一、web mvc开发的基础知识

1、spring-boot-starter-web介绍

spring boot崇尚的就是尽可能减少手动配置，尽量基于约定规则来完成所有的配置，同时尽量基于注解来完成配置。只要在依赖中引入spring-boot-starter-web，其实spring boot就会基于自己的Auto Configuration功能，完成web服务器和spring mvc的自动配置，然后我们上手就可以开发web程序，直接运行main类就可以启动内置的web服务器了。这能让我们快速启动web程序的开发、调试和验证。

2、Spring MVC介绍

spring mvc是一个mvc层的框架，可以接受http请求，将请求转发路由到对应的类方法中来处理，接着将响应映射到视图文件，或者直接返回给浏览器，常见的spring mvc的controller如下所示：

@RestController

@RequestMapping(value="/users")

public class MyRestController {

   @RequestMapping(value="/{user}", method=RequestMethod.GET)

   public User getUser(@PathVariable Long user) {

       // ...

  }

   @RequestMapping(value="/{user}/customers", method=RequestMethod.GET)

   List<Customer> getUserCustomers(@PathVariable Long user) {

       // ...

  }

   @RequestMapping(value="/{user}", method=RequestMethod.DELETE)

   public User deleteUser(@PathVariable Long user) {

       // ...

  }

}

3、spring boot为spring mvc做的auto configuration

（1）自动注册了ContentNegotiatingViewResolver和BeanNameViewResolver两个bean

（2）支持处理静态资源

（3）自动注册了Converter、GenericConverter、Formatter几个bean

（4）支持HttpMessageConverters

（5）自动注册了MessageCodesResolver

（6）静态index.html支持

（7）自定义Favicon图标支持

（8）自动使用了ConfigurableWebBindingInitializer bean

上面这一大坨是什么东东？其实我们都不用多管，因为很多东西很少会用到，只要知道这个auto configuration，会自动给我们完成一些spring mvc的配置，不需要我们手动去搞一个mvc-servlet.xml配置

但是接下来我们也会挑选一些东西重点讲一下，spring boot为spring mvc的方便使用，做了哪些事情

4、RESTful风格的接口

（1）在接口中增加api字样表明这是一个api接口、

http://localhost:8080/api

（2）需要在接口中增加版本标识

http://localhost:8080/api/v1.0

（3）按照一定风格写清楚标识的资源

http://localhost:8080/api/v1.0/system/employee

这里表示的就是系统管理模块中的员工管理

（4）RESTful中的HTTP method

POST：代表的是新增

PUT：代表的是修改

GET：代表的是查询

PATCH：代表的是更新部分属性

DELETE：代表的是删除

HEAD：获取头信息，要查找的信息在响应头中

OPTIONS：获取URI支持的方法，在响应头中会有Allow，包含了支持的方法

新增员工：POST http://localhost:8080/api/v1.0/system/employee

修改员工：PUT http://localhost:8080/api/v1.0/system/employee/1

查询员工：GET http://localhost:8080/api/v1.0/system/employee/1

查询所有员工：GET http://localhost:8080/api/v1.0/system/employee

删除员工：DELETE http://localhost:8080/api/v1.0/system/employee/1

（5）RESTful中的HTTP status

200，成功

400，错误的请求，比如请求参数匹配错误

404，没找到这个资源

405，使用的HTTP method不被允许（GET/POST使用错误）

500，服务器内部错误

5、spring boot支持RESTful接口

只要依赖了spring-boot-starter-web，就自动支持RESTful接口

RESTful接口的开发

// 支持RESTful接口的controller直接用@RestController即可

// 现在都是前后端彻底分离了，因此后端一般就是使用RestController就足够了

// 我们就对外暴露接口

@RestController

// 这里就是表明了是/api接口，版本是/v1，/employee表明这是员工管理模块

@RequestMapping("/api/v1/employee")

public class EmployeeController {

   @Autowired

   private EmployeeService employeeService;

   // 这里用了@GetMapping，就代表只能接受GET请求

   // {employeeId}表明用通配符匹配传递过来的员工ID

   @GetMapping("/{employeeId}")

   public Employee findById(@PathVariable("employeeId") Long employeeId) {

       return employeeService.findById(employeeId);

  }

   // 这里用了@PostMapping，表明仅仅接受POST请求

   @PostMapping

   public Long add(Employee employee) {

       return employeeService.add(employee);

  }

   // 这里用了@DeleteMapping，表明仅仅接受DELETE请求

   @DeleteMapping("/{employeeId}")

   public Boolean delete(@VariablePath("employeeId") Long employeeId) {

       return employeeService.delete(employeeId);

  }

}

测试的时候，使用curl命令：

curl -XGET 'http://localhost:8080/api/v1/employee/1'

curl -XPOST 'http://localhost:8080/api/v1/employee' -H 'Content-Type:application/json' -d'

{

  "name": "张三",

  "age": 30

}

'

curl -XDELETE 'http://localhost:8080/api/v1/employee/1'

chrome的postman，如果要传递参数的话，通常建议都是前端走ajax调用，用json格式+Content-Type=application/json，来传递对应的一些参数

通过Json格式来走输入和输出，是业界的通用的规范，包括我们系统开发也都是走json的。那就是用类似于dubbu走rpc调用。

6、系统中调用RESTful接口【百度内部的Ral调用】

spring boot提供了一个RestTemplate来辅助发起REST请求，底层通过HttpURLConnection来发送HTTP小覅西，使用Jackson来进行JSON序列化

这个我就给大家提一句，不想带着大家来做了，因为在实际的开发过程中，一般不会用这个东西。后面要去讲解spring cloud微服务架构，spring cloud已经把这个事情给干了。如果你是一套走微服务的架构，http来走restful接口，那就直接用spring cloud就可以了。

唯一的可能，就是别人开发了一个提供http接口的一个应用，需要你的系统去对他进行调用。那么可能需要使用到这个RestTemplate。我带着大家简单的来做一个例子，如果大家后面真的遇到了这样的这个情况，我觉得太简单了，网上查一查用法，留给你们的作业。

@Controller

@RequestMapping("/test")

public class TestController {

   @Autowired

   RestTemplateBuilder restTemplateBuilder;

@GetMapping("/user/{id}")

public User testGetUserById(@PathVariable("id") Long id) {

RestTemplate client = restTemplateBuilder.build();

       String uri = "http://localhost:8080/api/v1.0/user/{id}";

       return client.getForObject(uri, User.class, id);

}

}

RestTemplate还有其他的一些用法

// 传递参数

client.getForObject(uri, Order.class, map);

// 获取HTTP头信息

ResponseEntity<Order> responseEntity = client.getForEntity(uri, Order.class, orderId);

Order order = responseEntity.getBody();

HttpHeaders headers = responseEntity.getHeaders();

// 发送post请求

client.postForObject(uri, order, String.class);

// 查询一组订单

Integer offset = 1;

HttpEntity body = null;

ParameterizedTypeReference<List<Order>> typeRef = new ParameterizedTypeReference<List<Order>>(){};

ResponseEntity<List<Order>> rs = client.exchange(uri, HttpMethod.GET, body, typeRef, offset);

List<Order> orders = rs.getBody();

7、@RestController注解

RESTful风格的Controller中，直接用这个注解即可，对每个方法的返回值默认采用@ResponseBody的方式来处理

不会去找某个视图资源文件来渲染后发回浏览器，因为@RestController就代表了这是一个负责提供RESTful接口的Controller

8、@RequestMapping注解

（1）一般在Controller类级别加一个@RequestMapping注解，代表对一类资源的映射

（2）在Controller中每个方法级别加一个@RequestMapping，支持某一类请求

9、@PathVariable注解

// 这个功能在RESTful接口设计中很常见

// 就是在请求中附带一个id或者别的什么参数，代表了对一个资源的标识和定位

// 然后在方法入参中使用@PathVariable注解配置请求中参数和方法入参的映射关系

@RequestMapping(value = "/{userId}", method = HttpMethod.GET)

public User getUserById(@PathVariable("userId") Long userId) {

}

10、@XMapping系列注解

如果是做REStful风格的Controller，其实一般就更加简化，使用@GetMapping，@PutMapping，@PostMapping，@DeleteMapping。来代表了增删改查等一些操作和请求即可。

@GetMapping("/{userId}")

public User getUserById(@PathVariable("userId") Long userId) {

}

@PostMapping("/")

public String saveUser(User user) {

}

@PutMapping("/{userId}")

public String updateUser(@PathVariable("userId") Long userId,

   @ModelAttribute User user) {

}

@DeleteMapping("/{userId}")

public String removeUser(@PathVariable("userId") Long userId) {

}

11、ModelAndView

这个其实一般用于传统的MVC模式中，就是Controller接收请求，调用Service获取结果后，再将结果封装到ModelAndView中去，同时在里面指明请求要转发到哪个视图资源文件上去。接着JSP、FreeMarker、Thymeleaf等模板引擎技术，可以从ModelAndView中提取之前封装进去的数据，然后将数据渲染到网页上去

但是在现在的大趋势下，这种技术已经很少使用了，除非是一些一个人开发的极其极其小的小项目，才会用MVC模式，否则正经公司里的生产级项目，全部都是彻底的前后端分离了，我们是不care网页模板的，仅仅提供后端RESTful或者rpc风格的接口，提供数据即可

@GetMapping("/{userId}")

public ModelAndView getUserById(@PathVariable("userId") Long userId, ModelAndView mav) {

   User user = userSerivce.getUserById(userId);

   mav.addObject("user", user);

   mav.setViewName("/userInfo.jsp");

   return mav;

}

12、将请求参数映射到方法入参

使用@RequestParam可以将某个请求参数，映射到方法入参上

// 假设浏览器发送过来了一个user\_status=1的请求参数

// 可以将其映射到方法入参userStatus上去【下划线转驼峰】

@GetMapping("/{userId}")

public User getUserById(@PathVariable("userId") Long userId,

                       @RequestParam("user\_status") Integer userStatus) {

}

13、@ResponseBody

就是将方法的返回值直接用json等格式返回到浏览器，一般我们现在都直接用@RestController了，自带这个功能

二、web开发中对请求参数的校验

14、Spring Boot对请求校验的支持

Spring Boot支持JSR-303验证框架，默认实现是Hibernate Validator，只要在Java Bean上放一些校验注解，就可以实现校验支持

常用的校验注解包括下面这些：

（1）空检查：@Null，@NotNull，@NotBlank，@NotEmpty

（2）长度检查：@Size(min=20, max=50)，@Length

（3）数值检查：@Min，@Max，@Range(min=1, max=99)

（4）其他检查：@Email，@Pattern

而且这个校验是支持group的概念的，对于不同的group生效的校验不一样。这个很有用，因为对于增删改查等不同的操作，需要执行的校验本来就是不一样的

15、基本的校验案例

public class User {

   public interface Update{}

   public interface Add{}

   // 这就是一个典型的group分组

   // 更新操作的时候，要求userId必须不为空

   // 新增操作的时候，要求userId必须为空

   @NotNull(groups = {Update.class})

   @Null(groups = {Add.class})

   private Long userId;

   @Size(min=3, max=20)

   private String name;

   @Range(min=1, max=120)

   private Integer age;

}

// 这里就声明了要激活Add group对应的校验注解

// 那么就会校验user的userId不能为空

@PostMapping("/")

public String saveUser(@Validated({User.Add.class}) User user,

                     BindingResult result) {

// 如果校验失败，则将校验信息返回

   if(result.hasErrors()) {

       List<ObjectError> errors = result.getAllErrors();

       FieldError error = (FieldError)errors.get(0);

       System.out.println(error.getObjectName() + ","

                          + error.getField() + ","

                          + error.getDefaultMessage());

       return "error";

  }

   return "success";

}

16、自定义校验注解

public class AgeValidator implements ConstraintValidator<Age, Integer> {

   private Age age;

   private Integer max;

   public void initialize(Age age) {

       this.age = age;

       this.max = age.max();

  }

   public boolean isValid(Integer value, ConstraintValidatorContext context) {

       return value < max;

  }

}

@Constraint(validatedBy = {AgeValidator.class})

@Documented

@Target({ElementType.ANNOTATION\_TYPE, ElementType.METHOD, ElementType.FIELD})

@Retention(RetentionPolicty.RUNTIME)

public @interface Age {

   String message() default "年龄是非法的，不能超过{max}岁";

   int max() default 100;

   Class<?>[] groups default {};

   Class<? extends Payload>[] payload() default {};

}

三、spring boot+mvc中常用的一些全局配置

17、WebMvcConfigurer

@Configuration

public class MvcConfigurer implements WebMvcConfigurer {

   // 拦截器【对哪些url进行指定拦截器的内容】

   public void addInterceptors(InterceptorRegistry registry) {

       registry.addInterceptor(new UserLoginInterceptor()).addPathPatterns("/user/\*\*");

  }

   // 跨域访问配置【指定url，只允许指定网站的指定方式的请求访问】

   public void addCorsMappings(CorsRegistry registry) {

       registry.addMapping("/user/\*\*")

          .allowedOrigins("http://www.zhss.com")

          .allowedMethods("POST", "GET")

  }

   // 格式化【入参数据从json中提取后按指定方式转型后注入bean，比如日期格式变更】

   public void addFormatters(FormatterRegistry registry) {

       registry.addFormatter(new DateFormatter("yyyy-MM-dd HH:mm:ss"));

  }

   // URI到视图的映射【指定urI不去找controller，直接映射到对应页面 --- 很少用】

   public void addViewControllers(ViewControllerRegistry registry) {

       registry.addViewController("/addUserUI").setViewName("/addUser.jsp");

  }

}

【用户登录拦截器】

public class UserLoginInterceptor implements HandlerInterceptor {

   public boolean preHandle(HttpServletRequest request,

                            HttpServletResponse response, Object handler)

                            throws Exception {

Boolean result = Boolean.valueOf(request.getParameter(“\_login”));

sout(result);

return result;

  }

   public void postHandle(HttpServletRequest request,

                          HttpServletResponse response,

                          Object handler,

                          ModelAndView moodelAndView)

                          throws Exception {

  }

   public void afterCompletion(HttpServletRequest request,

                               HttpServletResponse response,

                               Object handler,

                               Exception ex)

                               throws Exception {

  }

}

18、spring监听器的使用

【监听事件为ApplicationStartedEvent，为系统自带的事件，已经废弃。可自定义监听的事件】

public class MyApplicationStartedEventListener implements ApplicationListener<ApplicationStartedEvent> {

   @Override

   public void onApplicationEvent(ApplicationStartedEvent event) {

  }

}

@SpringBootApplication

public class Application {

   public static void main(String[] args) {

       SpringApplication app = new SpringApplication(Application.class);

       app.addListeners(new MyApplicationStartedEventListener());

       app.run(args);

  }

}

19、全局异常处理

Controller中抛出的异常全部交给/error来处理，Spring Boot是有一个默认实现的，但是也可以我们自己覆盖掉

@ControllerAdvice

class GlobalErrorHandler {

   @ExceptionHandler(value = Exception.class)

   @ResponseBody

   public ErrorInfo<String> defaultErrorHandler(HttpServletRequest req, Exception e) throws Exception {

  ErrorInfo<String> r = new ErrorInfo<String>();

       r.setMessage(e.getMessage());

       r.setCode(ErrorInfo.ERROR);

       r.setData("自定义的全局异常处理");

       r.setUrl(req.getRequestURL().toString());

       return r;

  }

}

public class ErrorInfo<T> {

   public static final Integer OK = 0;

   public static final Integer ERROR = 100;

   private Integer code;

   private String message;

   private String url;

   private T data;

public Integer getCode() {

return code;

}

public void setCode(Integer code) {

this.code = code;

}

public String getMessage() {

return message;

}

public void setMessage(String message) {

this.message = message;

}

public String getUrl() {

return url;

}

public void setUrl(String url) {

this.url = url;

}

public T getData() {

return data;

}

public void setData(T data) {

this.data = data;

}

}

四、spring boot跟其他技术整合的一些要点

20、spring boot与模板的整合

spring boot的静态资源支持规范如下所示：

（1）/static：静态资源，js，css，图片之类的。这里放一张图片，然后访问http://localhost:8080/a.jpg，就会自动去/static下面找图片

（2）/templates：放置模板文件，支持thymeleaf，freemarker，jsp，velocity等。但是spring boot不建议使用jsp，同时velocity很多年不更新了。因此现在主流建议是thymeleaf或者是freemarker

当然，这个就不多说了，因为如果你做了一个项目，是前后端耦合在一起的，那赶紧走，因为真的就太low了，这种一定是小项目，而且绝对不是未来的趋势

加入依赖

<dependency>

   <groupId>org.springframework.boot</groupId>

   <artifactId>spring-boot-starter-thymeleaf</artifactId>

</dependency>

编写controller

@Controller

public class UserController {

   @Autowired

   private UserService userService;

   @GetMapping("/{userId}")

   public String getUserById(@PathVariable("userId") Long userId, ModelAndView modelAndView) {

       User user = userService.getUserById(userId);

       modelAndView.addObject("user", user);

       modelAndView.setViewName("userInfo.html");

       return modelAndView;

  }

}

编写thymeleaf模板页面

<!DOCTYPE html>

<html>

   <head lang="en">

   <meta charset="UTF-8" />

   <title>用户信息页面</title>

</head>

<body>

  用户ID：<p th:text="${user.userId}" ></p>

  用户名：<p th:text="${user.username}" ></p>

  用户年龄：<p th:text="${user.age}" ></p>

</body>

</html>

21、spring boot与mybatis的整合要点

@Insert("INSERT INTO user(name, age) VALUES(#{name}, #{age})")

@Options(useGeneratedKeys = true, keyProperty = "user.userId")

void saveUser(User user);

@Update("UPDATE user SET name=#{name}, age=#{age} WHERE user\_id=#{userId}")

void updateUser(User user);

@Delete("DELETE FROM user WHERE user\_id=#{userId}")

void removeUser(@Param("userId") Long userId);

@Select("SELECT FROM user WHERE user\_id=#{userId}")

@Results({

   @Result(property = "userId", column = "user\_id", id = true),

   @Result(property = "name", column = "name"),

   @Result(property = "age", column = "age")

})

User getUserById(@Param("userId") Long userId);

22、spring boot对事务的支持

<dependency>

   <groupId>org.springframework.boot</groupId>

   <artifactId>spring-boot-starter-jdbc</artifactId>

</dependency>

@Transactional

public void updateUser(User user) {

}

关于事务，有很多更加高级的东西，包括事务的隔离级别，事务的传播，不在这里深入，而是放在后面的MySQL高阶深入环节，要配合mysql事务的透彻剖析和讲解，再在这里来实践使用即可

853769620