러를 정의할 때, handler 함수 정의에 다양한 파라미터를 사용할 수 있음

```
@RequestMapping(value="/{day}", method = RequestMethod.GET)
  public Map<String, Appointment> getForDay(@PathVariable @DateTimeFormat(iso=ISO.DATE) Date day, Model model) {
    return appointmentBook.getAppointmentsForDay(day);
  }
```

  - HandlerApater는 handler 실행에 필요한 인자들을 미리 만듦
    - @PathVariable: Path 변수를 받을 때
    - @RequestParam: 요청 파라미터를 받을 때
    - @RequestBody: 대표적으로 요청 Body에 포함된 JSON과 같은 데이터를 조회할 때 사용
    - @ModelAttribute: 정의한 (getter/setter가 정의된)DTO에 자동 데이터 바인딩
    - @Valid: 입력의 유효성 검사

Supported method argument types

- **요청 파라미터와 HTTP 메시지 바디 구분**
  - 요청 파라미터의 종류
    - 쿼리스트링
    - Get
    - Post
  - 요청 파라미터 조회 방법
    - @RequestParam, @ModelAttribute
  - HTTP 메시지 바디를 조회하는 방법
    - @RequestBody

Supported method argument types

- **Binding request parameters to method parameters with @RequestParam**

```
@RequestMapping(method = RequestMethod.GET)
  public String setupForm(@RequestParam("petId") int petId, ModelMap model) {
    Pet pet = this.clinic.loadPet(petId);
    model.addAttribute("pet", pet);
    return "petForm";
  }
```

- **Mapping the request body with the @RequestBody annotation**

```
@RequestMapping(value = "/something", method = RequestMethod.PUT)
public void handle(@RequestBody String body, Writer writer) throws IOException {
  writer.write(body);
}
```

Using @ModelAttribute on a method

- **@ModelAttribute**
  - 메소드(Method Level)와 메소드의 인자(Method Parameter Level)에 사용 가능
    - 메소드 레벨 → 해당 컨트롤러 내의 모든 @RequestMapping 메소드의 Model에 삽입되어 반환
    - @RequestMapping가 붙은 핸들러의 실행 전에 @ModelAttribute가 붙은 메소드가 실행됨
    - (일반적으로 데이터베이스에서 조회한) 모델에 공통적으로 들어가야 할 정보를 미리 채움(populate)

```
@Controller
public class MyController {
  @ModelAttribute("productsList")
  public Collection<Product> populateProducts() {
    return this.productsService.getProducts();
  }
}
```

속성 하나 추가

```
@ModelAttribute
public void populateModel(@RequestParam
String number, Model model) {

model.addAttribute(accountManager.findAccount(
number));
  // add more ...
}
```

여러 개의 속성 추가

https://stackoverflow.com/questions/25011568/spring-mvc-modelattribute-method

Supported method return types

- **메소드는 다양한 리턴 타입을 사용할 수 있음**
  - ModelAndView object: 모델과 뷰 전달
  - Model object : 모델 전달
  - Map object : 모델과 뷰 이름 전달
  - View object : 뷰 이름 전달
  - String : @Controller → view의 논리 이름, @RestController → 데이터
  - void : 현재 호출한 url의 경로로 뷰를 찾음
    - (요청: /person/new, 실행: templates/person/new.html)

Supported method argument types

- **Mapping the response body with the @ResponseBody annotation**

```
@RequestMapping(value = "/something", method = RequestMethod.PUT)
@ResponseBody
public String helloWorld()  {
   return "Hello World";
}
```

- **Using HttpEntity<?>**
  - @RequestBody와 @ResponseBody의 기능과 더불어 헤더 정보까지 조회 가능

```
@RequestMapping("/something")
public ResponseEntity<String> handle(HttpEntity<byte[]> requestEntity) throws UnsupportedEncodingException {
   String requestHeader = requestEntity.getHeaders().getFirst("MyRequestHeader"));
   byte[] requestBody = requestEntity.getBody();
   // do something with request header and body

   HttpHeaders responseHeaders = new HttpHeaders();
   responseHeaders.set("MyResponseHeader", "MyValue");
   return new ResponseEntity<String>("Hello World", responseHeaders, HttpStatus.CREATED);
}
```