Spring MVC And Others

Spring MVC(基本sense，保证能回答问答题)

Repo: https://github.com/TAIsRich/springmvc5-demo.git

Branch: main

只需要clone，不需要自己写一遍。

要点：
理解下面三幅图，并能口述图三的流程。

切忌：去学JSP 细节。

Reference

Spring MVC: https://www.youtube.com/watch?v=BkRZfxznaOo

Spring MVC introduction

Spring Web MVC is the original web framework built on the Servlet API and has been included inthe Spring Framework from the very beginning.

MVC Pattern

Controller

The front controller maps the incoming request to a controller. Controllers contain the business logic of the application. They also handle requests and perform marshalling/unmarshalling. The function of the controller is to handle a user request. The incoming request may be handled in different ways like reading form data, processing it in some way, storing data in a database, or retrieving data from a web service etc. The controller places the data in the model which, simply put, is a container for the data. The controller returns the model (containing data) back to the front controller.

Model

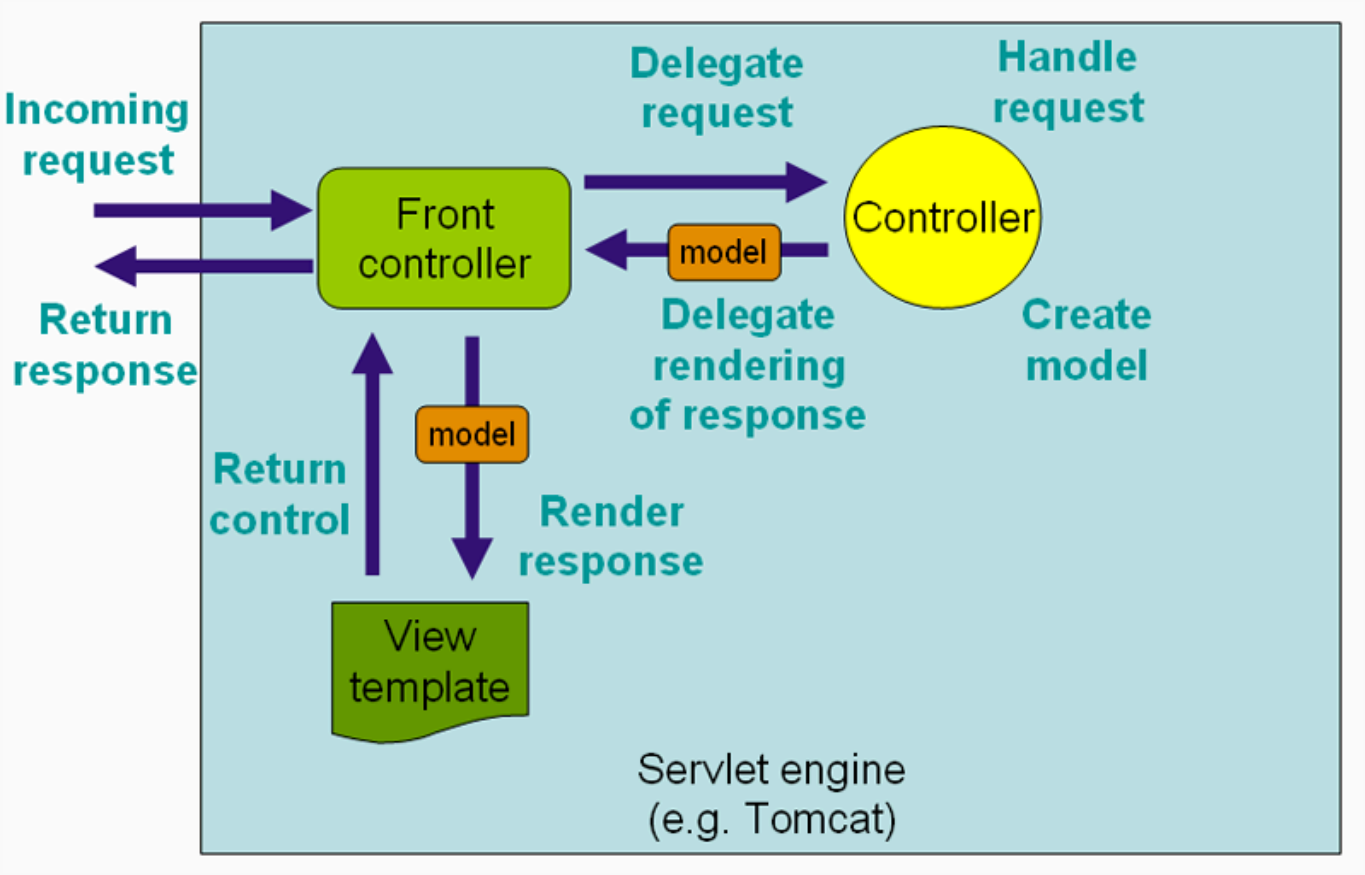
Model is a container for data. It is used to transfer data from one part of the Spring MVC application to another.

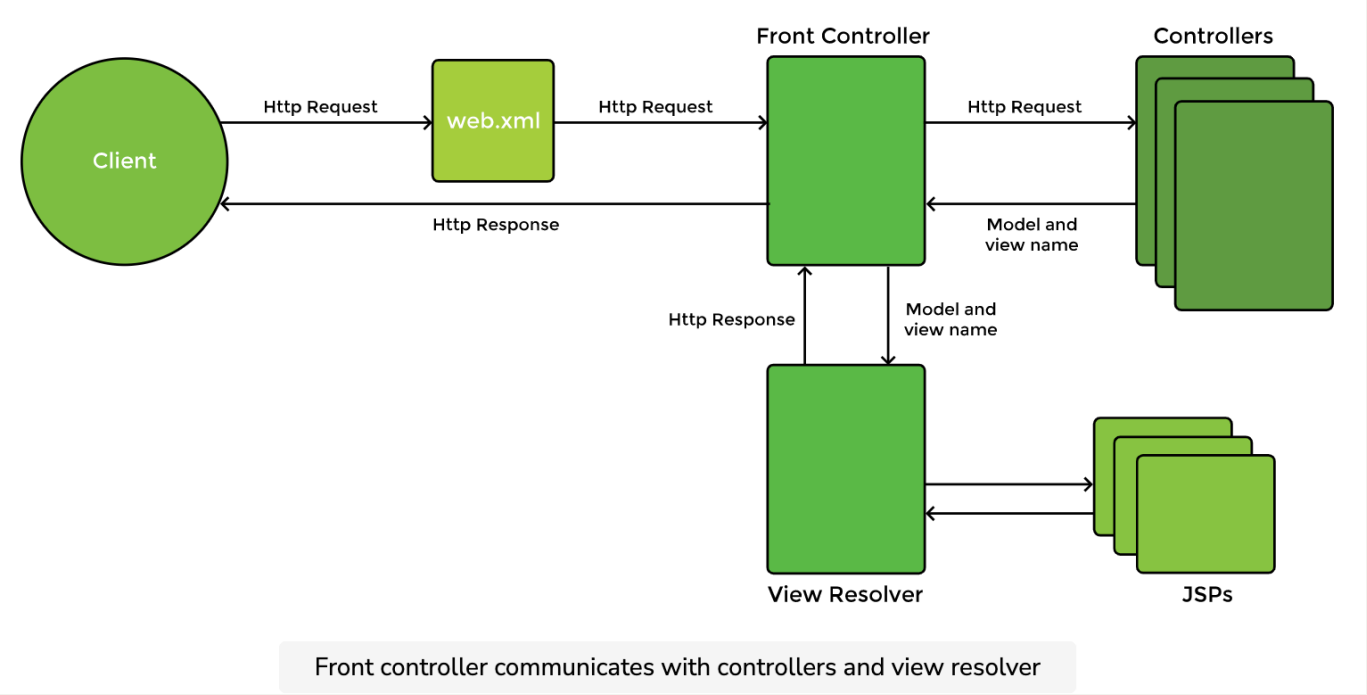
Controller populates the model with data from a form or a database or a web service. As can be seen from the Spring MVC request flow diagram, the controller passes the model to the front controller which forwards it to the view resolver. The view template displays the data in the model.

View (template)

Spring MVC supports a number of view templates. The most commonly used are JSP or JSTL (JSP Standard Tag Library). Other view templates like Thymeleaf, Groovy, Velocity, and FreeMarker etc., can also be plugged in.

View template receives the model containing data. It reads the model and displays the data. If, say, the model contains a list of players, the view template can create a table to display that list. In most cases a view template is a JSP page that provides data to the user.





Front controller communicates with controllers and view resolver

pring MVC offers many advantages. Some of them are listed below:

- Spring MVC provides a clear separation of concerns between the controllers, model and view objects.
- Spring MVC is flexible and does not force the developer to use JSP as the view technology. The developer is free to use Velocity, Thymeleaf or even design his own view mechanism and seamlessly integrate it. The controllers are not dependent on the view technology and can work independently even if the view technology is changed.
- Spring MVC application is testable as controllers are configured like other objects. The framework also supports Test Driven Development (TDD).
- The Spring tag library provides data binding and validation facility.
- Spring MVC uses `WebApplicationContext` to scope the beans to an HTTP Request or an HTTP Session. This helps in tracing the state of the web request.

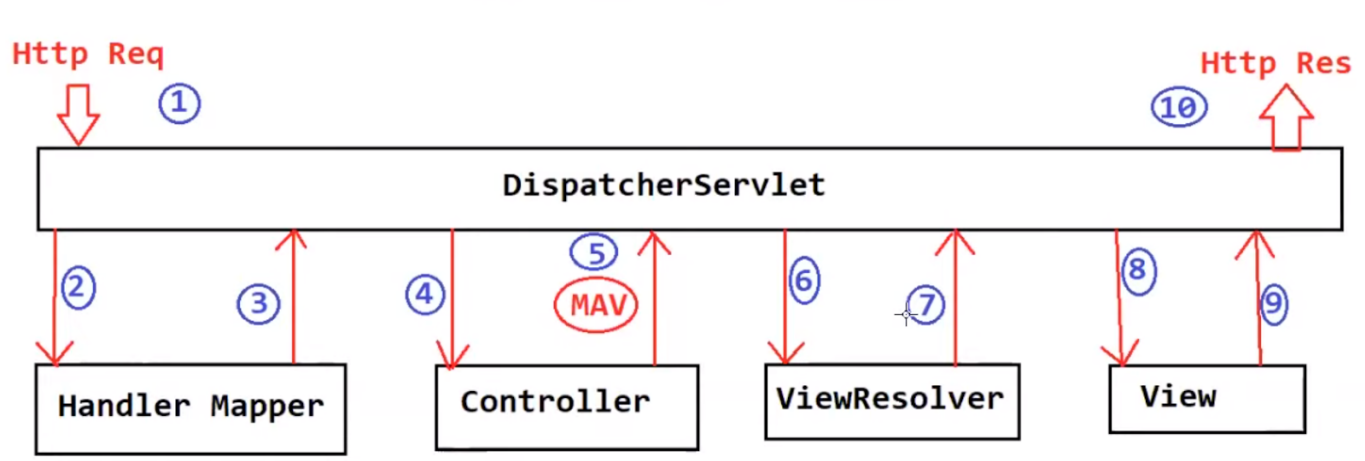
Front Controller is DispatcherServlet

DispatcherServlet(重点，难)

DispatcherServlet abstracts away the following tedious and boilerplate tasks and focus on useful business logic:

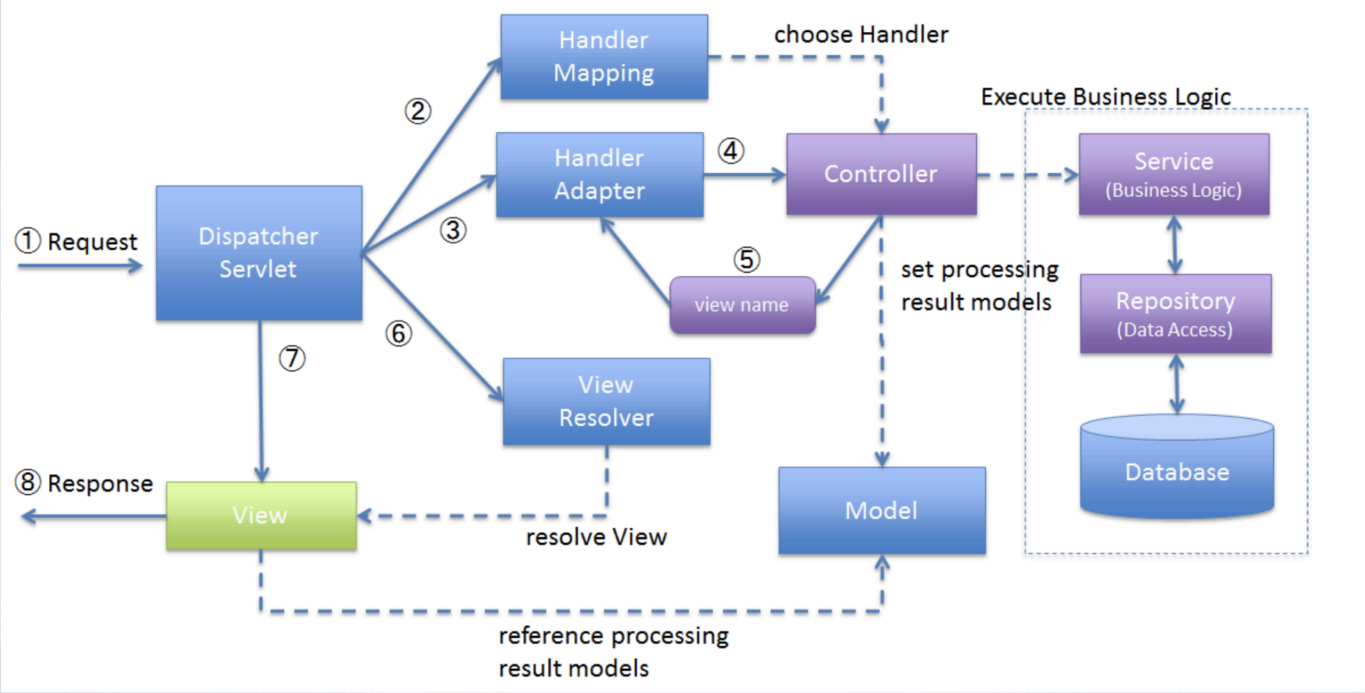
- Mapping an HTTP request to a certain processing method
- Parsing of HTTP request data and headers into data transfer objects (DTOs)or domain objects
- Model-view-controller interaction
- Generation of responses from DTOs, domain objects, etc.

流程图

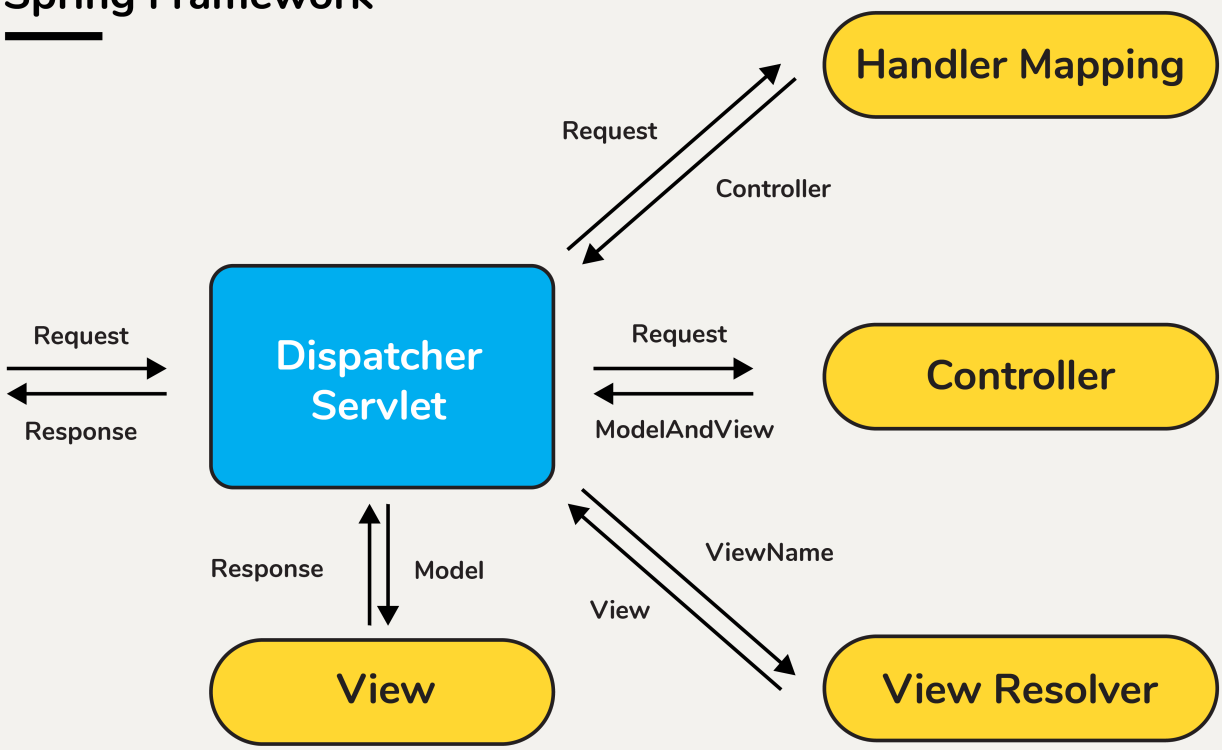


1. Tomcat 收到Http Request，将request交给DispatcherServlet来处理
2. DispatcherServlet 拿着该req去call HandlerMapper,
3. HanlerMapper将会找到对应的Controller以及对应的method，并返回给dispactherServlet
4. dispatcherServlet将call该Controller对应的method，此时会触发call service, repository and database.
5. 然后结果(ModelAndView)再通过Controller返回。一般是返回view name, 一个字符串，比如list-customers. 而数据是要set到Model 中。
6. dispatcerServlet拿着view name去call view Resolver,
7. View Resolver会帮助我们找到view template, 比如list-customers.jsp

8. 此时我们有了view template, 也有了数据model，则可以call View engine去帮助我们把数据放到view templatel里，然后转换成纯粹的HTML
9. 该HTML就是前端显示的内容，最终返回给browser。



Spring Framework

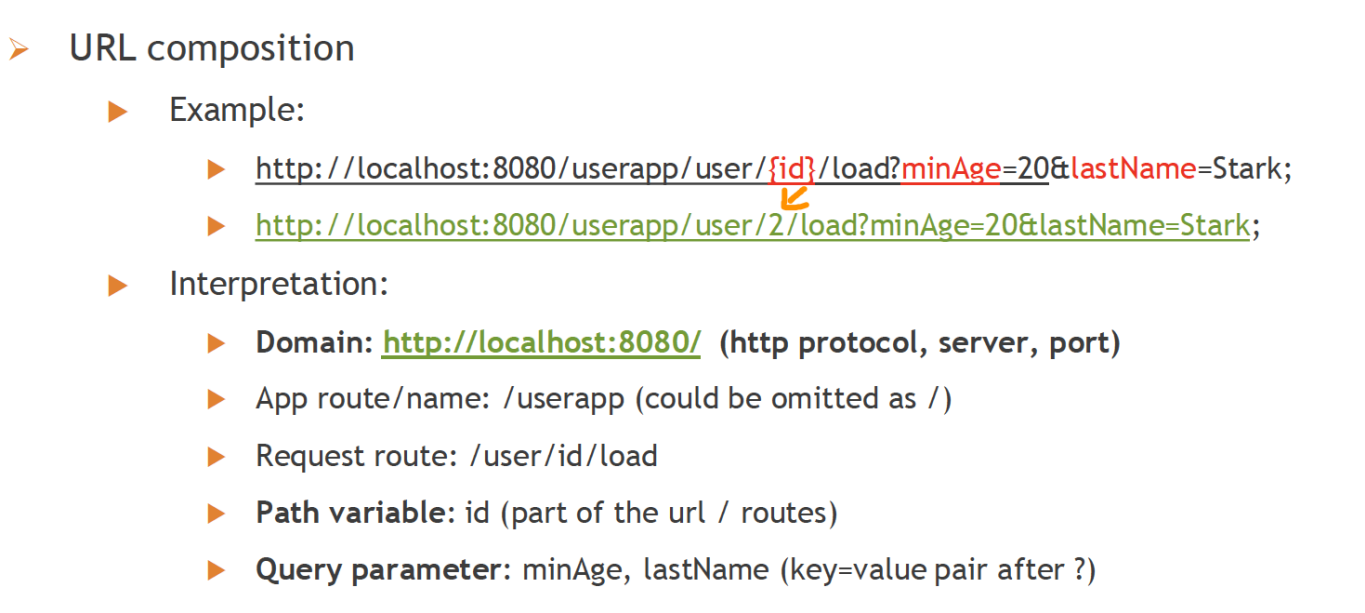


InterviewBit

We can configure the DispatchServlet by either annotation or XML (should we learn?)

Controller

URL Composition - recap



Request Mapping Methods - recap

- @GetMapping - R
- @PutMapping - U
- @PostMapping - C
- @DeleteMapping - D
- @RequestMapping(method=RequestMethod.GET)

@RestController vs. @Controller (面试)

@RestController = @Controller + @ResponseBody

```
@RestController
@RequestMapping("/api/v1/posts")
public class PostController {

    @Autowired
    private PostService postService;

    @PostMapping()
    public ResponseEntity<PostDto> createPost(@RequestBody PostDto postDto) {
        PostDto postResponse = postService.createPost(postDto);
        return new ResponseEntity<>(postResponse, HttpStatus.CREATED);
    }
}

vs.

@Controller
@RequestMapping("/api/v1/posts")
public class PostController {

    @Autowired
    private PostService postService;

    @PostMapping()
    public @ResponseBody ResponseEntity<PostDto> createPost(@RequestBody PostDto postDto) {
        PostDto postResponse = postService.createPost(postDto);
        return new ResponseEntity<>(postResponse, HttpStatus.CREATED);
    }
```

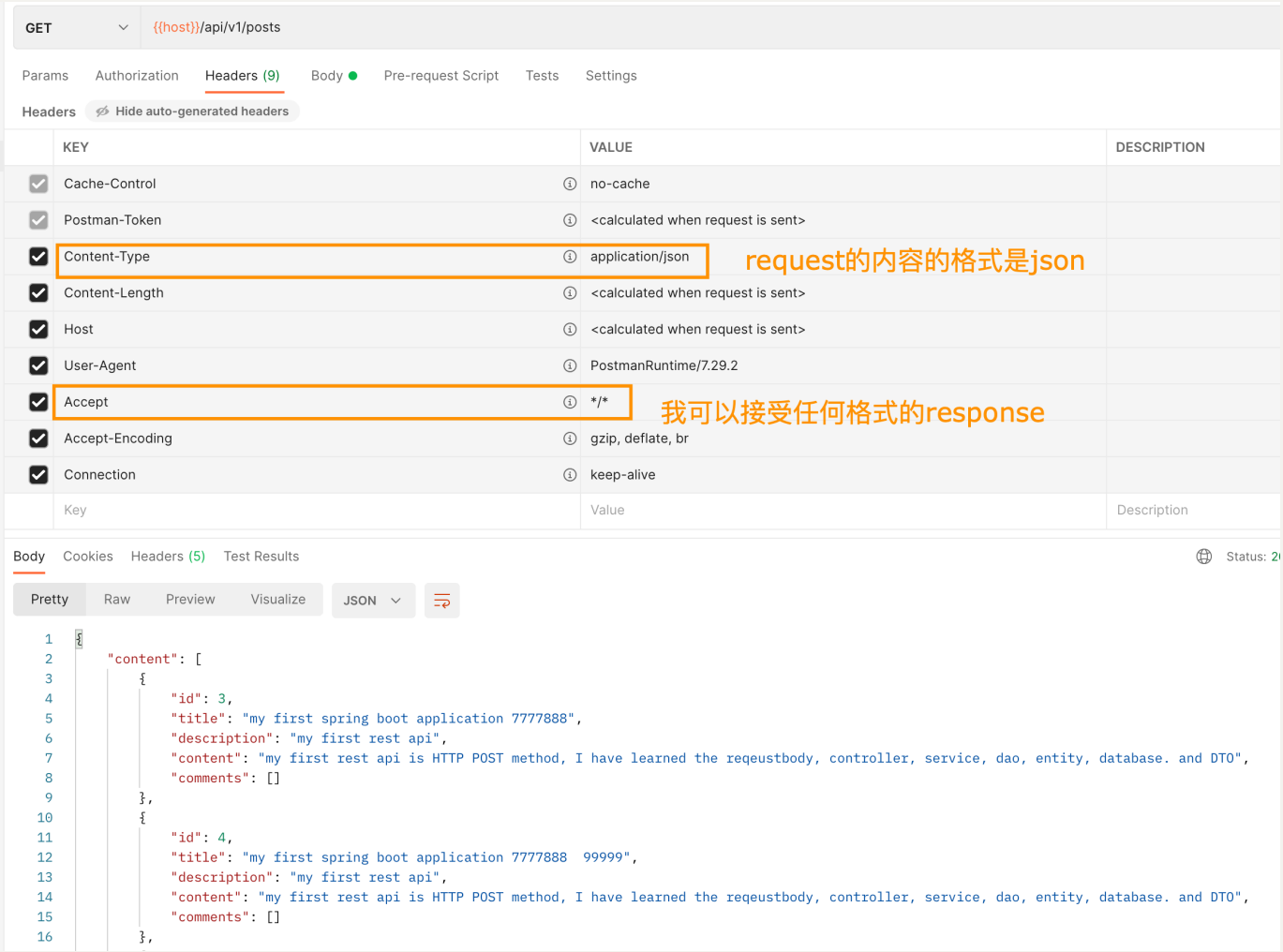
```
29  }
```

HttpHeader & Content Negotiation (Produce & Consume Type) (高频面试题)

RequestHeader (Front-end)

Content-Type: application/json 我们的request的内容格式是JSON

Accept: application/json 我们接受的内容格式必须是JSON，*/*代表任意格式。



Content Negotiation

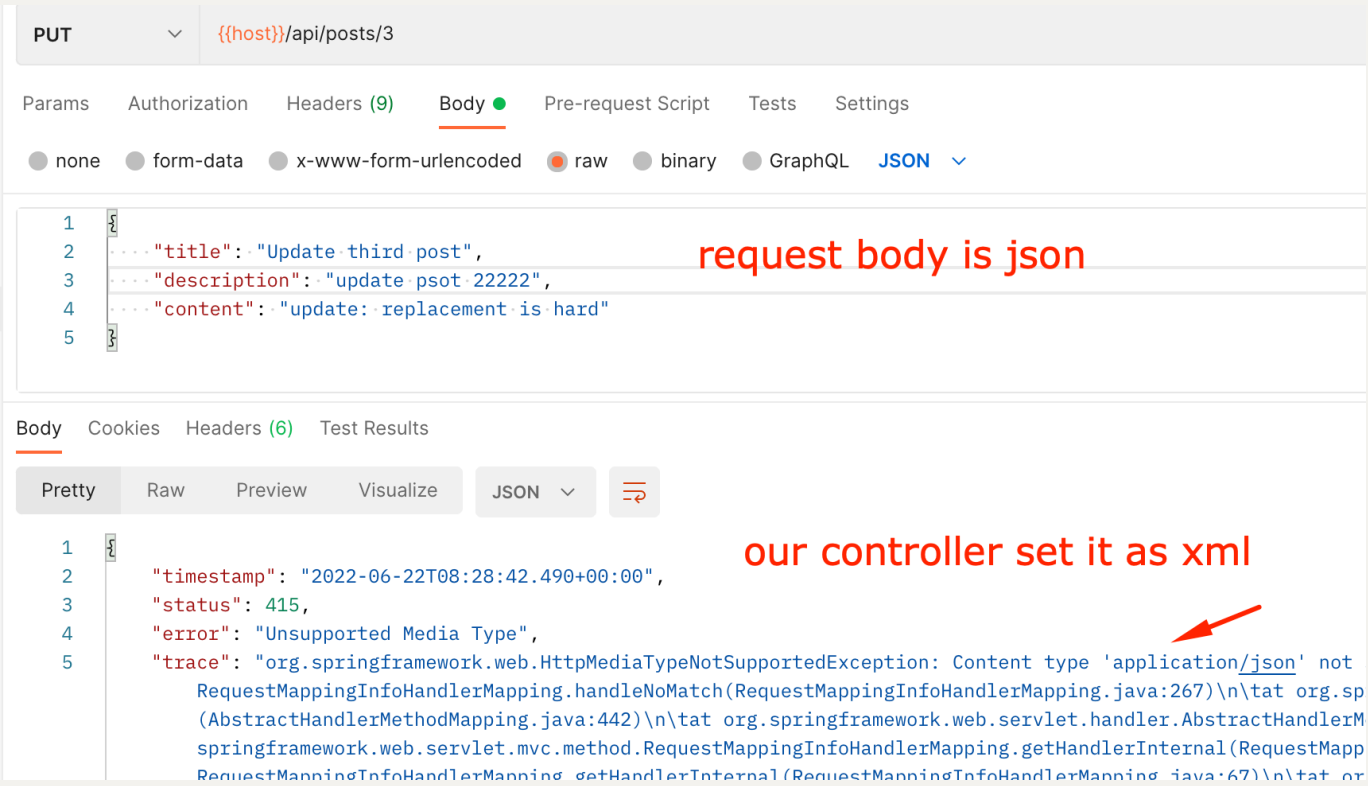
Back-end: **consumes** consume content-type from Reqeust

produces contents to front-end, frontend only **accept** the type it defined

set it to consume xml, and produce json

```
    @PutMapping(value = "/{id}", produces = "text/pdf", consumes = "application/xml")
    public ResponseEntity<PostDto> updatePostById(@RequestBody PostDto postDto, @PathVariable(name = "id") long id) {
        PostDto postResponse = postService.updatePost(postDto, id);

        return new ResponseEntity<>(postResponse, HttpStatus.OK);
    }
```



set it to consume json, and produce xml

```
    @PutMapping(value = "/{id}", produces = "application/xml",
consumes = "application/json")
    public ResponseEntity<PostDto> updatePostById(@RequestBody
PostDto postDto, @PathVariable(name = "id") long id) {
        PostDto postResponse = postService.updatePost(postDto, id);

        return new ResponseEntity<>(postResponse, HttpStatus.OK);
    }
```

Cookie vs Session

Cookie: front end storage, small size, can be used with every http request, popular for application with independent front end.

Session: back end storage, has expiration time (~20mins), corresponding to front end tab. Expires if time is reached without user interactions or the tab is closed. Usually used with jsp.