

MybatisPlus

Mybatis Plus 是 Mybatis 的增强版

特点

- Mapper 接口 MybatisPlus 延续了 BaseMapper 接口

```
public interface UserMapper extends BaseMapper<User> {  
}
```

- BaseMapper 增加了 CRUD 操作方法
- MybatisPlus 增加了更多操作方法

注解

- 表注解

 - User -> user
 - UserInfo -> user_info

- id 注解
- 字段注解

表注解

- @TableName 表名
- @TableId 主键注解
 - values 自增
 - type 自增类型 ASSIGN_ID
 - IdType.AUTO 自增
 - IdType.INPUT 手动设置
 - IdType.ASSIGN_ID 手动设置
- @TableField 字段注解
 - values 字段值
 - 多个 @TableField
 - 自增
 - 是否是主键
 - 字段名 `order`
 - 存在 = false

```
@TableName("tb_user")  
public class User{  
    @TableId(type=IdType.ASSIGN_ID)  
    private long id;  
  
    private String name;  
  
    private Boolean isMarried;  
    @TableField("`order`")  
    private Integer order;  
    @TableField(exist=false)
```

```
    private String address;  
}
```

映射文件

Mp 映射文件where

Wrapper 通过Wrapper实现SQL语句的封装 where 语句

- QueryWrapper 实现语句封装
 - .select() 选择语句
 - .like() 模糊查询语句
 - .ge() 大于等于语句
- UpdateWrapper 实现更新语句 where 和 set 语句

测试

```
// UserMapper  
public interface UserMapper extends BaseMapper<User>{  
  
}  
  
@Autowired  
private UserMapper userMapper;  
  
void testQueryMapper(){  
    // 1. 构造语句  
    QueryWrapper<User> wrapper = new QueryWrapper<User>()  
        .select("id", "username", "info", "balance")  
        .like("username", "o")  
        .ge("balance", 1000);  
  
    // 2. 执行  
    List<User> users = userMapper.selectList(wrapper);  
    users.forEach(user-> System.out.println(user));  
}  
  
void testUpdateByQueryWrapper(){  
    // 1. 构造对象  
    User user = new User();  
    user.setBalance(2000);  
    // 2. 构造语句  
    QueryWrapper<User> wrapper = new QueryWrapper<User>.eq("username", "jack");  
    // 3. 执行  
    userMapper.update(user,wrapper);  
}  
  
void testUpdateWrapper(){  
    List <long> ids = List.of(1L, 2L, 4L);  
    UpdateWrapper<User> wrapper = new UpdateWrapper<User>()  
        .setSql("balance = balance - 200") // update语句set语句
```

```

    .in("id",ids);

    // ...
    userMapper.update(null,wrapper);
}

```

Lambda

- `LambdaQueryWrapper` function
 - `User::getUsername`

```

void testLambdaQueryMapper(){
    // 1. Lambda
    LambdaQueryWrapper<User> wrapper = new LambdaQueryWrapper<User>()
        .select(User::getId,User::getUsername,User::getInfo,User::getBalance) //...
    .like(User::getUsername,"o")
    .ge(User::getBalance,1000);

    // 2. 打印
    List<User> users = userMapper.selectList(wrapper);
    users.forEach(user-> System.out.println(user));
}

```

自定义SQL

MP Mapper 定义自定义where语句的SQL语句

Mapper 定义SQL语句的Service 定义SQL语句

1. 定义Mapper Wrapper 定义 where 语句

```

List <long> ids = List.of(1L,2L,4L);
int amount = 200;
// 1. 定义语句
LambdaQueryWrapper<User> wrapper = new LambdaQueryWrapper<User>
().in(User::getId,ids);

// 2. 定义SQL语句
userMapper.updateBalanceByIds(wrapper,amount);

```

2. 定义 mapper 定义参数 Param 定义 wrapper 定义实体类 ew

```

void updateBalancenByIds(@Param("ew") LambdaQueryWrapper<User>
wrapper,@Param("amount") int amount);

```

3. 定义SQL语句Wrapper

```

<update id="updateBalanceByIds">
    UPDATE tb_user

```

```
    SET balance = balance - #{amount}
    ${ew.customSqlSegment} <!-- MySQL的自定义SQL-->
</update>
```

Service

接口 IService

- save
- remove
- update
- get

Service 和 Mapper 一样 service 有三个

实现类

- 实现 IService 接口
- 定义 ServiceImpl 为实现类
- 定义抽象类或接口



```
// 服务
// UserMapper和User
@Service
public class UserServiceImpl extends ServiceImpl<UserMapper,User> implements
UserService{

}

// 接口
public interface UserService extends IService<User>{

}

// 针对
@.Autowired
private UserService userService;
// userService
```

对象模型

- DTO
- VO
- PO

构造函数

- 带 @RequiredArgsConstructor 的构造函数
- 带 final 的构造函数

```
@RequestMapping("/users")
@RequiredArgsConstructor
```

```

public class UserController{
    private final IUserService userService;

    @DeleteMapping("{id}")
    public void deleteUserById(@PathVariable("id") Long id){
        /*
        *
        */
    }
}

```

IService@Lambda@

• IQuery、IUpdate、IDelete

IService 実装

- lambdaQuery() 実装
 - condition 実装
 - .list() も .one() 実装
 - query() 実装
- lambdaUpadte() 実装
 - .update() 実装
 - .update() 実装

```

// 実装
List<User> list = lambdaQuery()
    .like(name != null,User::getName,name)
    .eq(status != null,User::getStatus,status)
    .ge(minBalance != null,User::getBalance,minBalance)
    .le(maxBalance != null,User::getBalance,maxBalance)
    .list();

// 実装
lambdaUpdate()
    .set(User::getBalance,remainBalance)
    .set(remainBalance == 0,User::getStatus,2)
    .eq(User::getId,id)
    .eq(User::getBalance,user.getBalance()) // 実装
    .update();

```

IService@MyBatis@

- Mp 実装
 - 1kSQLを1kSQLに変換
- MySQL 実装
 - rewriteBatchedStatements=true 実装
 - MySQLのsql 実装

実装

** MybatisPlus **

-
-
-
-



MP

MP

MP



MP

MP

application.yaml

```
mybatis-plus:  
  global-config:  
    db-config:  
      logic-delete-field: flag #  
      logic-delete-value: 1 #  
      logic-not-delete-value: 0 #
```

MP

User

```
private Integer status;
```

MP

MP

- @EnumValue MP
- @JsonValue SpringMVC

```
@Getter  
public enum UserStatus{  
  NORMAL(1,"正常"),  
  FREEZE(2,"冻结")  
;  
  @EnumValue  
  @JsonValue  
  private final int value;
```

```
private final String desc;

UserStatus(int value, String desc){
    this.value = value;
    this.desc = desc;
}

}
```

enum

```
mybatis-plus:
  configuration:
    default-enum-type-handler:
      com.baomidou.mybatisplus.core.handlers.MybatisEnumTypeHandler
```

JSON

json转java

注解

- @TableName @TableField autoResultMap = true
- json @TableField typeHandler = JacksonTypeHandler.class

```
@Data
@TableName(value = "user",autoResultMap = true)
public class User{
    private long id;
    private String name;

    @TableField(typeHandler = JacksonTypeHandler.class)
    private UserInfo info;
}

@Data
public class UserInfo{
    private Integer age;
    private String intro;
    private String gender;
}
```

注解

MP注解

```
@Configuration
public class MybatisConfig{

    @Bean
    public MybatisPlusInterceptor mybatisPlusInterceptor(){
```

```

    // ...
    MybatisPlusInterceptor interceptor = new MybatisPlusInterceptor();
    // ...
    PaginationInnerInterceptor pageInterceptor = new
    PaginationInnerInterceptor(DbType.MYSQL);
    pageInterceptor.setMaxLimit(1000L); // ...
    interceptor.addInnerInterceptor(pageInterceptor);
    return interceptor;
}
}

```

PageQuery

- PageQuery
- ** Page **
- MP₂Page
- .addOrder()
 - new OrderItem("余额" "balance")
 - true 顺序 false 反序
- .getTotal() 总数
- .getPages() 页数
- .getRecords() 记录数

```

@Test
void testPageQuery() {
    // 1. 准备
    int pageNo = 1, pageSize = 5;

    // 1.1 新建
    Page<User> page = Page.of(pageNo, pageSize);

    // 1.2 添加排序balance
    page.addOrder(new OrderItem("balance", false));

    // 1.3 执行
    Page<User> p = userService.page(page);

    // 2. 输出
    System.out.println("total=" + p.getTotal());

    // 3. 输出
    System.out.println("pages=" + p.getPages());

    // 4. 输出
    List<User> records = p.getRecords();
    records.forEach(System.out::println);
}

```

Page

Page<T>

```

@Data
public class PageQuery{
    private Integer pageNo;
    private Integer pageSize;
    private String sortBy;
    private Boolean isAsc;
}

@Data
public class UserQuery extends PageQuery{
    private String name;
    private Integer status;
}

```

四四四

```

public class PageDTO<T>{
    // ...
    private Integer total;
    // ...
    private Integer pages;
    // ...
    private List<T> list;
}

```

□ PageQuery 三四三MP四 Page □

```

@Data
public class PageQuery{
    private Integer pageNo = 1;
    private Integer pageSize = 5;
    private String sortBy;
    private Boolean isAsc = true;

    public <T> Page<T> toMyPage(OrderItem ...items){
        // ...
        Page<T> page = Page.of(pageNo,pageSize);
        // ...
        if(StrUtil.isNotBlank(sortBy)){
            page.addOrder(new OrderItem(sortBy,isAsc));
        }else if(items != null){
            page.addOrder(items);
        }
        return page;
    }
}

```

```
}
```

□ Page 旣定の PageDTO 旣

```
public class PageDTO<T>{
    // ...
    private Integer total;
    // ...
    private Integer pages;
    // ...
    private List<T> list;

    public static <P0,V0> PageDTO<V0> of(Page<P0> page,Function<P0,V0> converter){
        PageDTO<V0> dto = new PageDTO<>();
        // ...
        dto.setTotal(page.getTotal());
        // ...
        dto.setPages(page.getPages());
        // ...
        List<P0> records = page.getRecords();

        if(CollUtil.isEmpty(records)){
            dto.setList(Collections.emptyList());
            return dto;
        }
        // ...
        dto.setList(records.stream().map(convertor).collect(Collectors.toList()));
        return dto;
    }

}
```

□□□

□□□□□

- □□project
- Maven□□

□□□□

Spring□□□□□ RestTemplate □□□□□□□□□□ Http □□□□□□□□□□□□

- □□ RestTemplate □Spring□□

```
package com.hmall.cart;

import org.mybatis.spring.annotation.MapperScan;
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
```

```
import org.springframework.context.annotation.Bean;
import org.springframework.web.client.RestTemplate;

@MapperScan("com.hmall.cart.mapper")
@SpringBootApplication
public class CartApplication {
    public static void main(String[] args) {
        SpringApplication.run(CartApplication.class, args);
    }

    @Bean
    public RestTemplate restTemplate() {
        return new RestTemplate();
    }
}
```

-

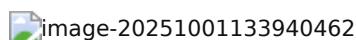
```
    ResponseEntity<List<ItemDT0>> response = restTemplate.exchange(
        "http://localhost:8081/items?ids={ids}",
        HttpMethod.GET,
        null,
        new ParameterizedTypeReference<List<ItemDT0>>() {
            },
            Map.of("ids", CollUtil.join(itemIds, ","))
    );
    if(!response.getStatusCode().is2xxSuccessful()){
        return;
    }
}

List<ItemDTO> items = response.getBody();
```

1

1

- 亂世の政治家としての才能と手腕
 - 亂世の政治家としての才能と手腕



Nacos

□□□□□□□□□□□□

1

- docker ip:8848/nacos nacos

- XML 配置文件与 YAML 配置文件

```
<!--nacos 配置-->
<dependency>
    <groupId>com.alibaba.cloud</groupId>
    <artifactId>spring-cloud-starter-alibaba-nacos-discovery</artifactId>
</dependency>
```

```
spring:
  application:
    name: item-service # 服务名
  cloud:
    nacos:
      server-addr: 192.168.150.101:8848 # nacos地址
```

三、Nacos

Nacos 是阿里巴巴开源的配置管理平台

- Nacos 架构
- Nacos API
- 客户端
 - DiscoveryClient 为 Spring 提供的服务发现功能

```
private final DiscoveryClient discoveryClient;

private void handleCartItems(List<CartVO> vos){
    // 1.从Nacos中拉取服务列表
    List<ServiceInstance> instances = discoveryClient.getInstances("item-
    service");
    // 2.从服务列表中随机选一个
    ServiceInstance instance =
    instances.get(RandomUtil.randomUUID());
    // 3.从IP列表中随机选一个
    URI uri = instance.getUri();
    // 4.调用
}
```

OpenFeign

SpringCloud http 客户端与 SpringMVC 客户端的区别在于 http 客户端

四、OpenFeign

- 定义接口
 - @FeignClient(** loadbalancer **)

```

<!--openFeign-->
<dependency>
    <groupId>org.springframework.cloud</groupId>
    <artifactId>spring-cloud-starter-openfeign</artifactId>
</dependency>
<!--网关-->
<dependency>
    <groupId>org.springframework.cloud</groupId>
    <artifactId>spring-cloud-starter-loadbalancer</artifactId>
</dependency>

```

- `@EnableFeignClients` `启动 OpenFeign`

```

@MapperScan("com.hmall.cart.mapper")
@EnableFeignClients
@SpringBootApplication
public class CartApplication {
    public static void main(String[] args) {
        SpringApplication.run(CartApplication.class, args);
    }
}

```

- `FeignClient`

```

// 客户端
@FeignClient("item-service")
public interface ItemClient {
    // URL
    @GetMapping("/items")
    List<ItemDTO> queryItemByIds(@RequestParam("ids") Collection<Long> ids);
}

```

- `FeignClient` `实现类`

```

List<ItemDTO> items = itemClient.queryItemByIds(List.of(1,2,3));

```

④

OpenFeign `Http` `URLConnection` `HttpClient` `Http` `OKHttp`

- `HttpURLConnection` `URLConnection`
- `Apache HttpClient` `HttpClient`
- `OKHttp` `OkHttp`

`URLConnection`

`HttpClient`

- `Http`

```
<!--OK http -->
<dependency>
    <groupId>io.github.openfeign</groupId>
    <artifactId>feign-okhttp</artifactId>
</dependency>
```

- OKHttp

```
feign:
  okhttp:
    enabled: true # OKHttp
```

OKHttp

OKHttp通过FeignClient实现SpringBootApplication FeignClient实现

- FeignClient 实现

```
@EnableFeignClients(basePackages="com.hmall.api.clients")
```

- FeignClient 实现

```
@EnableFeignClients(clients = UserClient.class)
```

OKHttp

OpenFeign通过 FeignClient 实现OKHttpDEBUG模式下400

- **NONE**不使用OKHttp
- **BASIC**通过URL+用户名+密码
- **HEADERS**通过BASIC+HTTP头部
- **FULL**通过OKHttp

OK Feign 不使用NONE模式

通过Logger.Level Bean实现

```
public class DefaultFeignConfig{
    @Bean
    public Logger.Level feignLogLevel(){
        return Logger.Level.FULL;
    }
}
```

- OKHttp @FeignClient 实现

```
@FeignClient(value = "item-service",configuration = DefaultFeignConfig.class)
```

- `@EnableFeignClients`

```
@EnableFeignClients(defaultConfiguration = DefaultFeignConfig.class)
```

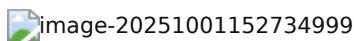
10 of 10

A decorative horizontal bar consisting of a series of small, evenly spaced rectangular blocks.

-
 - 

SpringCloud

- SpringCloudGateWay 『』
 - Netflix Zuul 『』



6

application.yaml


```
spring:
  application:
    name: gateway
  cloud:
    nacos:
      server-addr: 192.168.150.101:8848
    gateway:
      routes:
        - id: item # 项目id
          uri: lb://item-service # 负载均衡器
          predicates: # 预测子
            - Path=/items/**,/search/** # 路径匹配
        - id: cart
          uri: lb://cart-service
          predicates:
            - Path=/carts/**
```

1

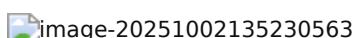
Java ** RouteDefinition **

- id ৰাখিবলৈ
 - uri ৰাখিবলৈ
 - predicates ৰাখিবলৈ
 - ৱিশেষ প্ৰক্ৰিয়া

- filters ကြည့်မှတ်ချက်များ
 - ၁၃၃ကြည့်မှတ်ချက်များ
 - ကြည့်မှတ်ချက် routes များ၏ default-filters များ

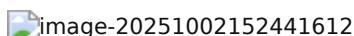
1

- **Java** **Socket**
 - **Java** **Server-Socket**
 - PRE **Java** **Server-Socket**
 - POST **Java** **Server-Socket**
 - **Java** **Server-Socket**** **Netty** **Server-Socket**** **Java** **Server-Socket**



- Http OpenFeign

1



1

1

- `GatewayFilter`
 - `GlobalFilter`

□□□□□□□□□□

1

```
@Override
public Mono<Void> filter(ServerWebExchange exchange, GatewayFilterChain chain) {

    // 1. 获取request
    ServerHttpRequest request = exchange.getRequest();
    // 2. 判断是否排除
    if(isExclude(request.getPath().toString())){
        return chain.filter(exchange);
    }
    // 3. 生成token
}
```

```

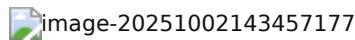
String token = null;
List<String> headers = request.getHeaders().get("authorization");
if(headers != null && !headers.isEmpty()){
    token = headers.get(0);
}
// 4.解析token
Long userId = null;
try{
    userId = jwtTool.parseToken(token);
} catch (UnauthorizedException e){
    // 401
    ServerHttpResponse response = exchange.getResponse();
    response.setStatusCode(HttpStatus.UNAUTHORIZED);
    return response.setComplete();
}

// TODO 5.打印userId
System.out.println("userId = " + userId);
// 6.返回
return chain.filter(exchange);
}

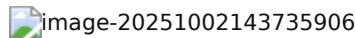
```

二、ThreadLocal

- 通过RequestWrapper实现ThreadLocal



- 通过RequestWrapper实现exchange.mutate() 实现ThreadLocal



- 在 common 实现 RequestWrapper 实现 ThreadLocal
 - SpringMvc 在 RequestWrapper 实现 common 实现 ThreadLocal
 - WebMvcConfigurer 实现 ThreadLocal
 - @ConditionalOnClass 实现 ThreadLocal

OpenFeign

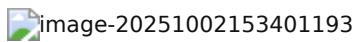
OpenFeign实现RequestInterceptor RequestInterceptor 实现OpenFeign

- RequestTemplate 实现RequestInterceptor
- RequestWrapperBean 实现RequestInterceptor
- RequestWrapper
 - @FeignClient 实现 RequestWrapper @EnableFeignClients 实现 RequestWrapper

四、

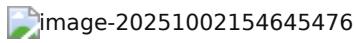
五、

- 实现RequestWrapper
- 实现RequestWrapper



██████

██████████



████nacos████████████████████████

- 『\${hm.db.port:3306}』 『3306』

██████████ bootstrap.yaml 『nacos』

- 『nacos』

```
<!--nacos-->
<dependency>
    <groupId>com.alibaba.cloud</groupId>
    <artifactId>spring-cloud-starter-alibaba-nacos-config</artifactId>
</dependency>
<!--bootstrap-->
<dependency>
    <groupId>org.springframework.cloud</groupId>
    <artifactId>spring-cloud-starter-bootstrap</artifactId>
</dependency>
```

- 『bootstrap.yaml』

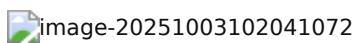
```
config:
    file-extension: yaml # 『nacos』
    shared-configs: # 『nacos』
        - dataId: shared-jdbc.yaml # 『mybatis』
        - dataId: shared-log.yaml # 『日志』
        - dataId: shared-swagger.yaml # 『Swagger』
```

██████

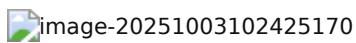
██

██████

- nacos『████████████████████████』
 - profile 『████████████████████』



- 『████████████████████████████████』

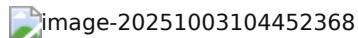


1

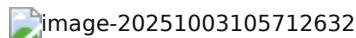
Nacos Nacos

1

- Nacos چیست و چه کاربردهایی دارد
 - Nacos چگونه کار می‌کند و آنرا با NacosConfigManager معرفی کنید

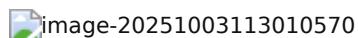


- `RouteDefinitionWriter` 類別
 - `yaml` `RouteDefinition` 記載檔
 - `Nacos` `RouteDefinition` 記載檔 `Nacos` json 記載檔
`RouteDefinition`
 - `RouteDefinitionWriter` `save` `delete` 記載檔 Mono `RouteDefinition`
`.subscribe()`

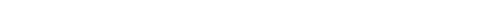


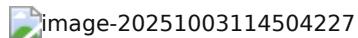
1

1

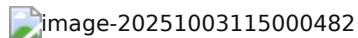


1

- 
 - 



- `font-family: sans-serif; font-weight: bold;`
 - `font-family: sans-serif; font-weight: bold; font-style: italic; color: red;` **fallback** `color`



Sentinel

5

- **SpringCloud容错降级神器Sentinel**开源项目
 - **Sentinel**与**SpringMVC**结合实现**HTTP**
 - **熔断器**

- RestfulAPI
◦ http-method-specified + yaml

```
http-method-specified: true
```

HTTP

HTTP+yaml

HTTP

HTTP+yaml

Fallback

- FeignClient + Sentinel + yaml

```
feign:  
  sentinel:  
    enabled: true
```

- Fallback
◦ FallbackClass
◦ FallbackFactory
 - FallbackFactory Client
 - create new Client
 - Bean
 - Client @FeignClient fallbackFactory class

HTTP

image-20251003150134978

HTTP

HTTP+yaml

- FallbackClass
- FallbackFactory

Seata

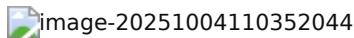
HTTP

image-20251003153535418

Seata

- TC-
◦ TM-

- RM-TC



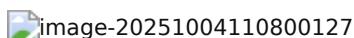
TC

- TC

```
<!-- 阿里云 -->
<dependency>
    <groupId>com.alibaba.cloud</groupId>
    <artifactId>spring-cloud-starter-alibaba-nacos-config</artifactId>
</dependency>
<!-- bootstrap -->
<dependency>
    <groupId>org.springframework.cloud</groupId>
    <artifactId>spring-cloud-starter-bootstrap</artifactId>
</dependency>
<!-- seata -->
<dependency>
    <groupId>com.alibaba.cloud</groupId>
    <artifactId>spring-cloud-starter-alibaba-seata</artifactId>
</dependency>
```

- Nacos TC

```
seata:
  registry: # TC
  type: nacos # 使用 nacos
  nacos:
    server-addr: 172.21.172.16:8848 # nacos
    namespace: "" # namespace
    group: DEFAULT_GROUP # 默认DEFAULT_GROUP
    application: seata-server # seata
    username: nacos
    password: nacos
  tx-service-group: hmall # 服务组
  service:
    vgroup-mapping: # 服务名
      hmall: "default"
```



Seata

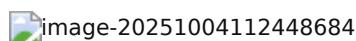
- XA
- AT

XA

X/Open TM/RM

1

- TM
◦ TMの構成要素TCとRM
 - TCの構成要素RM
 - RMの構成要素TC
 - RMの構成要素sql
 - データベースの構成要素
 - RMの構成要素TC
 - TC
◦ TMの構成要素TCとRM
 - TCの構成要素RM
 - RMの構成要素TC
 - データベースの構成要素



三

-                                                                                                                                                                    <img alt="Icon representing a large document or file" data-bbox

```
seata:  
    data-source-proxy-mode: XA
```

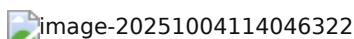

AT

Seata AT AT XA

1

- ①②③
◦ TM④⑤⑥⑦⑧⑨⑩TC⑪⑫
◦ ⑬TM⑭⑮⑯⑰RM
◦ RM⑯TC⑪⑫⑬⑭
◦ RM⑮⑯⑰⑪⑫
◦ RM⑮⑯⑰sql⑪⑫⑬

- RMTC
- AT
 - TMTC
 - TC
 - 亂碼轉換
 - TC轉RM



AT

- AT

AT

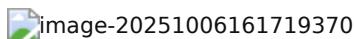
- .yaml 配置AT

```
seata:
  data-source-proxy-mode: AT
```

Elasticsearch

Lucene

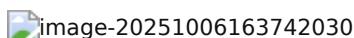
- document
- Restful
- kibana □ Logstash □ Beats 集成** ELK **



分析器

elasticsearch

- document
- term



IK

Elasticsearch □ Lucene 分词器 ik_smart ik_max_word

- ik_smart
- ik_max_word
 - config 为 IkAnalyzer.cfg.xml 包含(.dic)

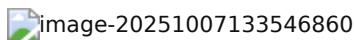
- Elasticsearch

Elasticsearch

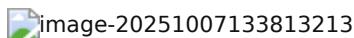
Elasticsearch json Elasticsearch

index

mapping



Mysql Elasticsearch



Kibana

Dev Tools

Elasticsearch API Restful

- Elasticsearch mapping

```
PUT /_index
{
  "mappings": {
    "properties": {
      "字段": {
        "type": "text",
        "analyzer": "ik_smart"
      },
      "字段2": {
        "type": "keyword",
        "index": "false"
      },
      "字段3": {
        "properties": {
          "子字段": {
            "type": "keyword"
          }
        }
      },
      // ...
    }
  }
}
```

- Elasticsearch

```
GET /_index
```

- _id, _score

```
DELETE /_index
```

- _id, _index, _score, _score, _typemapping

```
PUT /_index/_mapping
```

```
{  
  "properties": {  
    "id": {  
      "type": "integer"  
    }  
  }  
}
```

Mapping

mapping

- type 文档类型
 - text 文本 keyword 词典
 - long 整数 integer 短整数 short 字节 byte 双精度浮点数 double 浮点数 float
 - 布尔 boolean
 - 日期 date
 - 对象 object
- index 是否索引 true
- analyzer 分析器
- properties 属性

Document

- _id, _index, _score

```
POST /_index/_doc/_id
```

```
{  
  "id": "1",  
  "id": "2",  
  "id": {  
    "id1": "3",  
    "id2": "4"  
  },  
  // ...  
}
```

- ドキュメント

```
GET /{index}/_doc/{id}
```

- ドキュメント

```
DELETE /{index}/_doc/{id}
```

- ドキュメント

- ドキュメントを更新する

```
PUT /{index}/_doc/{id}
{
  "field1": "value1",
  "field2": "value2",
  // ...
}
```

- ドキュメントのidを更新する

```
POST /{index}/_update/{id}
{
  "doc": {
    "field": "value"
  }
}
```

- ドキュメント

- index ドキュメント
 - _index ドキュメント
 - _id ドキュメントid
 - { "field1" : "value1" } ドキュメント
 - delete ドキュメント
 - _index ドキュメント
 - _id ドキュメントid
 - update ドキュメント
 - _index ドキュメント
 - _id ドキュメントid
 - { "doc" : { "field2" : "value2" } } ドキュメント

```
POST _bulk
{ "index" : { "_index" : "test", "_id" : "1" } }
{ "field1" : "value1" }
{ "delete" : { "_index" : "test", "_id" : "2" } }
{ "create" : { "_index" : "test", "_id" : "3" } }
```

```
{ "field1" : "value3" }
{ "update" : { "_id" : "1", "_index" : "test" } }
{ "doc" : { "field2" : "value2" } }
```

JavaRestClient

Elasticsearch

- Elasticsearch

```
<dependency>
    <groupId>org.elasticsearch.client</groupId>
    <artifactId>elasticsearch-rest-high-level-client</artifactId>
</dependency>
```

- Elasticsearch RestHighLevelClient
 - HttpHost.create IP주소

```
RestHighLevelClient restHighLevelClient = new
RestHighLevelClient(RestClient.builder(
    HttpHost.create("172.21.172.16:9200")
));
```

CreateIndexRequest

DeleteIndexRequest

- CreateIndexRequest request
 - source
- request.source() source
- .indices() indices
 - .create() Create
 - request request

```
@Test
void testCreateIndex() throws IOException {
    // 1. Request
    CreateIndexRequest request = new CreateIndexRequest("items");
    // 2. Source
    request.source(MAPPING_TEMPLATE, XContentType.JSON);
    // 3. Indices
    client.indices().create(request, RequestOptions.DEFAULT);
}
```

DeleteIndexRequest

- DeleteIndexRequest request
 - source

- .indices() メソッド

- .delete() メソッド
 - request パラメータ

```
@Test
void testDeleteIndex() throws IOException {
    // 1. Request
    DeleteIndexRequest request = new DeleteIndexRequest("items");
    // 2. API
    client.indices().delete(request, RequestOptions.DEFAULT);
}
```

delete API

- GetIndexRequest パラメータ request パラメータ
 - .source パラメータ
- .indices() メソッド
 - .get() メソッド .exists() メソッド
 - request パラメータ

```
@Test
void testExistsIndex() throws IOException {
    // 1. Request
    GetIndexRequest request = new GetIndexRequest("items");
    // 2. API
    client.indices().get(request, RequestOptions.DEFAULT);
}
```

exists API

exists API

- IndexRequest パラメータ Request パラメータ .id() パラメータ id
- .source() パラメータ
- .index() パラメータ

```
@Test
void testIndexDoc() throws IOException {
    // テスト用データ
    Item item = iItemService.getById(100000011127L);
    ItemDoc itemDoc = BeanUtil.copyProperties(item, ItemDoc.class);

    IndexRequest request = new IndexRequest("items").id(itemDoc.getId());
    request.source(JSONUtil.toJsonStr(itemDoc), XContentType.JSON);
```

```
    restHighLevelClient.index(request, RequestOptions.DEFAULT);  
}
```

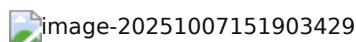
1

- `DeleteRequest` `Request` `id`
 - `.delete()`

```
@Test
void testDeleteDocument() throws IOException {
    // 1. 创建请求对象并设置唯一标识符
    DeleteRequest request = new DeleteRequest("item", "100002644680");
    // 2. 调用
    client.delete(request, RequestOptions.DEFAULT);
}
```

1

- `GetRequest` `Request` `id`
 - `.get()` `GetResponse`
 - `.getSourceAsString()` `_source`



```
@Test
void testGetDocumentById() throws IOException {
    // 1. 创建请求
    GetRequest request = new GetRequest("items").id("100002644680");
    // 2. 执行
    GetResponse response = client.get(request, RequestOptions.DEFAULT);
    // 3. 读取响应源
    String json = response.getSourceAsString();

    ItemDoc itemDoc = JSONUtil.toBean(json, ItemDoc.class);
    System.out.println("itemDoc= " + itemDoc);
}
```

1

```
@Test
void testUpdateDocument() throws IOException {
    // 1.准备Request
    UpdateRequest request = new UpdateRequest("items", "100002644680");
    // 2.设置请求体
    request.doc(
        "price", 58800,
        "commentCount", 1
    );
    // 3.发送请求
    client.update(request, RequestOptions.DEFAULT);
}
```

1

- `BulkRequest` 用于执行CRUD操作的 request 对象
 - `.add()` 方法添加操作
◦ `IndexRequest`
 - `.bulk()` 执行 bulk 操作

```
@Test
void testBulk() throws IOException {
    // 1.准备Request
    BulkRequest request = new BulkRequest();
    // 2.准备请求体
    request.add(new IndexRequest("items").id("1").source("json doc1",
XContentType.JSON));
    request.add(new IndexRequest("items").id("2").source("json doc2",
XContentType.JSON));
    // 3.发送请求
    client.bulk(request, RequestOptions.DEFAULT);
}
```

DSL

ISO 9001

DSI □□□□□□□□

- 二〇二〇年九月三十日
 - 二〇二〇年十月一日

A horizontal row of 15 small squares, likely representing a scale or a series of items.

- `SELECT 1` မှတ်ယူနေရန်
 - `SELECT from size` မှတ်ယူနေ MySQL
 - `SELECT` မှတ်ယူနေရန်

- `query`

•••DSL•••••

```
GET /{index}/_search
{
  "query": {
    "term": {
      // ...
    }
  }
}
```

•••••

- `response`
 - `took` `毫秒数`
 - `time_out` `毫秒数`
 - `_shards` `信息`
 - `hits` `信息`
 - `total` `总条数`
 - `max_score` `最高分`
 - `hits` `结果`

```
GET /items/_search
{
  "query": {
    "match_all": {}
  }
}
```

•••••

- `term`
 - `match_query`
 - `multi_match_query`
- `range`
 - `ids` `id`
 - `range` `范围`
 - `term` `词`
- `script`

match ••••• •••••

```
GET /{index}/_search
{
  "query": {
    "match": {
      ...
    }
  }
}
```

```
        "query": "测试"
    }
}
}
```

multi_match

```
GET /{index}/_search
{
  "query": {
    "multi_match": {
      "query": "测试",
      "fields": ["字段1", "字段2"]
    }
  }
}
```

term

```
GET /{index}/_search
{
  "query": {
    "term": {
      "字段名": {
        "value": "测试"
      }
    }
  }
}
```

range

- gte 测试开始 gt 测试结束
- lte 测试开始 lt 测试结束

```
GET /{index}/_search
{
  "query": {
    "range": {
      "字段名": {
        "gte": {值},
        "lte": {值}
      }
    }
  }
}
```

注意

范围查询

- `bool`
- `function_score`
- `dis_max`

• `must` `should` `must_not` `filter`

- `must` `must`
- `should` `should`
- `must_not` `must_not`
- `filter` `filter`

```
GET /items/_search
{
  "query": {
    "bool": {
      "must": [
        {"match": {"name": "华为"}}
      ],
      "should": [
        {"term": {"brand": { "value": "vivo" }}},
        {"term": {"brand": { "value": "荣耀" }}}
      ],
      "must_not": [
        {"range": {"price": {"gte": 2500}}}
      ],
      "filter": [
        {"range": {"price": {"lte": 1000}}}
      ]
    }
  }
}
```

• `script` `script_score`

• `geo_point`

• `geo_shape`

- elasticsearch`score` `script_score` `script` `geo_point` `geo_shape` `range` `function_score` `function` `function_score` `function`
- `highlight`

```
GET /indexName/_search
{
  "query": {
    "match_all": {}
  },
  "sort": [
```

```
{
  {
    "field1": {
      "order": "asc|desc"
    }
  },
  {
    "field2": {
      "order": "asc|desc"
    }
  }
}
]
```

结果

- top10
- from size
 - from
 - size
 - from + size 10000

```
GET /items/_search
{
  "query": {
    "match_all": {}
  },
  "from": 0, // 从第0个开始
  "size": 10, // 取10个
  "sort": [
    {
      "price": {
        "order": "desc"
      }
    }
  ]
}
```

结果

- es** search after **
- 从第10个开始取10个

结果

从第10个开始取10个

注意

- 从第10个开始取10个 em

```

GET /{index}/_search
{
  "query": {
    "match": {
      "title": "毛泽东"
    }
  },
  "highlight": {
    "fields": {
      "content": {
        "pre_tags": "<em>",
        "post_tags": "</em>"
      }
    }
  }
}

```

聚合

TermAggregation Date Histogram

- Count
- TermAggregation Date Histogram
- Avg Max Min Status max,min,avg,sum
- Total

聚合

- Aggs
 - aggs

```

GET /items/_search
{
  "query": {
    "match_all": {}
  }, //查询
  "size": 0, //size 0 表示不返回结果
  "aggs": { //聚合
    "cateAgg": { //category聚合
      "terms": { //term聚合
        "field": "category", //category
        "size": 20 // 20个结果
      }
    }0
  }
}

```

}

- 
 - 

```
GET /items/_search
{
  "query": {
    "bool": {
      "filter": [
        {
          "term": {
            "category": "服饰"
          }
        },
        {
          "range": {
            "price": {
              "gte": 300000
            }
          }
        }
      ]
    }
  },
  "size": 0,
  "aggs": {
    "brand_agg": {
      "terms": {
        "field": "brand",
        "size": 20
      },
      "aggs": {
        "stats_meric": {
          "stats": {
            "field": "price"
          }
        }
      }
    }
  }
}
```

JavaRestClient

1

- `SearchRequest` သည် `request` ဖြစ်
 - `source`
 - `request.source()` ဖြစ်ပေါ်လိမ့်မည်
 - `request.query()` ဖြစ်ပေါ်လိမ့်မည်

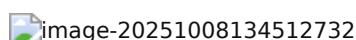
- `QueryBuilders` `...`
 - `matchAllQuery()` `...`
 - `.search()` `...`
 - `request` `...`

```
@Test  
void testMatchAll() throws IOException{  
    // 1.构建request  
    SearchRequest request = new SearchRequest();  
    // 2.构建DSL语句  
    request.source()  
        .query(QueryBuilder().matchAllQuery());  
    // 3.执行请求  
    client.search(request,Req)  
}
```

Query QueryBuilders

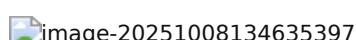
- `QueryBuilders` API
 - `QueryBuilders.MatchQuery` សោរអាជីវកម្ម
 - `QueryBuilders.multiMatchQuery` សោរអាជីវកម្ម

```
// 指定条件  
QueryBuilders.MatchQuery("name", "猫")  
  
// 多条件  
QueryBuilders.multiMatchQuery("猫", "name", "category");
```



- `org.elasticsearch.index.query` API
 - `QueryBuilders.termQuery` 词语查询
 - `QueryBuilders.rangeQuery` 范围查询

```
// 指定条件  
QueryBuilders.termQuery("category", "服饰");  
// 指定范围  
QueryBuilders.rangeQuery("price").gte(100).lte(150);
```



- `BoolQueryBuilder` `QueryBuilders.boolQuery` `must`
 - `.must()` `must`

```
// 逻辑查询语句
BoolQueryBuilder boolQuery = QueryBuilders.boolQuery();
// must
boolQuery.must(
        QueryBuilders.termQuery("brand", "华为"));
// should
boolQuery.should(
        QueryBuilders.rangeQuery("price").lte(2500));
```

request.source() 00

- `.from()` \square `.size()` `java.util.List`
 - `.sort()` `java.util`
 - `SortOrder` `java.util`
 - `Comparator` `java.util`

```
// ...
request.source().from(0).size(5);
// ...
request.source().sort("price",SortOrder.ASC);
```

```
    request.source()
```

- `.highlighter()` `highlight`
 - `SearchSourceBuilder.highlight()` `highlight`

```
request.source().highlighter(  
    SearchSourceBuilder.highlight()  
        .field("name")  
        .preTags("〈em〉")  
        .postTags("〈/em〉")  
);
```

-

```
private static void parseResponseResult(SearchResponse search) {
    SearchHits hits = search.getHits();

    long total = hits.getTotalHits().value;

    SearchHit[] searchHits = hits.getHits();

    for (SearchHit searchHit : searchHits) {
        String json = searchHit.getSourceAsString();

        // ██████
    }
}
```

```
Map<String, HighlightField> hfs = searchHit.getHighlightFields();
if(hfs!=null && !hfs.isEmpty()){
    HighlightField hf = hfs.get("name");
    json = hf.getFragments()[0].string();
}
System.out.println("json"+json);
}
```

1

```
    request.source()
```

- `.aggregation()` `AggregationBuilders`
 - `AggregationBuilders`

```
request.source().aggregation(  
    AggregationBuilders  
        .terms("brand_agg")      // گروه‌بندی بر اساس نام برنده  
        .field("brand")           // فیلد نام برنده  
        .size(20)                 // تعداد 20 نتایج  
);
```

-

```
@Test
void testAgg() throws IOException {
    SearchRequest request = new SearchRequest("items");
    request.source().size(0);

    String brandAggName = "brandAgg";
    request.source().aggregation(
        AggregationBuilders
            .terms(brandAggName)
            .field("brand")
            .size(10)
    );
    SearchResponse search = restHighLevelClient.search(request,
    RequestOptions.DEFAULT);
    System.out.println("search results:" + search);

    Aggregations aggregations = search.getAggregations();
    Terms brandTerms = aggregations.get(brandAggName);
    List<? extends Terms.Bucket> buckets = brandTerms.getBuckets()
```

```
        for (Terms.Bucket bucket : buckets) {  
            System.out.println("brand:"+bucket.getKey());  
            System.out.println("count:"+bucket.getDocCount());  
        }  
    }  
}
```

████████

████████

██

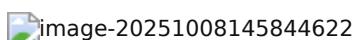
- **CP**██
- **XA**████
- **AP**██
- **AT**████

CAP████

████████████████████

- Consistency ████████
- Availability ████████
- Partition tolerance ██████████

████████████████████████████████████



Base████

BASE████CAP████████████████████████

- **Basically Available** ███
- **Soft State** ███
- **Eventually Consistent** ███

AT████████████