# A4设计报告

1.数据层设计

将数据库改为MongoDB-react以适应Reactive编程。

User类：包括id，用户名、密码

public class User extends Base /\*implements Serializable \*/{

@Id

private Integer id;

private String username;

private String password;

}

UserRepository 类：根据用户名和密码返回User，根据用户名返回User

public interface UserRepository extends ReactiveMongoRepository<User, String> {

Mono<User> findByUsernameAndPassword(String username, String password);

Mono<User> findByUsername(String username);

}

Gym类：id，健身馆名，健身馆信息

public class Gym extends Base implements Serializable {

@Id

private Integer id;

private String name;

private String info;

}

GymRepository 类：根据名称查找Gym，根据Id查找Gym，查找所有Gym

public interface GymRepository extends ReactiveMongoRepository<Gym, String> {

Mono<Gym> findByName(String name);

Mono<Gym> findById (Integer name);

Flux<Gym> findAll();

}

Trainer类：id，教练名称、教练信息

public class Trainer extends Base implements Serializable {

@Id

private Integer id;

private String name;

private String info;

}

TrainerRepository 类：根据名称查找Trainer，根据Id查找Trainer，查找所有Trainer

public interface TrainerRepository extends ReactiveMongoRepository<Trainer, String> {

Mono<Trainer> findByName(String name);

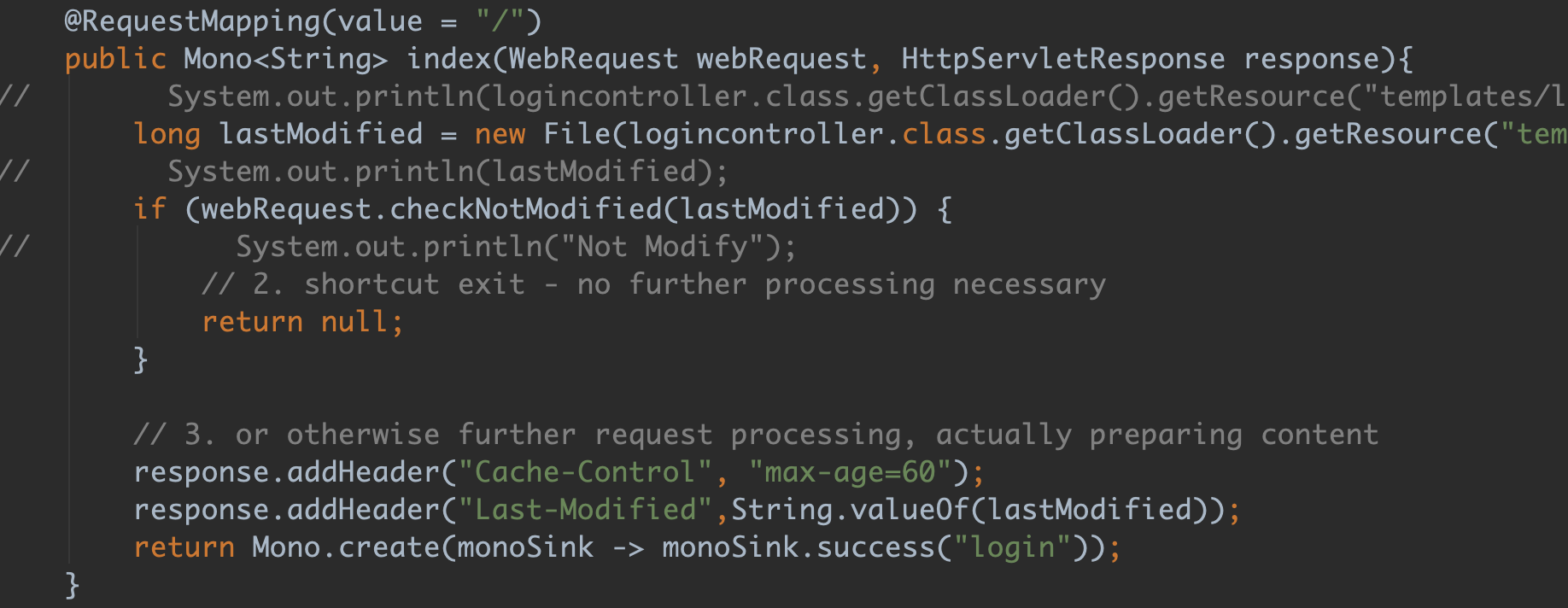
Mono<Trainer> findById (Integer name);

Flux<Trainer> findAll();

}

2.Mono+thymeleaf

在Controller中新增方法，view层使用thymeleaf模版引擎，处理MongoDB返回的数据并显示在前端页面上



EchoHandler

@Override

public Mono<Void> handle(final WebSocketSession session) {

return session.send(

session.receive()

.map(msg -> session.textMessage("ECHO -> " + msg.getPayloadAsText())));

}

}

SearchFluxcontroller根据请求类型，返回MongoDB数据

@Controller

public class SearchFluxcontroller {

@Autowired

TrainerService ts;

@Autowired

GymService gs;

//限流器

@Autowired

GuavaRateLimiterServiceImpl gr;

/\*List<Gym> g1list=gs.\*/

@GetMapping("/info")

@SneakyThrows(NoSuchElementException.class)

public Mono<String> gogogo(@RequestParam("gym") String gym,

@RequestParam("coach") String coach,

Model mod) {

String str = gr.executeSeckill();

System.out.println(str + "," + new Date());

while (str == "被限制") {

str = gr.executeSeckill();

System.out.println(str + "," + new Date());

}

Object a;

Flux<Trainer> glist = ts.queryAll();

if (glist == null)

return Mono.create(monoSink -> monoSink.success("search"));

if (((gym == null || gym.equals("")) && (coach == null || coach.equals(""))))

mod.addAttribute("hello", glist);

if (!gym.equals("") && (coach == null || coach.equals(""))) {

/\*System.out.println(ts.toString());\*/

if (isNumeric(gym)) {

a = Integer.parseInt(gym);

} else

a = gym;

List<Trainer> trainerOfGym = null;

if (ts.query(a) != null)

mod.addAttribute("hello2", ts.query(a));

return Mono.create(monoSink -> monoSink.success("search"));

}

if (!gym.equals("") && !coach.equals("")) {

if (isNumeric(gym)) {

a = Integer.parseInt(gym);

} else

a = gym;

Mono<Trainer> glist4=null;

if (ts.query(a) != null) {

glist4 = ts.query(a);

}

LinkedList<Trainer> glist5 = new LinkedList<>();

if (isNumeric(coach)) {

a = Integer.parseInt(coach);

} else

a = coach;

Mono<Trainer> glist3 = ts.query(a);

/\*mod.addAttribute("hello",glist3);\*/

mod.addAttribute("hello", glist3);

}

if ((gym == null || gym.equals("")) && !coach.equals("")) {

if (isNumeric(coach)) {

a = Integer.parseInt(coach);

} else

a = coach;

Mono<Trainer> glist3 = ts.query(a);

mod.addAttribute("hello", glist3);

}

return Mono.create(monoSink -> monoSink.success("search"));

}

public static boolean isNumeric(String str) {

for (int i = 0; i < str.length(); i++) {

System.out.println(str.charAt(i));

if (!Character.isDigit(str.charAt(i))) {

return false;

}

}

return true;

}

}