Mapping Request

`@RequestMapping`

- **You can use the `@RequestMapping` annotation to declare a class to handle request. Use HTTP method shortcut to handle specific HTTP methods. These variants include `@GetMapping`, `@PostMapping`, `@PutMapping`, `@DeleteMapping`, and `@PatchMapping`.Multiple of the same request annotations to different url end points in a** *SINGLE *element is not allowed. The following example demonstrates this with type and method level mapping*

```
@RestController
@RequestMapping("/persons")
class PersonConctroller {

    @GetMapping("/{id}")
    public Person getPersonById(@PathVariable Long id) {
        // ...
    }

    @PostMapping
    @ResponseStatus(HTTP.CREATED)
    public void create(@RequestBody Person person) {
        // ...
    }
}
```

`@HttpExchange`

- the main purpose of this annotation is to abstract HTTP client code with a generated proxy. Additionally, HTTP interfaces are convenient way to expose a projects api to their client side
- Not recommended for external api's but is a great choice for an internal api
- `@HttpExchange` replaces `@RequestMapping` by declaring a single api endpoint with a concrete HTTP method, class, and types.

```
@HttpExchange("/persons")
interface PersonService {

    @GetExchange("/{id}")
```

```
    Person getPerson(@PathVariable Long id);

    @PostExchange
    void add(@RequestBody Person person);
}
```

```
@RestController
class PersonController implements PersonService {

    public Person getPerson(@PathVariable Long id) {
        // ...
    }

    @ResponseStatus(HttpStatus.CREATED)
    public void add(@RequestBody Person person) {
        // ...
    }
}
```

Matrix Variables

- Method variables let you embed filters into uri path segments. Think iof it as certain criteria for a search or url path to go through
- Matrix variables beed to be configured with MVC config
- See Matrix Variables for more

`@RequestBody`

- This annotations is useful for when you need the body of the object you are requesting and deserialized into a java `Object`

```
@PostMapping("/accounts")
public void handle(@RequestBody Account account) {
    // ...
}
```

`@RequestParam`

- This annotation is good for binding servlet request parameters. Think of these as filters for a search feature on your website.
- The following example would be a search function that takes in optional values. Its routed at the endpoint /docs. On request, the method will return a list based on provided parameters

```java
@GetMapping("/api/docs")
public Page<Doc> listDocuments(
    @RequestParam(required = false) String author,
    @RequestParam(required = false) String tag,
    @RequestParam(required = false) String title,
    @RequestParam(defaultValue = "0") int page,
    @RequestParam(defaultValue = "20") int size
) {
    // Any of author/tag/title may be null.
    // Your service can build a query based on whichever filters are non-
null.
    return docService.search(author, tag, title, page, size);
}
```

- To separate the request explicit endpoints you can also define distinct handler methods

```java
@GetMapping("/api/docs/by-author")
public List<Doc> byAuthor(@RequestParam String author) { … }

@GetMapping("/api/docs/by-tag")
public List<Doc> byTag(@RequestParam String tag) { … }

@GetMapping("/api/docs/search")
public List<Doc> search(
    @RequestParam String keyword,
    @RequestParam(required = false) String author
) { … }
```

HttpEntity

- Container object equivalent to `@RequestBody` that exposes request headers and body
- Only use when you need access to headers and body

```java
@PostMapping("/accounts")
public void handle(HttpEntity<Account> entity) {
    // ...
}
```

`@ResponseBody`

- You can use the `@ResponseBody` annotation on a method to have the return serialized to the response body through an HttpMessageConverter

```
@GetMapping("/accounts/{id}")
@ResponseBody
public Account handle() {
    // ...
}
```

Response Entity

- Same as `@ResponseBody` but include status and headers
- the body will be provided as a value object

```
@GetMapping("/something")
public ResponseEntity<String> handle() {
    String body = ... ;
    String etag = ... ;
    return ResponseEntity.ok().eTag(etag).body(body);
}
```