8 - 图片功能扩展 - 智能协同云图库项目教程 - 编程导航教程

> 初始化环境 1、[配置 Maven](https://www.codefather.cn/post/1836689783992958977)2、[安装 Java8](https://www.codefa。

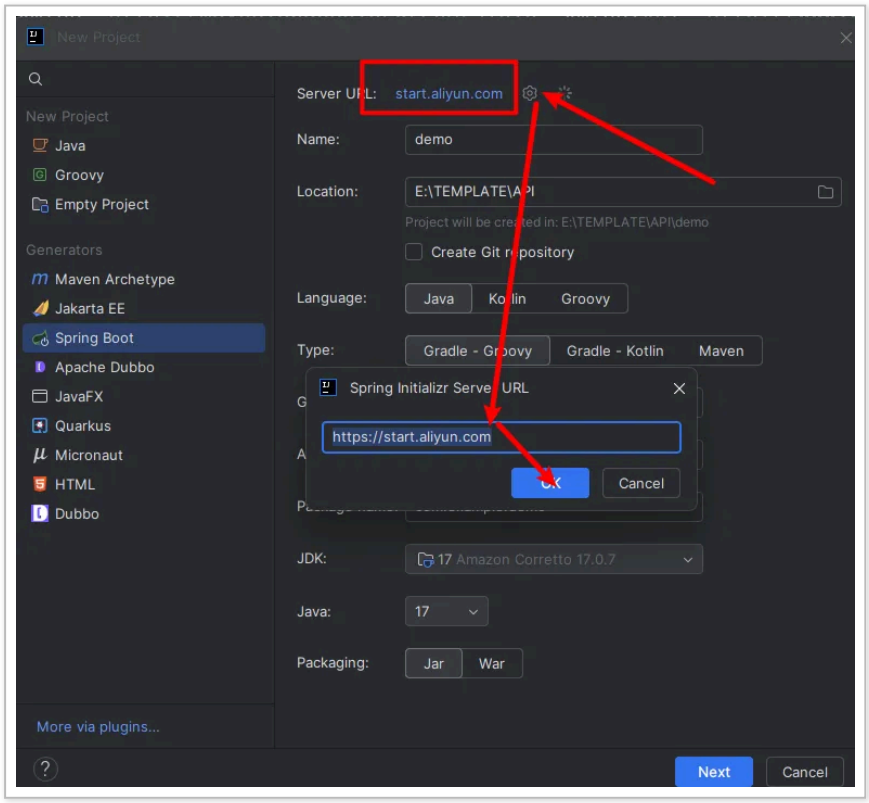
1、 配置 Maven

2、 安装 Java8

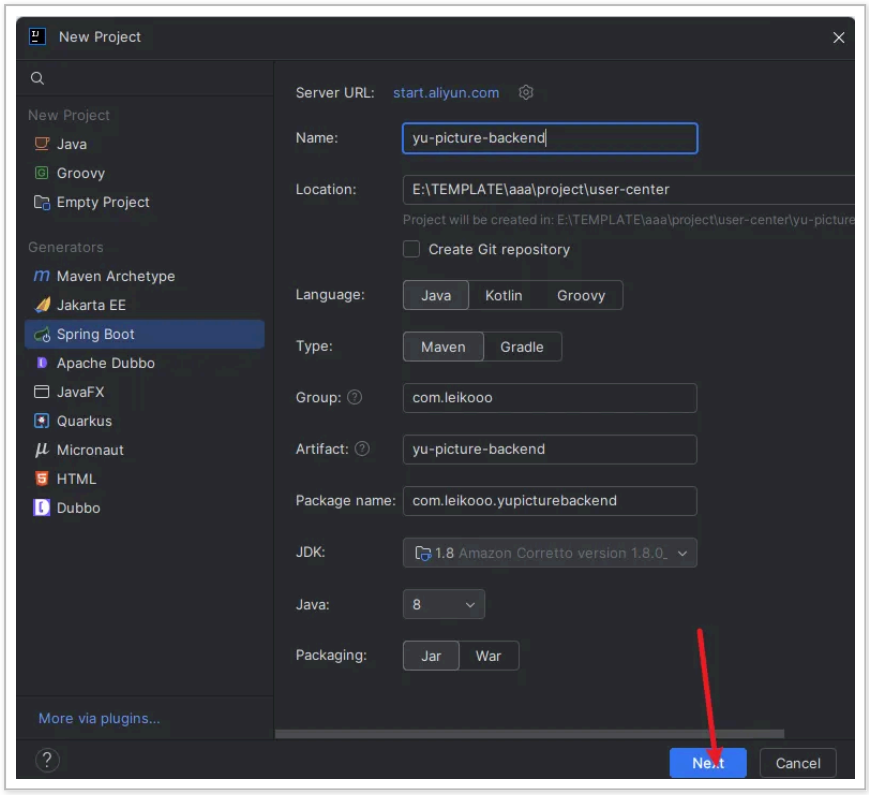
3、安装 MySQL8 msi 安装包 链接: https://pan.baidu.com/s/1O6TrRCpb66A5hdgy0EY9HA 提取码: g17s

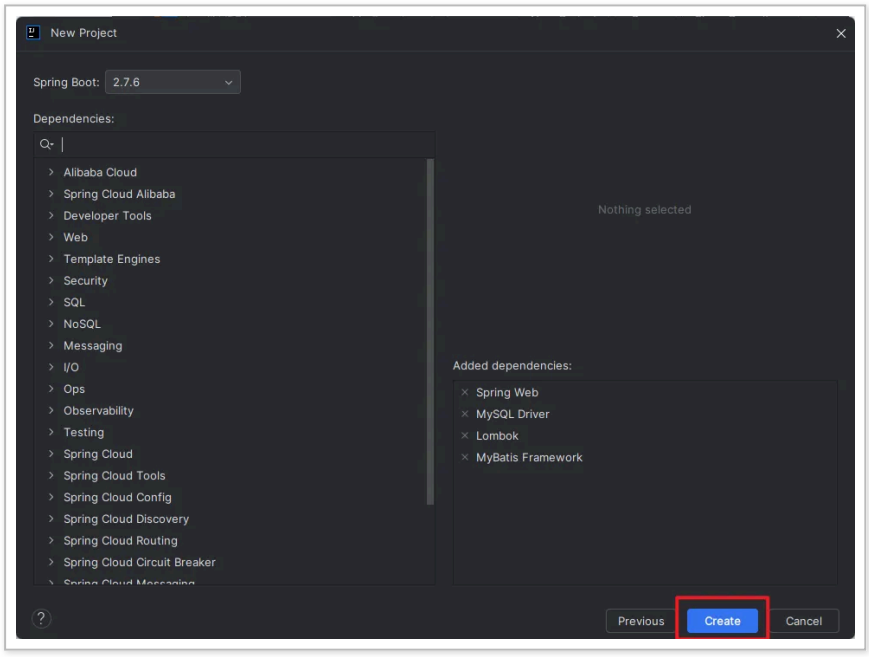
MySQL 安装教程 并且确定 MySQL 启动状态

1、设置阿里云的 Spring Initializr https://start.aliyun.com/

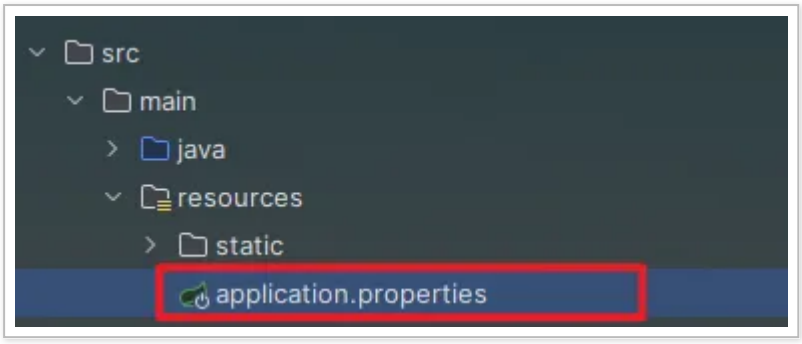


2、选择 `2.7.6` 版本





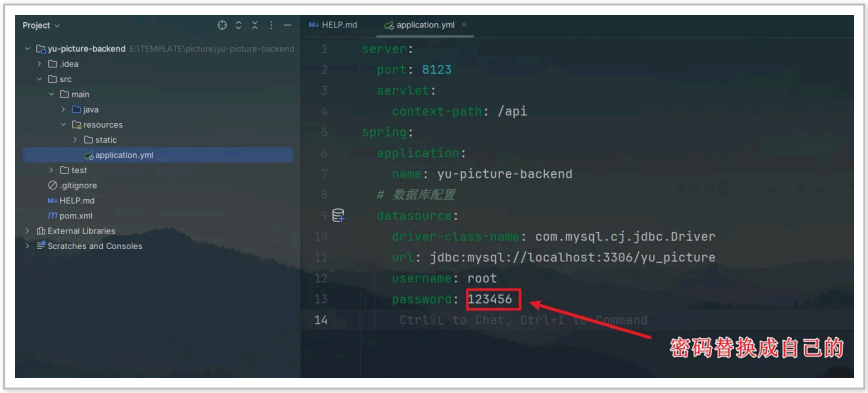
3、修改配置文件 `application.properties`



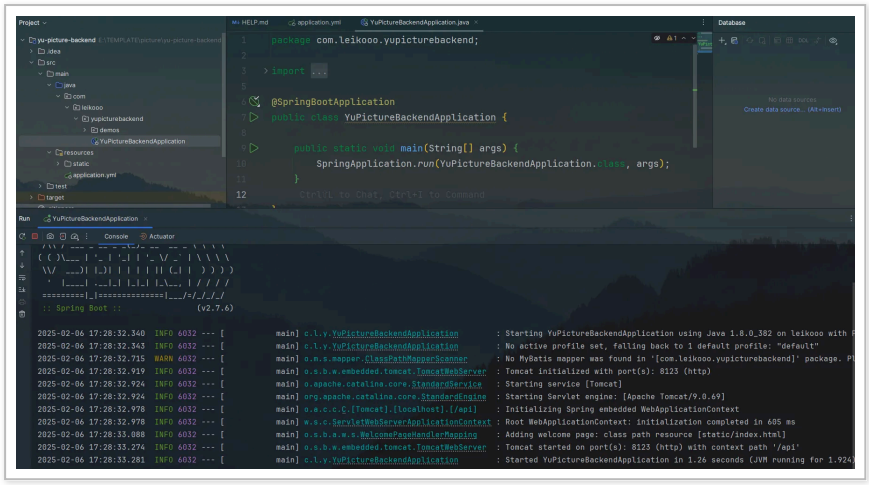
修改后缀为 yml 格式，把名字改成 `application.yml` ，并且添加以下内容

```yaml
server:
  port: 8123
  servlet:
    context-path: /api
spring:
  application:
    name: yu-picture-backend

  datasource:
    driver-class-name: com.mysql.cj.jdbc.Driver
    url: jdbc:mysql://localhost:3306/yu_picture
    username: root
    password: 123456
```



4、尝试启动看看是否报错

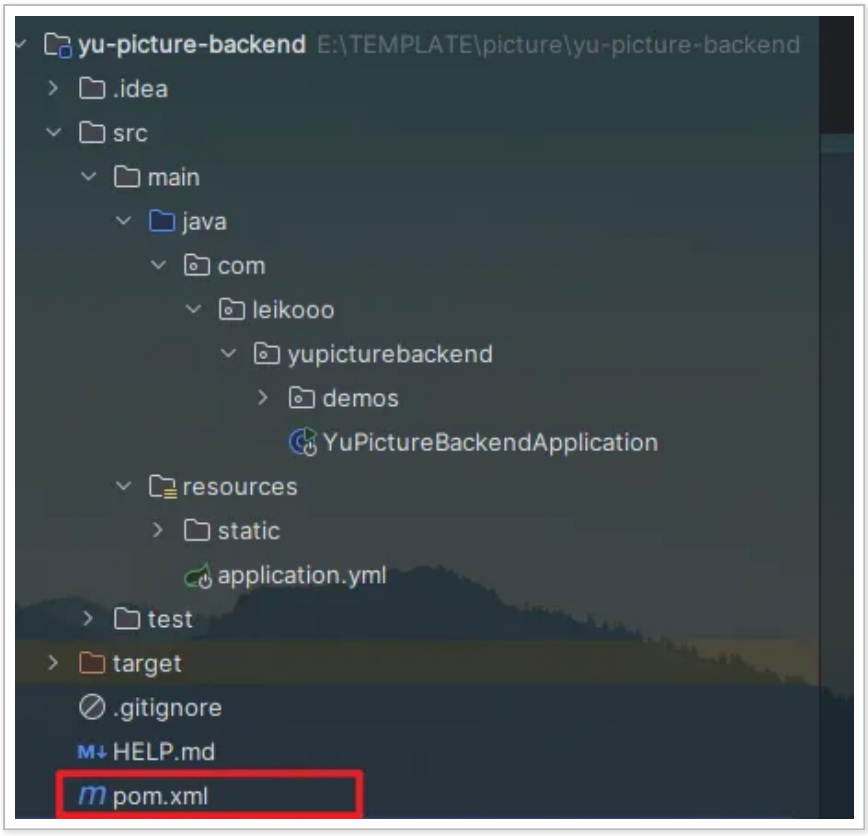


MyBatis-plus

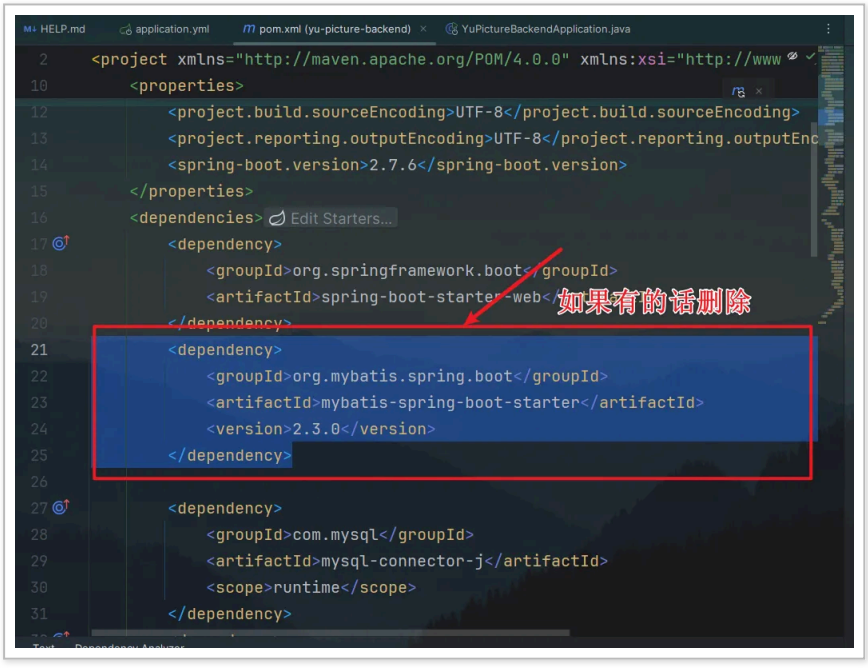
官网： https://baomidou.com/

1、在 pom.xml 添加下面的依赖并且删除 MyBatis 的依赖防止依赖冲突

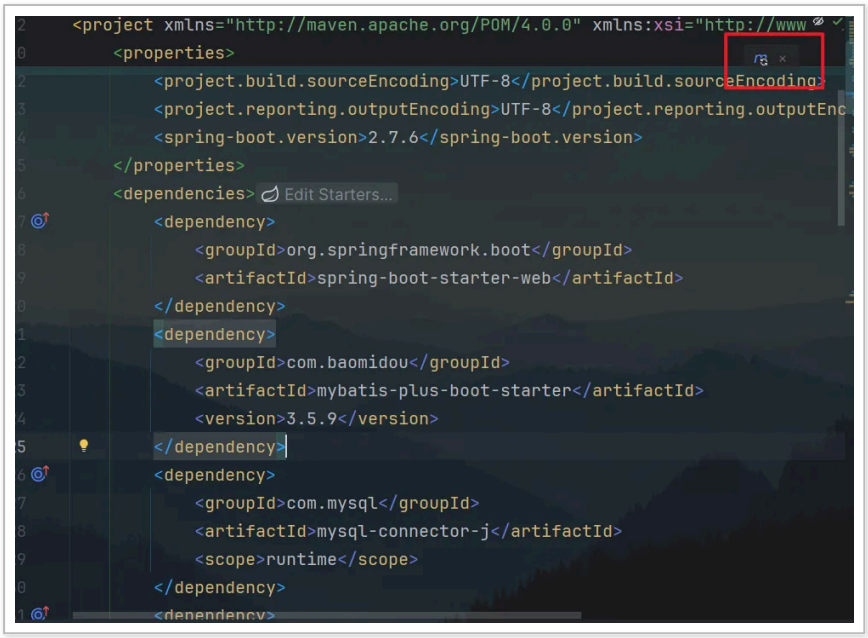
```
<dependency>
    <groupId>com.baomidou</groupId>
    <artifactId>mybatis-plus-boot-starter</artifactId>
    <version>3.5.9</version>
</dependency>
```



如果有相关 MyBatis 的依赖需要删除



2、添加到 pom.xml 之后，点击右上角的 `Load Maven Changes`



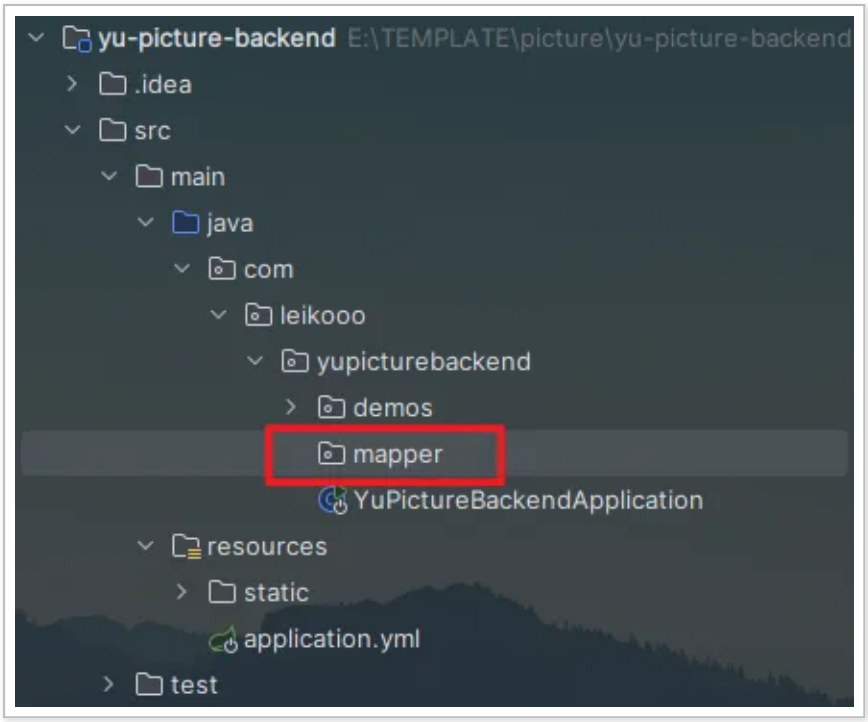
3、在启动文件添加 `@MapperScan`

```
@SpringBootApplication
@MapperScan("com.leikooo.yupicturebackend.mapper")
public class YuPictureBackendApplication {

    public static void main(String[] args) {
        SpringApplication.run(YuPictureBackendApplication.class, args
    }

}
```

然后在 `com.leikooo.yupicturebackend` 下面创建 `mapper` 文件夹



Hutool 工具类

官网： https://hutool.cn

1、在 pom.xml 添加下面的依赖

```
<dependency>
    <groupId>cn.hutool</groupId>
    <artifactId>hutool-all</artifactId>
    <version>5.8.26</version>
</dependency>
```

2、点击右上角的 `Load Maven Changes`

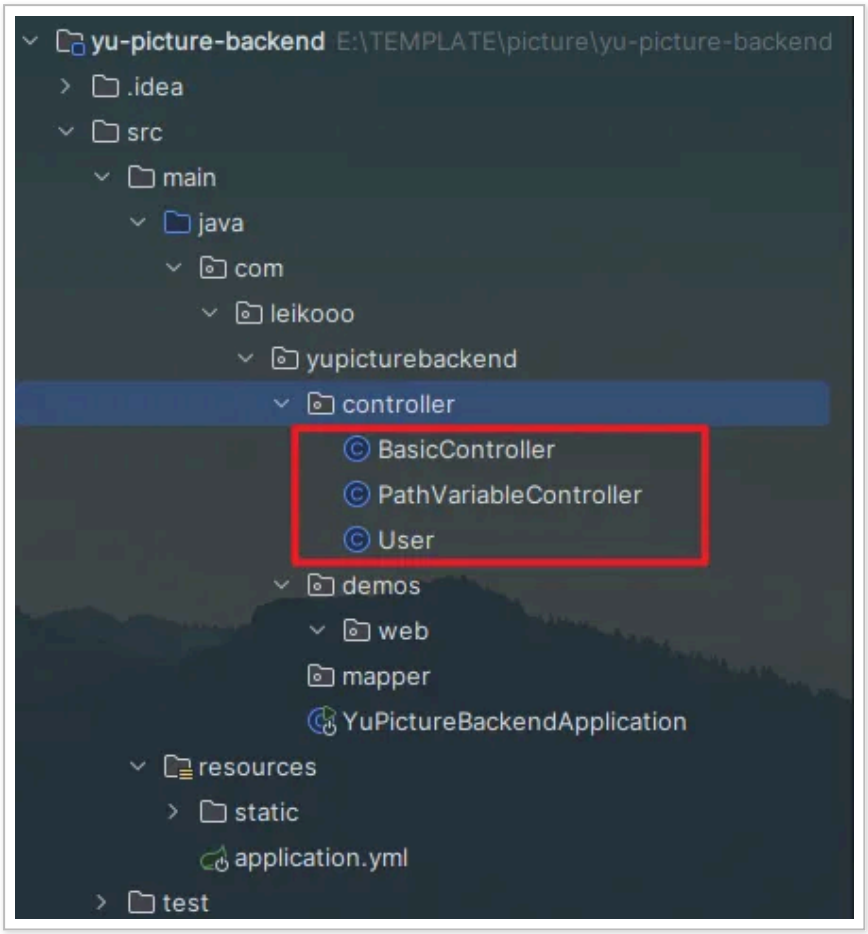
Knife4j

官网： https://doc.xiaominfo.com/docs/quick-start#spring-boot-2

1、在 pom.xml 添加下面的依赖

```
<dependency>
    <groupId>com.github.xiaoymin</groupId>
    <artifactId>knife4j-openapi2-spring-boot-starter</artifactId>
    <version>4.4.0</version>
</dependency>
```

2、新建 controller 包，然后在 `demos.web` 下面的东西

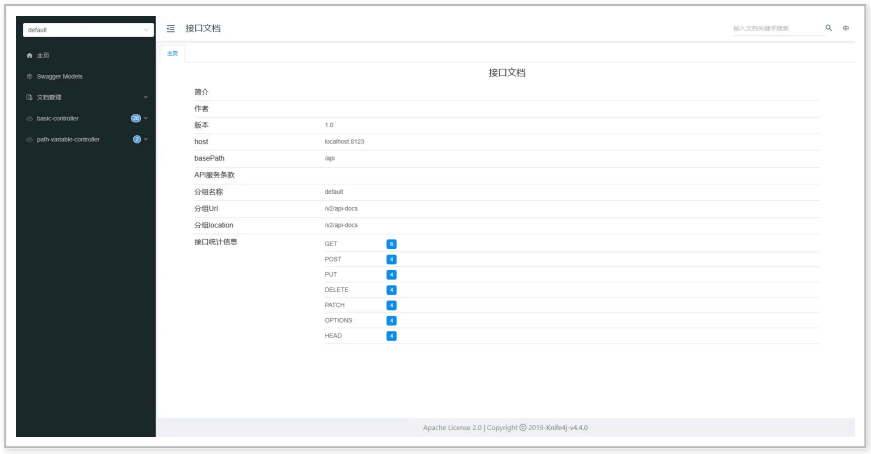


3、在 application.yml 添加以下依赖

注意下面的 `group` 是 `default`

```
knife4j:
  enable: true
  openapi:
    title: "接口文档"
    version: 1.0
    group:
      default:
        api-rule: package
        api-rule-resources:
          - com.leikooo.yupicturebackend.controller
```

访问 http://localhost:8123/api/doc.html 可以看到下面的接口文档



其他依赖

1、aop 切面

```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-aop</artifactId>
</dependency>
```

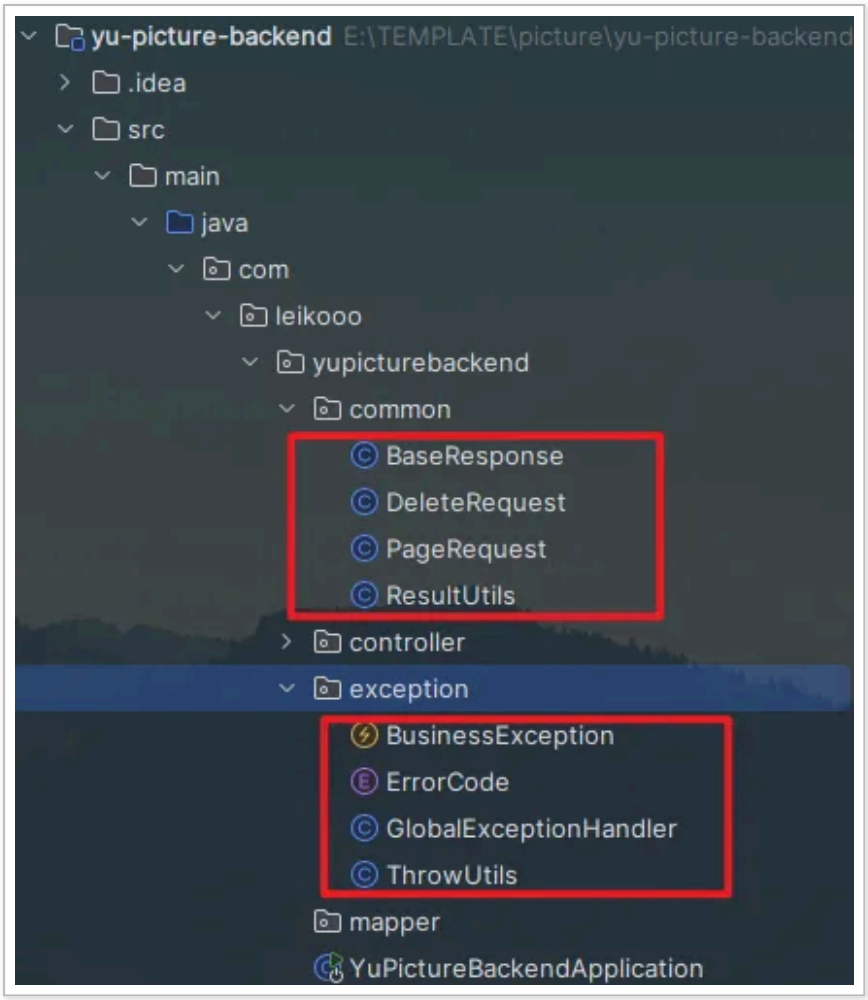
2、可以在主类上加上

```
@EnableAspectJAutoProxy(exposeProxy = true)
```

加上之后可以在代码之中使用， 获取当前类的代理对象

```
AopContext.currentProxy()
```

具体代码的位置



自定义异常类

`exception` 包下面

异常枚举类

```
@Getter
public enum ErrorCode {

    SUCCESS(0, "ok"),
    PARAMS_ERROR(40000, "请求参数错误"),
    NOT_LOGIN_ERROR(40100, "未登录"),
    NO_AUTH_ERROR(40101, "无权限"),
    NOT_FOUND_ERROR(40400, "请求数据不存在"),
    FORBIDDEN_ERROR(40300, "禁止访问"),
    SYSTEM_ERROR(50000, "系统内部异常"),
    OPERATION_ERROR(50001, "操作失败");


    private final int code;


    private final String message;

    ErrorCode(int code, String message) {
        this.code = code;
```

```
        this.message = message;
    }

}
```

自定义业务异常

```
@Getter
public class BusinessException extends RuntimeException {


    private final int code;

    public BusinessException(int code, String message) {
        super(message);
        this.code = code;
    }

    public BusinessException(ErrorCode errorCode) {
        super(errorCode.getMessage());
        this.code = errorCode.getCode();
    }

    public BusinessException(ErrorCode errorCode, String message) {
        super(message);
        this.code = errorCode.getCode();
    }

}
```

为了方便抛出异常

```
public class ThrowUtils {


    public static void throwIf(boolean condition, RuntimeException ru
        if (condition) {
            throw runtimeException;
        }
    }


    public static void throwIf(boolean condition, ErrorCode errorCode
        throwIf(condition, new BusinessException(errorCode));
    }


    public static void throwIf(boolean condition, ErrorCode errorCode
        throwIf(condition, new BusinessException(errorCode, message))
    }
}
```

响应包装类

common 包下面

通用返回包装类

```
@Data
public class BaseResponse<T> implements Serializable {

    private int code;

    private T data;

    private String message;

    public BaseResponse(int code, T data, String message) {
        this.code = code;
        this.data = data;
        this.message = message;
    }

    public BaseResponse(int code, T data) {
        this(code, data, "");
    }

    public BaseResponse(ErrorCode errorCode) {
        this(errorCode.getCode(), null, errorCode.getMessage());
    }
}
```

为了方便返回信息

```
public class ResultUtils {


    public static <T> BaseResponse<T> success(T data) {
        return new BaseResponse<>(0, data, "ok");
    }


    public static BaseResponse<?> error(ErrorCode errorCode) {
        return new BaseResponse<>(errorCode);
    }


    public static BaseResponse<?> error(int code, String message) {
        return new BaseResponse<>(code, null, message);
    }
```

```
    public static BaseResponse<?> error(ErrorCode errorCode, String m
        return new BaseResponse<>(errorCode.getCode(), null, message)
    }
}
```

全局异常处理器

exception 包下面

防止服务器的报错信息返回前端，增加系统风险

```
@RestControllerAdvice
@Slf4j
public class GlobalExceptionHandler {

    @ExceptionHandler(BusinessException.class)
    public BaseResponse<?> businessExceptionHandler(BusinessException
        log.error("BusinessException", e);
        return ResultUtils.error(e.getCode(), e.getMessage());
    }

    @ExceptionHandler(RuntimeException.class)
    public BaseResponse<?> runtimeExceptionHandler(RuntimeException e
        log.error("RuntimeException", e);
        return ResultUtils.error(ErrorCode.SYSTEM_ERROR, "系统错误");
    }
}
```

请求包装类

像是 分页 、删除 之类的都有很多重复的字段，其他请求直接继承这两个类即可减少代码书写

分页请求包装类

```
@Data
public class PageRequest {

    private int current = 1;

    private int pageSize = 10;
```

```
    private String sortField;


    private String sortOrder = "descend";
}
```

请求删除包装类

```
@Data
public class DeleteRequest implements Serializable {


    private Long id;

    private static final long serialVersionUID = 1L;
}
```

全局跨域配置

在 config 包下面

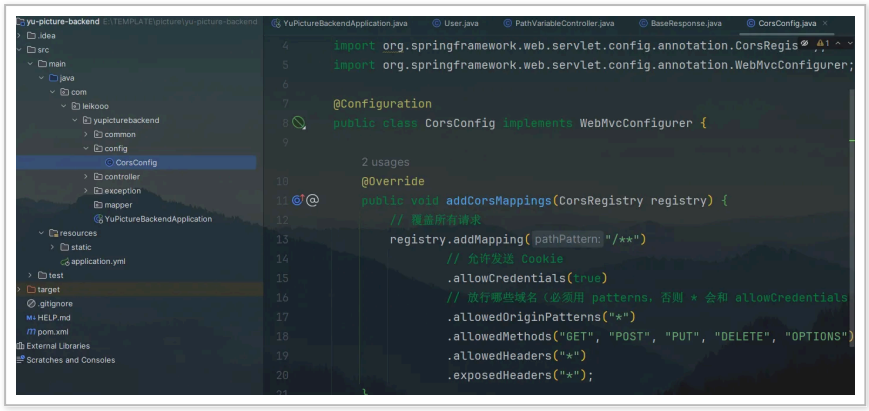
```
@Configuration
public class CorsConfig implements WebMvcConfigurer {

    @Override
    public void addCorsMappings(CorsRegistry registry) {

        registry.addMapping("/**")

                .allowCredentials(true)

                .allowedOriginPatterns("*")
                .allowedMethods("GET", "POST", "PUT", "DELETE", "OPTI
                .allowedHeaders("*")
                .exposedHeaders("*");
    }
}
```

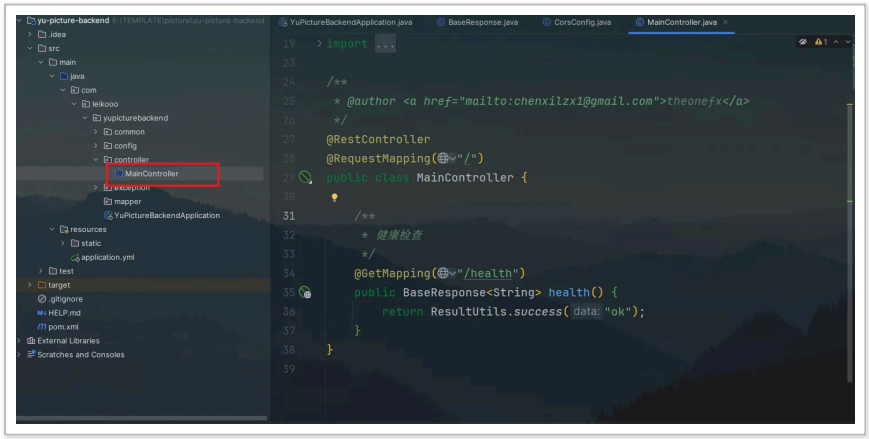


编写示例代码

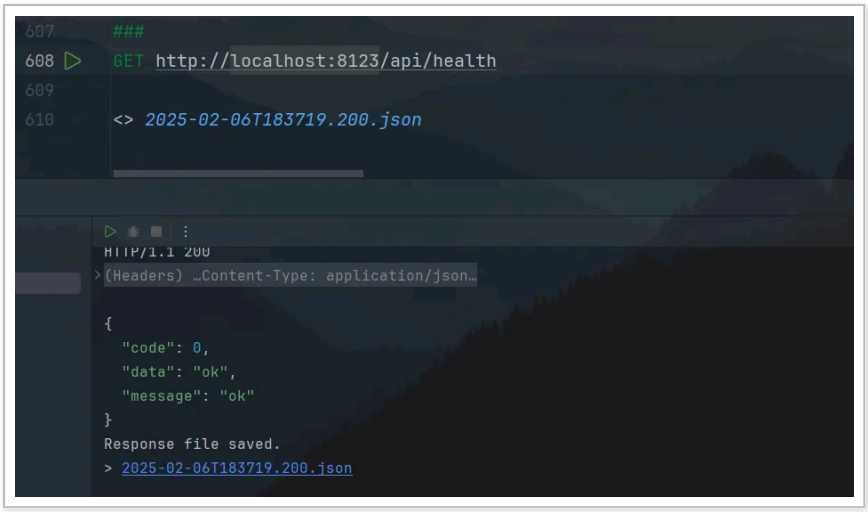
移除 controller 包下面的其他代码

```
@RestController
@RequestMapping("/")
public class MainController {


    @GetMapping("/health")
    public BaseResponse<String> health() {
        return ResultUtils.success("ok");
    }
}
```

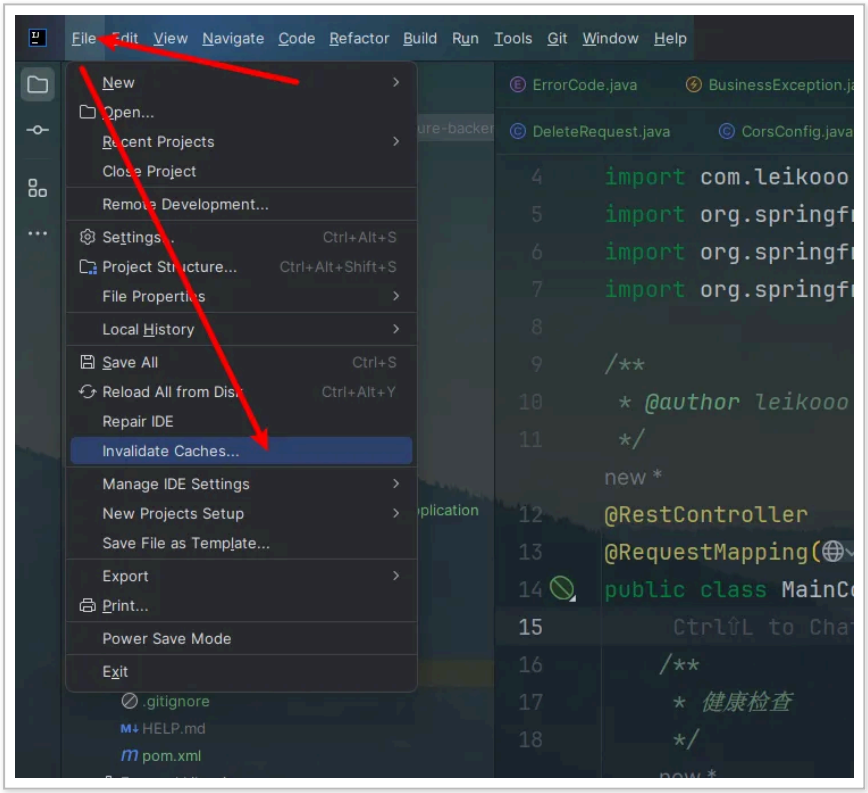


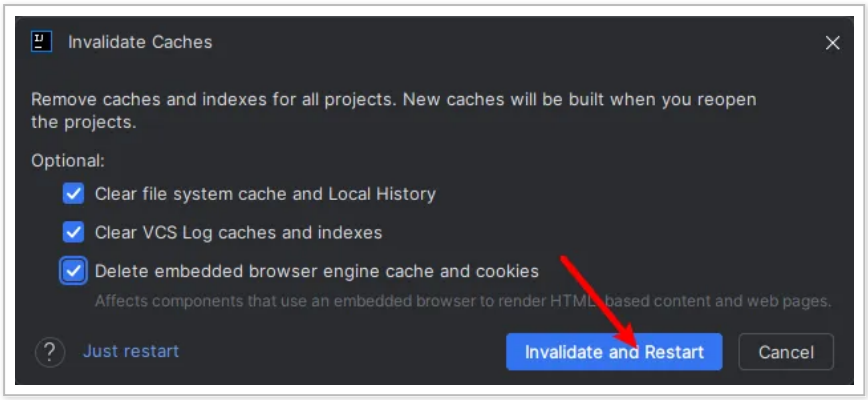
测试，只要看到下面的结果就初始化成功！q(≧▽≦q)



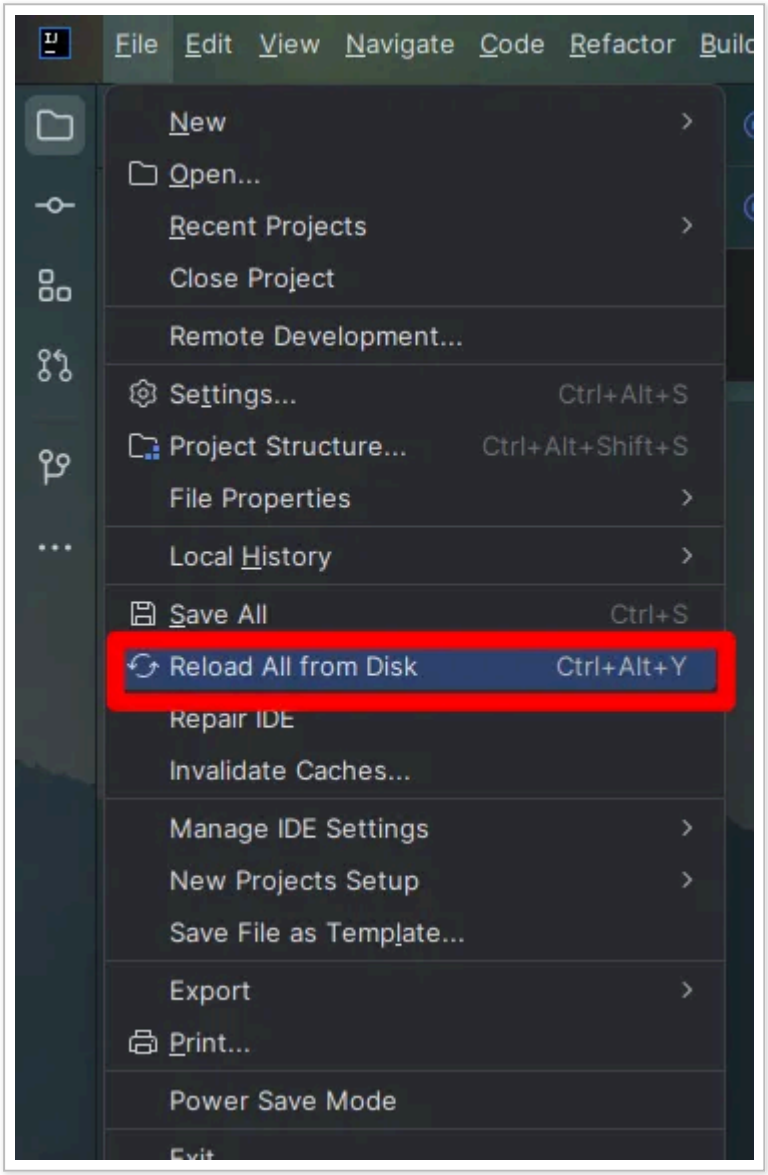
如果遇到编译器无端爆红但是不影响运行，清除一下缓存即可。

清除缓存 File | Invalidate Caches 全部勾上清楚一下





或者直接 Reload All from Disk 即可解决



全文完