基于STM32节点和阿里云IoT平台的物联网应用开发 系列课程

第四章

服务端的应用开发

life.augmented

阿里云

课程内容下载、观看

- 视频观看：AI电堂、阿里云大学IoT课堂
- 课件胶片下载：STMCU中文官网、阿里云大学IoT课堂
- 课件项目下载：STMCU中文官网、阿里云大学IoT课堂

STM32公众号

STM中文官网

电堂公众号

阿里云大学

第四章内容简介

- 第一节：综合软件架构介绍
  - 软件架构，知识结构梳理
- 第二节：后端服务开发
  - 后端框架，后端项目：和阿里云IoT平台对接，操作数据库，和前端应用对接
- 第三节：前端服务开发体验
  - 系统应用前端开发详解

STM32-阿里云IoT　联合课件开发

第四章．第二节
后端服务开发

life.augmented

阿里云

第四章.第二节 内容简介

- 认识后端框架
  - Mysql数据库介绍
  - MyBatis框架
  - SpringBoot框架
  - Maven介绍
- 初始化并运行第一个后端项目
  - 第一个springboot项目详解
  - 从无到有搭建第一个springboot项目
- 应用系统开发
  - 存储数据库设计
  - 配置服务端订阅
  - 应用系统与iot平台连接
  - 调用阿里云物联网接口
  - 接口编写
  - 跨域请求
- 应用调试与部署
  - 项目打包
  - 项目部署

了解后端基础概念

- 后端的基本概念
  - 在软体架构和程序设计领域，前端是软件系统中**直接和用户交互的部分**，而后端**控制着软件的输出**
  - 前端通过ajax等技术向后端进行网络请求；后端收到请求后对数据库进行操作，返回给前端JSON数据；前端把相应数据展示在页面上
  - 将软件分为前端和后端是一种将软件不同功能的部分相互分离的抽象

后端常用开发语言

Java

Java是一门面向对象编程语言，不仅吸收了C++语言的各种优点，还摒弃了C++里难以理解的多继承、指针等概念，因此Java语言具有功能强大和简单易用两个特征。

```
public class Test {
    public static void
main(String[] args)
{    System.out.println("hello!
!");
    }}}
```

Mysql

MySQL 是最流行的关系型数据库管理系统，操作数据库 MySQL 使用标准的 SQL 数据语言形式。

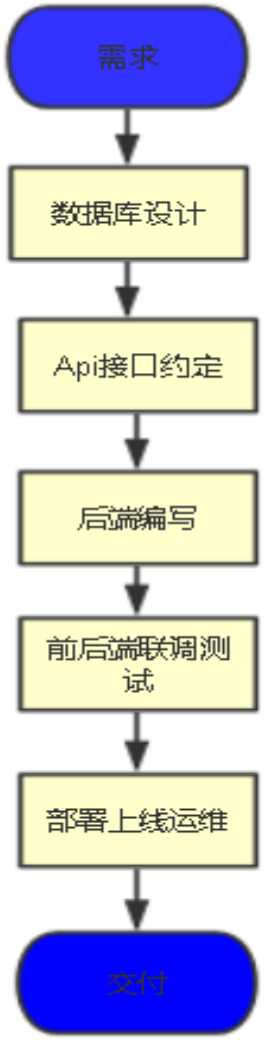
```
insert into student
values(null,'aa','男
','1988-10-2','.....');
```

XML

可扩展标记语言，标准通用标记语言的子集，是一种用于标记电子文件使其具有结构性的标记语言。它可以用来标记数据、定义数据类型，是一种允许用户对自己的标记语言进行定义的源语言，它非常适合万维网传输。

```
<note>
<to>George</to>
<from>John</from>
<body>Don't forget the
meeting!</body>
</note>
```

后端Web应用的开发流程



- 明确前后端系统需要实现的功能
- 后端数据库设计，明确数据库表字段，表之间关系
- 前后端约定API调用接口数据格式
- 后端代码编写
- 通过API接口进行前后端功能联调
- 后端程序部署到服务器，并依据系统能力进行扩容与负载均衡
- 软件交付验收

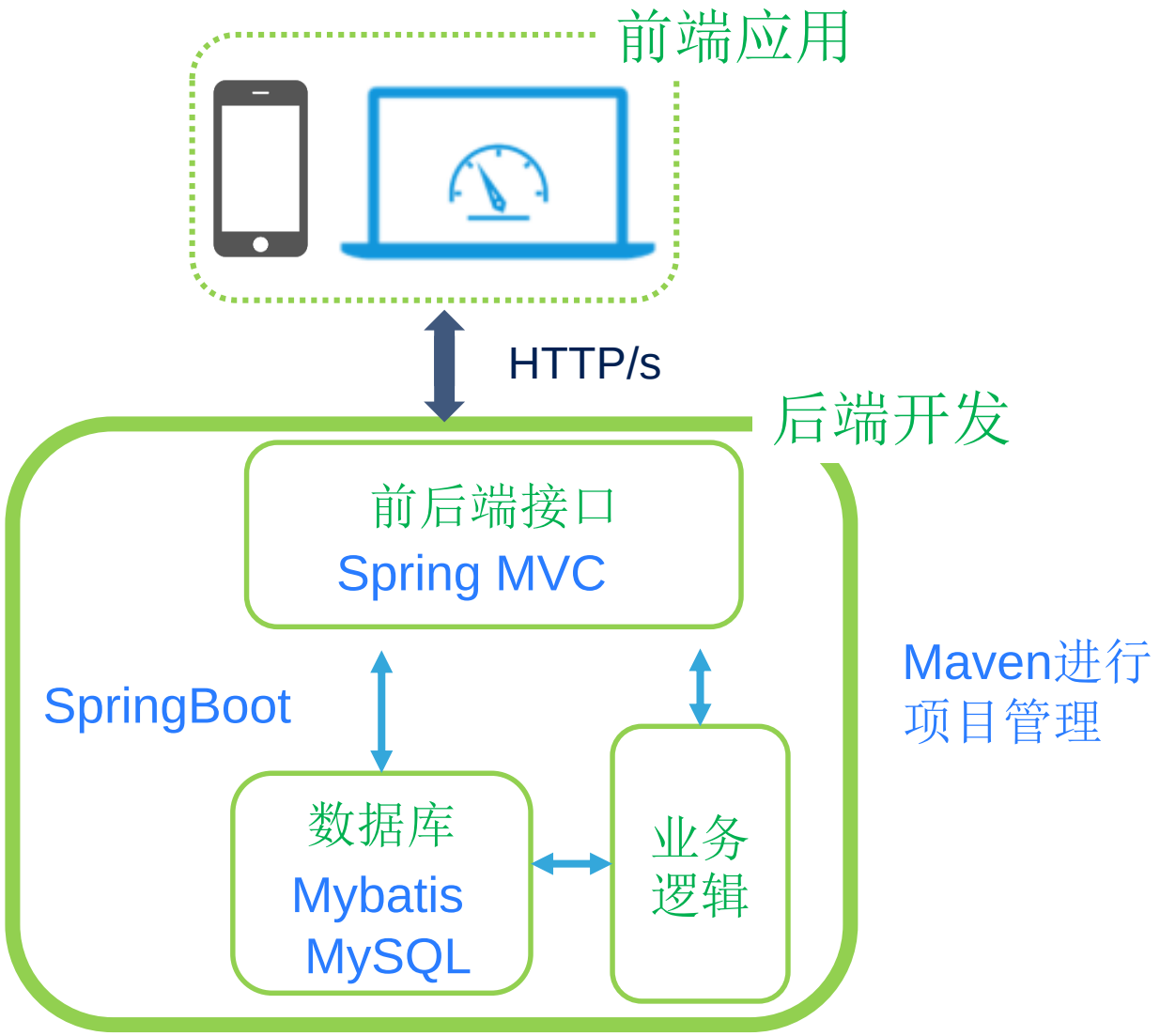
认识后端框架

- 基于框架开发
  - 随着计算机技术的不断发展，针对后端相关部分的开发，涌现出了大量具有强大功能的相关框架
  - 基于框架，开发者可以更方便快速的实现相关功能，并且在性能上也有良好的保证
- 本例程场景的智能家居平台，基于以下框架进行开发
  - MySQL
  - Mybatis
  - SpringBoot

MyBatis

认识后端框架

- MySQL：
  - 后端中各种数据的存储需要设计和使用数据库，MySQL是一种开放源代码的关系型数据库管理系统
- Mybatis
  - 在Java中操作Mysql语句一般用到持久层框架Mybatis;
- SpringBoot
  - 整合了常用框架Mybatis+springmvc等，省去了复杂的配置；
- Maven
  - 跨平台的项目管理工具。



Mysql数据库

- Mysql简介
  - SQL被美国国家标准局（ANSI）确定为关系型数据库语言的美国标准，后来被国际化标准组织（ISO）采纳为关系数据库语言的国际标准
  - MySQL是一种开放源代码的关系型数据库管理系统（RDBMS），关系数据库将数据保存在不同的表中，而不是将所有数据放在一个大仓库内，这样就增加了速度并提高了灵活性。
- **关系型数据库的典型概念**

  数据库：数据的仓库

  表：数据是保存在表内，保存在一个表内的数据，应该具有相同的数据格式

  行：每一行的数据

  列：列用于规定数据格式

  字段：数据的某个列

Mysql数据库

- 对Mysql数据库的操作
  - 操作Mysql主要是对数据库、表操作；可以用Navicat可视化工具，也可以用命令行操作。

  - 操作Mysql常用操作有：
    对数据库和表进行操作：创建数据库，删除数据库，切换数据库，创建表
    对数据库记录(行)进行操作：对数据库表记录插入，更新，删除
    对数据库查询操作，主要是用来查询数据，不会对数据造成变化。

Mybatis框架

- MyBatis简介
  - 支持定制化SQL、存储过程以及高级映射的优秀的持久层框架。MyBatis 避免了几乎所有的 JDBC 代码和手动设置参数以及获取结果集。MyBatis 可以对配置和原生Map使用简单的 XML或注解，将接口和 Java 的 POJOs(Plain Old Java Objects,普通的 Java对象)映射成数据库中的记录
- Mybatis优点
  - 简单易学：本身就很小且简单
  - 没有任何第三方依赖，最简单安装只要两个jar文件+配置几个sql映射文件，易于学习，易于使用，通过文档和源代码，可以比较完全的掌握它的设计思路和实现

MyBatis的功能架构

- API接口层
  - 提供给外部使用的接口API，开发人员通过这些本地API来操纵数据库。接口层一接收到调用请求就会调用数据处理层来完成具体的数据处理
- 数据处理层
  - 负责具体的SQL查找、SQL解析、SQL执行和执行结果映射处理等。它主要的目的是根据调用的请求完成一次数据库操作
- 基础支撑层
  - 负责最基础的功能支撑，包括连接管理、事务管理、配置加载和缓存处理，这些都是共用的东西，将他们抽取出来作为最基础的组件。为上层的数据处理层提供最基础的支撑

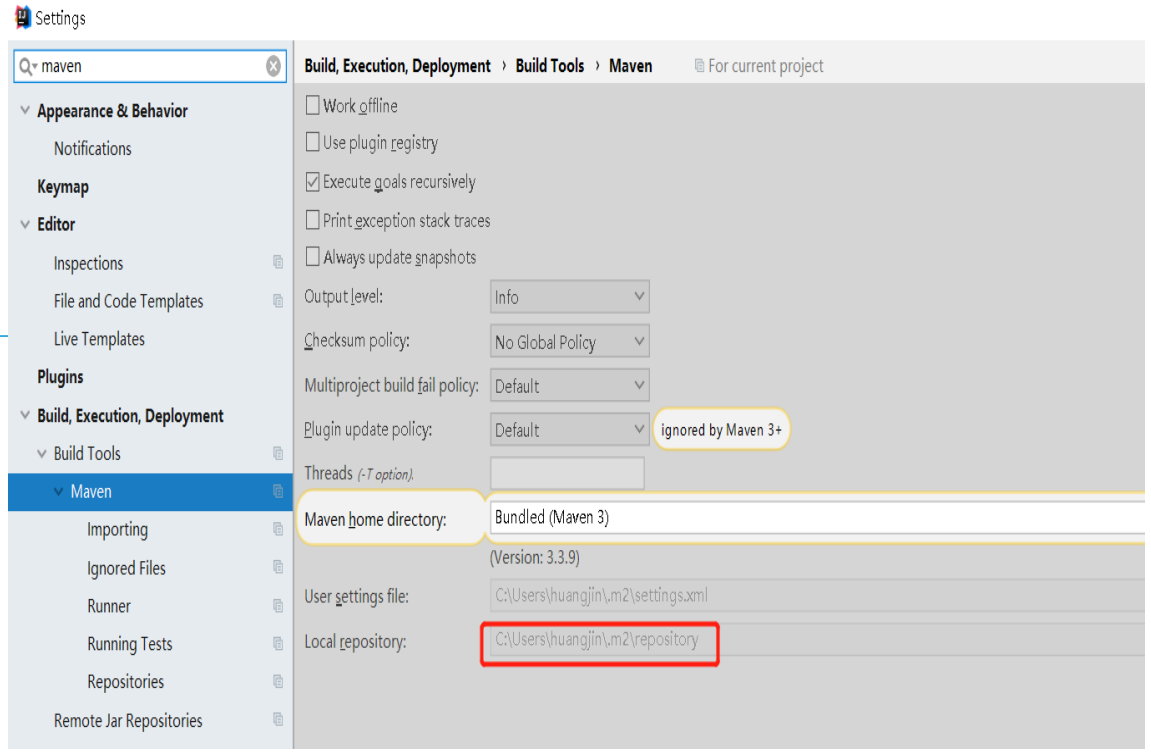
Springboot介绍

- Springboot解决的问题
  - 以往我们采用SpringMVC+Spring+Mybatis框架进行开发的时候，搭建和整合三大框架，我们需要做很多工作，比如配置web.xml，配置Spring，配置Mybatis,并将它们整合在一起等，而Spring boot框架对此开发过程进行了革命性的颠覆，抛弃了繁琐的xml配置过程，采用大量的默认配置简化我们的开发过程
- Springboot的特点
  - 可以完全不使用xml配置
  - 内嵌servlet容器，降低了对环境的要求，可用命令直接执行项目
  - 提供了starter POM，能够非常方便的进行包管理
  - 对主流框架无配置集成。

Maven介绍

- 主要作用是统一开发规范与工具以及统一管理jar包
  - 在idea配置好maven以后只需要在项目的pom.xml文件中加入依赖，例如

```
<!-- https://mvnrepository.com/artifact/tk.mybatis/mapper -->
<dependency>
    <groupId>tk.mybatis</groupId>
    <artifactId>mapper</artifactId>
    <version>4.1.5</version>
</dependency>
```

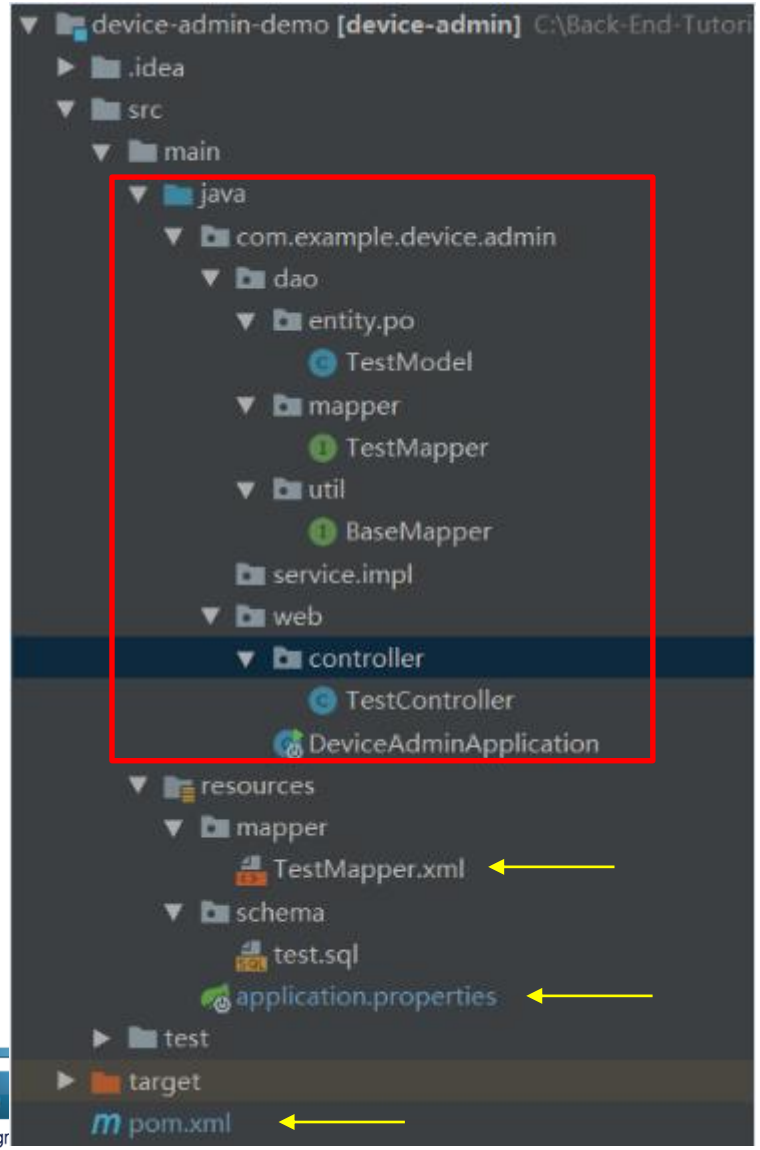


- Maven就从远程仓库中下载jar包至Idea配置的本机仓库地址中 IDEA自动携带Maven, 可以在idea settings中查看配置的maven 本地仓库

第四章.第二节 内容简介

- 认识后端框架
  - Mysql数据库介绍
  - MyBatis框架
  - SpringBoot框架
  - Maven介绍
- 初始化并运行第一个后端项目
  - 第一个springboot项目详解
  - 从无到有搭建第一个springboot项目
- 应用系统开发
  - 存储数据库设计
  - 配置服务端订阅
  - 应用系统与iot平台连接
  - 调用阿里云物联网接口
  - 接口编写
  - 跨域请求
- 应用调试与部署
  - 项目打包
  - 项目部署

第一个后端项目. 功能和结构



- 项目功能
  - 通过Navicat建立数据库表，数据记录
  - 通过Mybatis调用数据表的记录
  - 调用HTTP接口并返回查询结果，在浏览器客户端显示
- 项目结构
  - java文件
    - 5个Java文件
  - 配置文件application.properties
    - 配置数据库url，用户名、密码等
  - Mapper文件
    - 动态生成数据库操作的实现类
  - Maven工程配置：pom.xml
    - 打包方式、jar包依赖关系
    - 依赖库放在IDEA指定路径 (C:\Users\xxx\.m2\repository)

Maven工程配置

- 打包方式：jar
- 父工程：spring-boot-starter-parent
  - 是SpringBoot的父依赖，当前项目就是SpringBoot项目了
- 依赖关系

依赖的jar包	本项目要用到它的什么功能
mybatis-spring-boot-starter	引入Mybatis框架，方便对数据库的操作
mapper-spring-boot-starter	基于Mybatis的增强类，进一步简化对数据库的操作
spring-boot-starter-jdbc	连接数据库
spring-boot-starter-web	默认使用嵌套式的Tomcat作为Web容器对外提供HTTP服务
mysql-connector-java	是MySQL的JDBC驱动包
HikariCP	数据库连接池：负责分配、管理和释放数据库连接

应用配置文件

- 路径：src/main/resources/application.properties
- 主要配置了数据库等相关操作
  - 配置连接池
  - mapper文件位置
  - 前后端交互的端口号
  - 数据库url，用户名、密码

```
spring.application.name=device-admin
spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver
#配置数据源 jdbc数据库连接池参数
spring.datasource.type=com.zaxxer.hikari.HikariDataSource
spring.datasource.hikari.pool-name=AppHikariCP
spring.datasource.hikari.minimum-idle=5
spring.datasource.hikari.maximum-pool-size=25
spring.datasource.hikari.idle-timeout=30000
spring.datasource.hikari.max-lifetime=1800000
spring.datasource.hikari.connection-timeout=10000
spring.datasource.hikari.auto-commit=true
spring.datasource.hikari.connection-test-query=SELECT 1
#全局配置返回时间格式
spring.jackson.date-format= yyyy-MM-dd HH:mm
spring.jackson.time-zone=GMT+8

#设置扫描mapper文件的位置（存放sql的xml文件）
mybatis.mapper-locations=classpath:mapper/*.xml
#mybatis别名包
mybatis.type-aliases-package=com.example.device.admin.dao.entity
#禁用mybatis缓存
mybatis.configuration.cache-enabled=false

mapper.mappers=com.example.device.admin.dao.util.BaseMapper
mapper.not-empty=false
mapper.identity=MYSQL
#端口号
server.port=9099
server.servlet.context-path=/
```

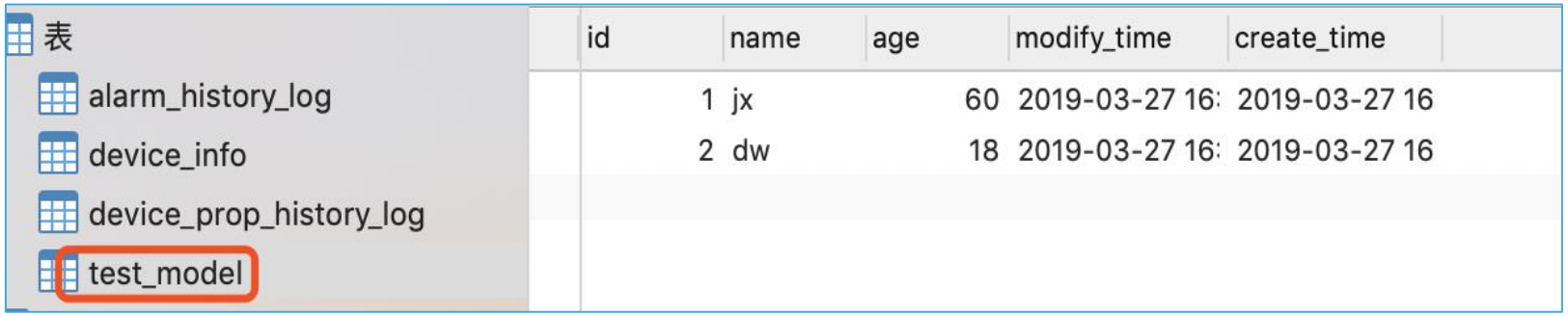
1 2 3

```
#连接数据库url
spring.datasource.url=jdbc:mysql://localhost:3306/test?useUnicode=true\
  &characterEncoding=UTF-8&autoReconnect=true\
  &useSSL=false&zeroDateTimeBehavior=convertToNull&serverTimezone=Asia/Shanghai
#连接数据库用户名
spring.datasource.username=root
#连接数据库密码
spring.datasource.password=****

#设置com.example.device.admin包下日志级别debug
logging.level.com.example.device.admin=DEBUG
```

4

数据库：使用Navicat建表

- 使用Navicat建立MySQL数据库
- 建表 test_model

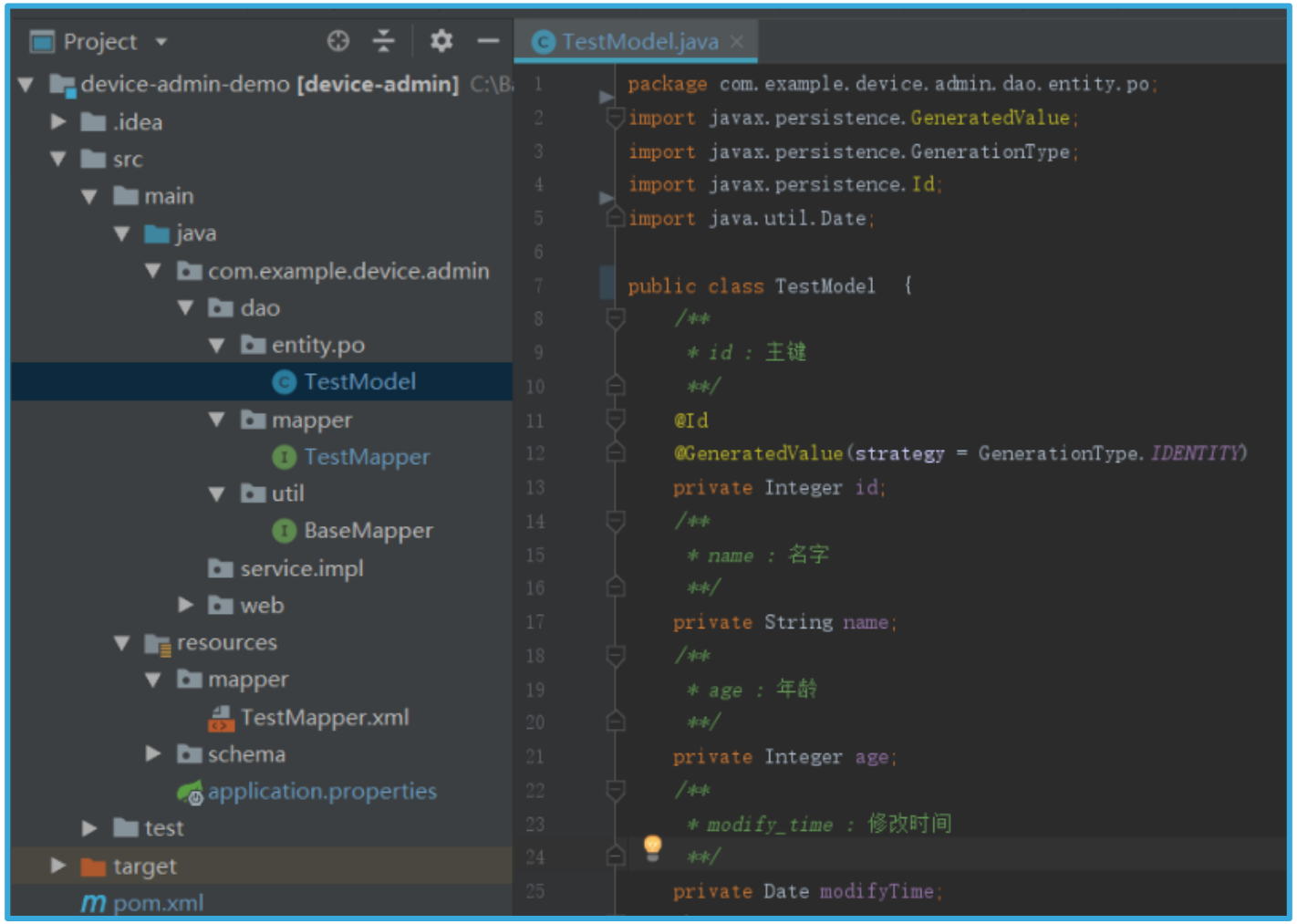


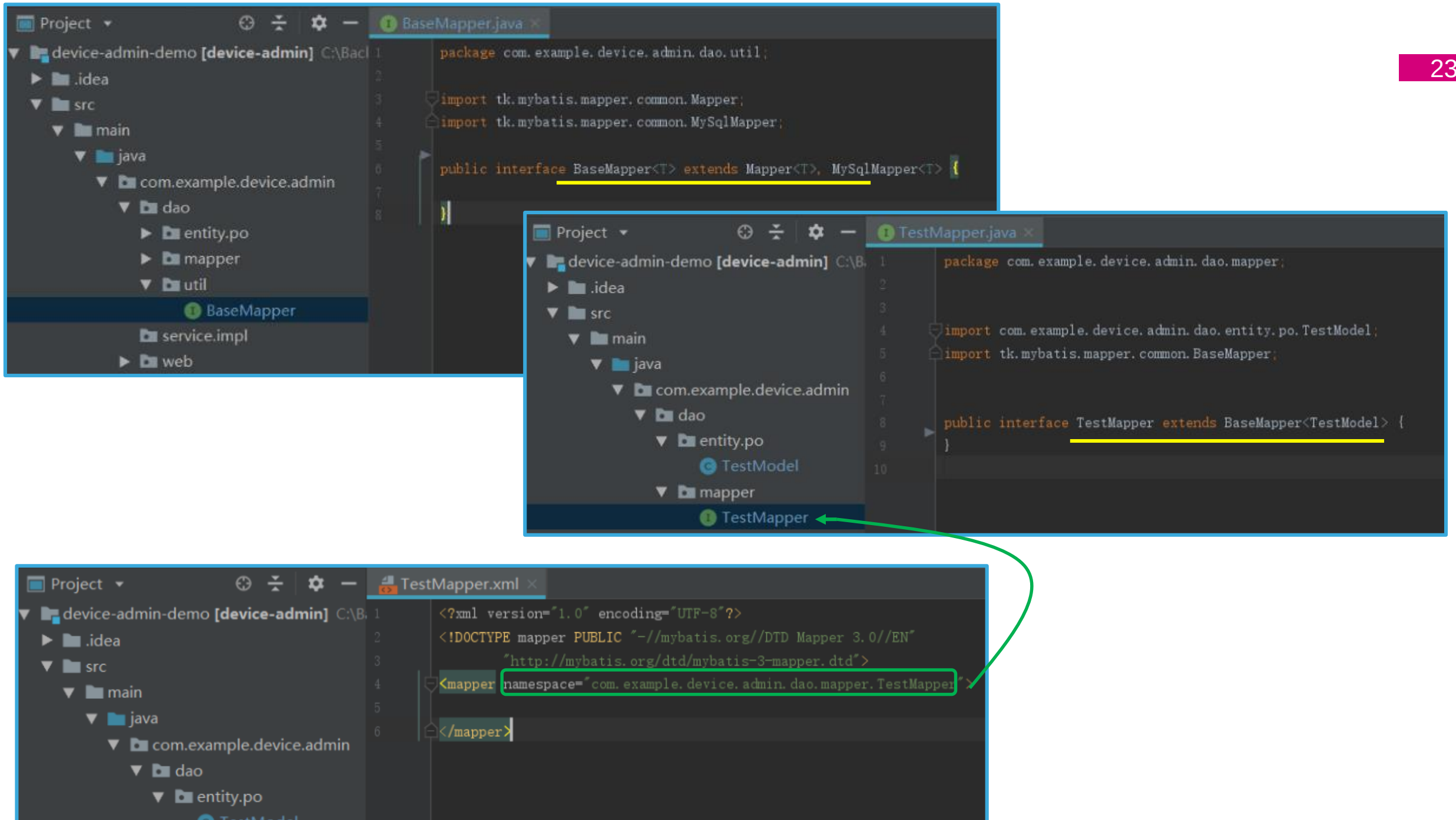
- 插入一条测试数据

```
insert into  `test_model`(`id`,`name`,`age`,`modify_time`,`create_time`)
values(1,‘李四’,18,‘2019-03-04 09:52:09’,‘2019-03-04 09:52:11’);
```

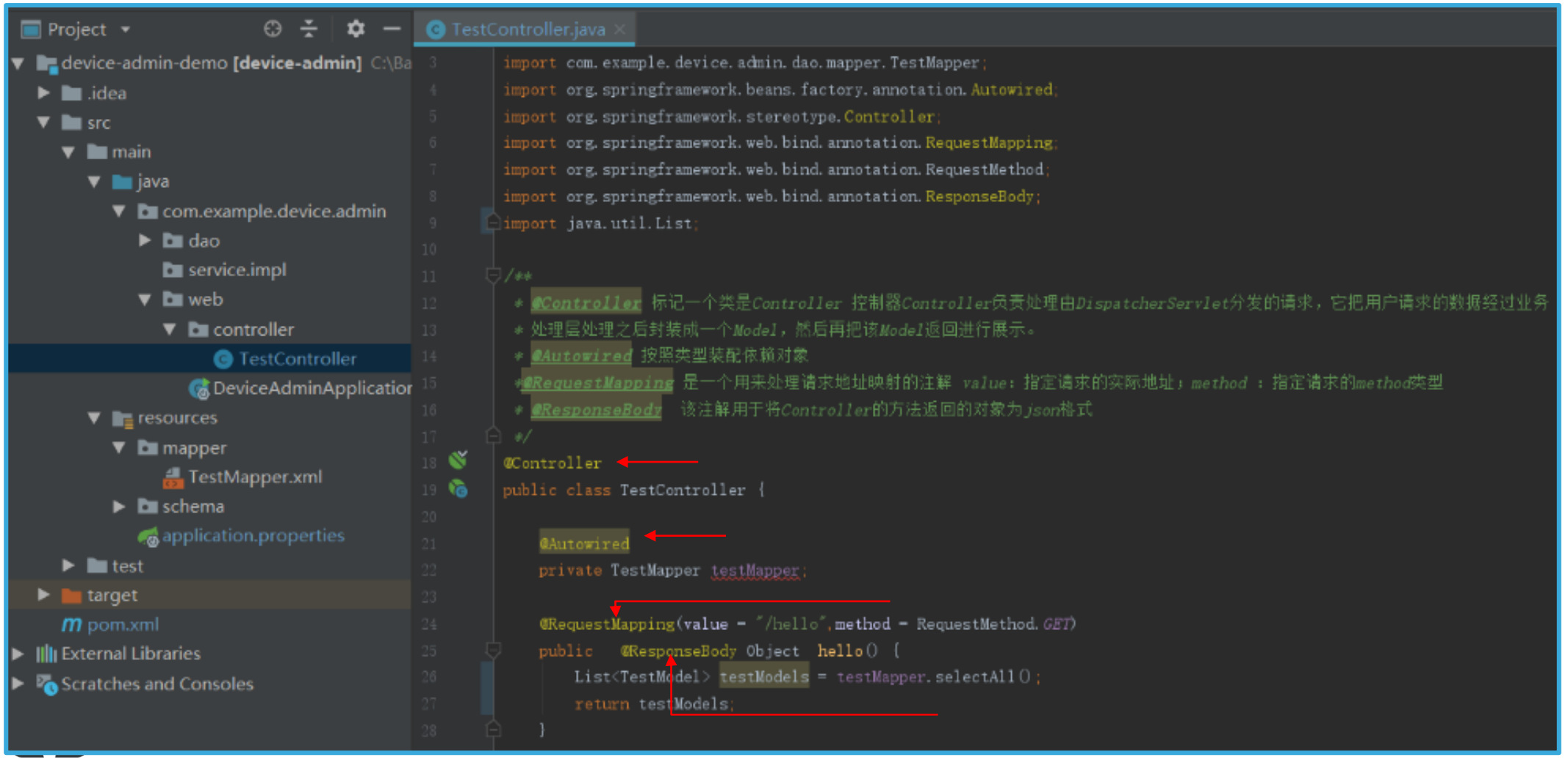
操作数据库：创建TestModel类

- 创建Java类：TestModel
  - 和新建数据表结构一一对应
    - id,name, age, modifyTime, createTime
- Mapper文件的使用
  - BaseMapper继承Mapper
  - TestMapper继承BaseMapper
  - 在TestMapper.xml中映射TestMapper中方法，此例为空（用到的方法已被封装好了）



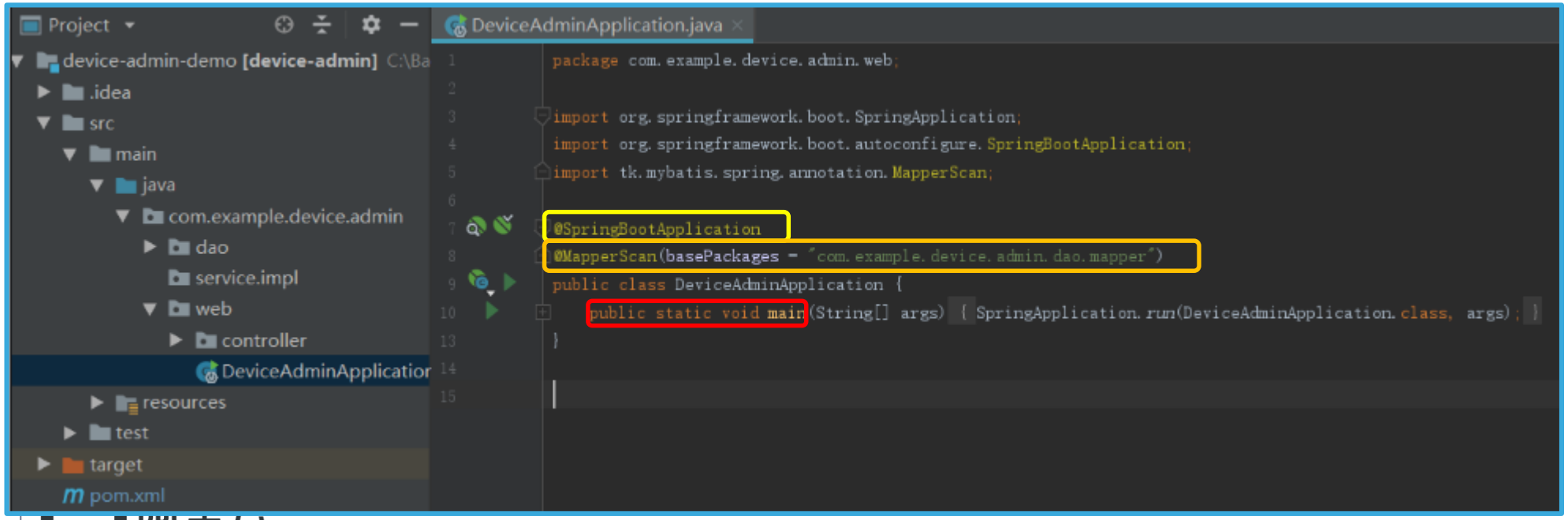

Project
device-admin-demo [device-admin]
.idea
src
main
java
com.example.device.admin
dao
entity.po
mapper
util
BaseMapper
service.impl
web
BaseMapper.java
package com.example.device.admin.dao.util;
import tk.mybatis.mapper.common.Mapper;
import tk.mybatis.mapper.common.MySqlMapper;
public interface BaseMapper<T> extends Mapper<T>, MySqlMapper<T> {
}
Project
device-admin-demo [device-admin]
.idea
src
main
java
com.example.device.admin
dao
entity.po
TestModel
mapper
TestMapper
TestMapper.java
package com.example.device.admin.dao.mapper;
import com.example.device.admin.dao.entity.po.TestModel;
import tk.mybatis.mapper.common.BaseMapper;
public interface TestMapper extends BaseMapper<TestModel> {
}
Project
device-admin-demo [device-admin]
.idea
src
main
java
com.example.device.admin
dao
entity.po
TestMapper.xml
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE mapper PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN"
"http://mybatis.org/dtd/mybatis-3-mapper.dtd">
<mapper namespace="com.example.device.admin.dao.mapper.TestMapper">
</mapper>

响应web请求：创建Controller类



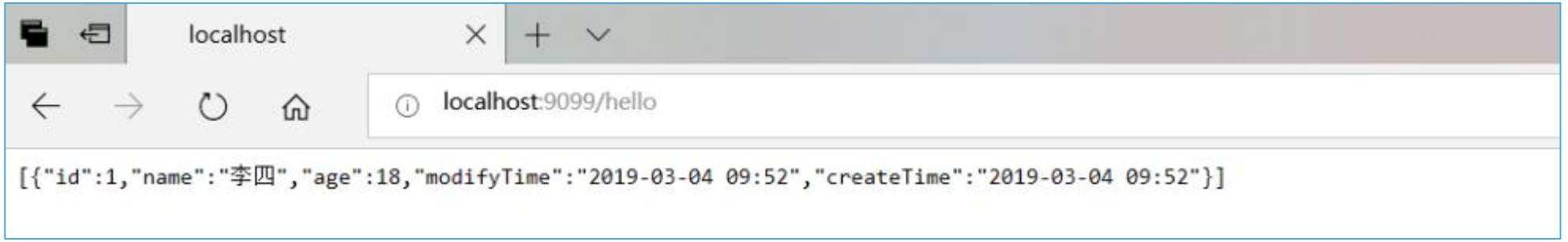
项目启动入口

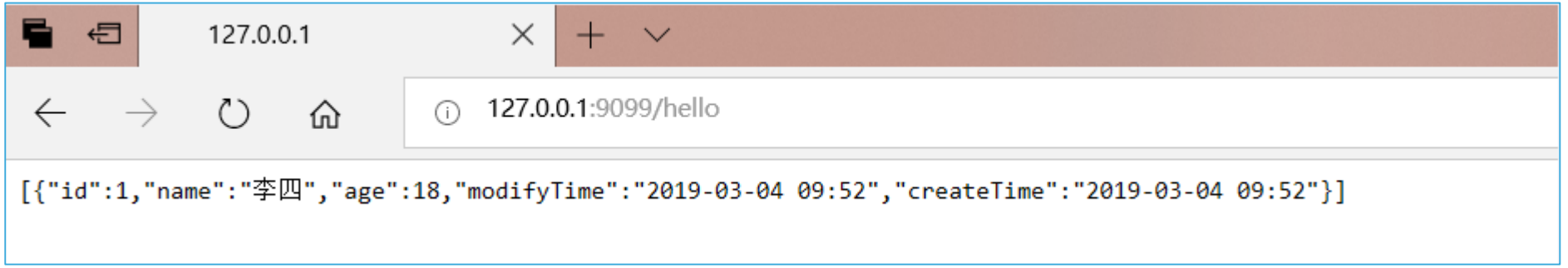
- 启动类是注解为@SpringBootApplication的类，以main方法为入口
- 使用@MapperScan可以指定要扫描的Mapper类的包的路径
- 启动时会读取application.properties全局配置文件里面的内容



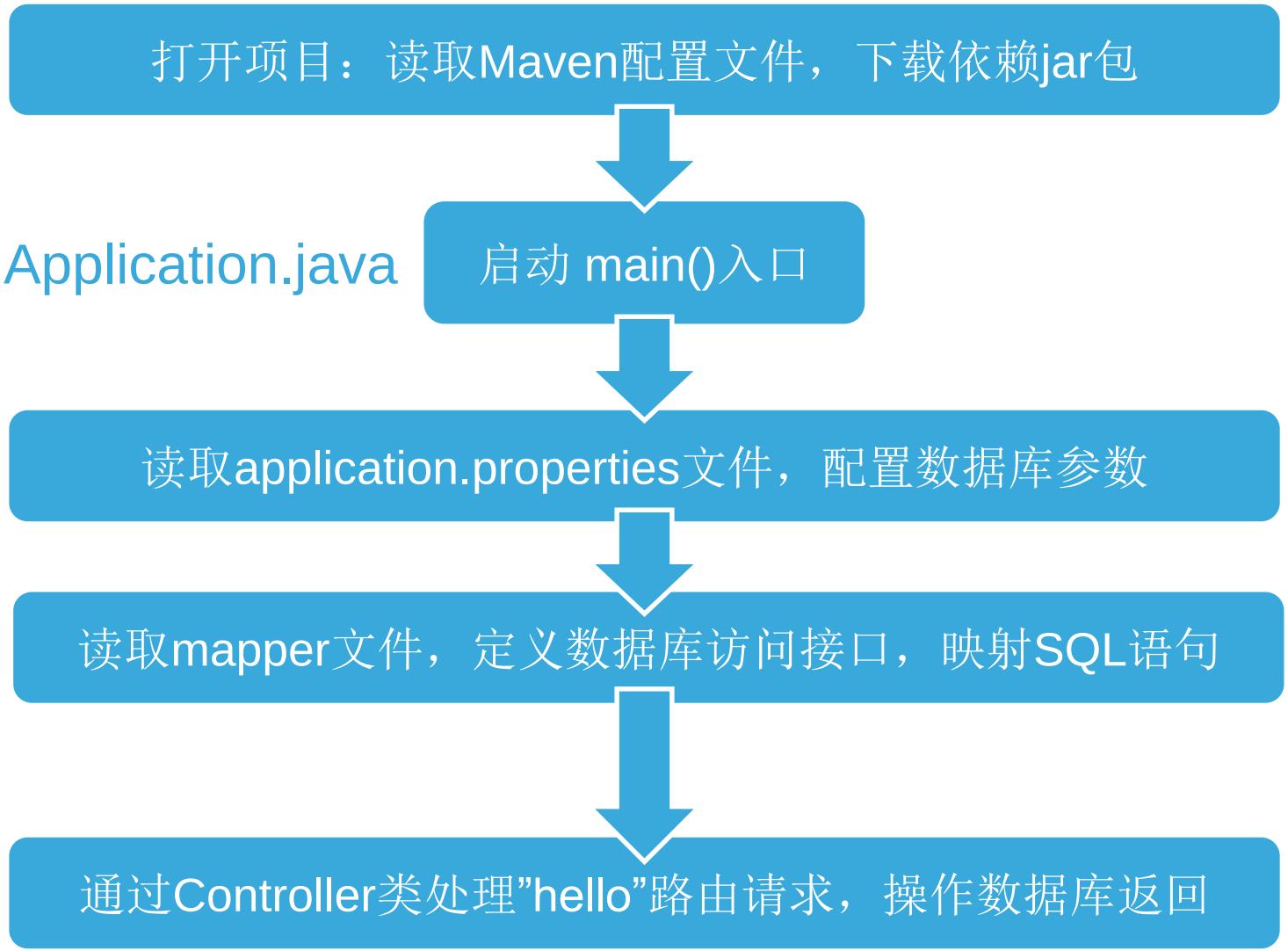
第一个后端项目. 运行

- localhost表示本机IP地址，也可使用127.0.0.1
- 后续还需替换成项目部署的IP地址 （详情参加本节第四部分内容：调试与部署）
- 9099作为端口号，在application.properties文件里有指定





项目运行的逻辑流程



mybatis-spring-boot-starter
mapper-spring-boot-starter
spring-boot-starter-jdbc
spring-boot-starter-web
mysql-connector-java
HikariCP

Basemapper.java
Testmapper.java
Testmapper.xml

TestModule.java

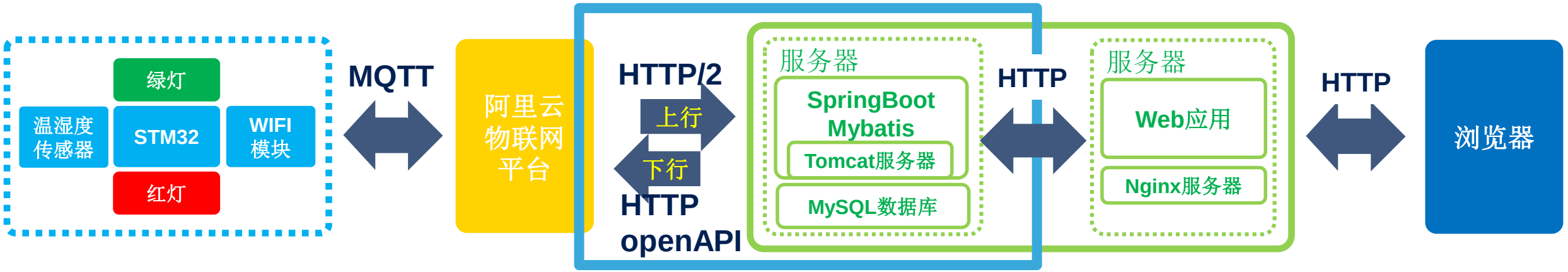
TestController.java

第四章.第二节 内容简介

- 认识后端框架
  - Mysql数据库介绍
  - MyBatis框架
  - SpringBoot框架
  - Maven介绍
- 初始化并运行第一个后端项目
  - 第一个springboot项目详解
  - 从无到有搭建第一个springboot项目
- 应用系统开发
  - 存储数据库设计
  - 配置服务端订阅
  - 应用系统与iot平台连接
  - 调用阿里云物联网接口
  - 接口编写
  - 跨域请求
- 应用调试与部署
  - 项目打包
  - 项目部署

后端项目例程.介绍

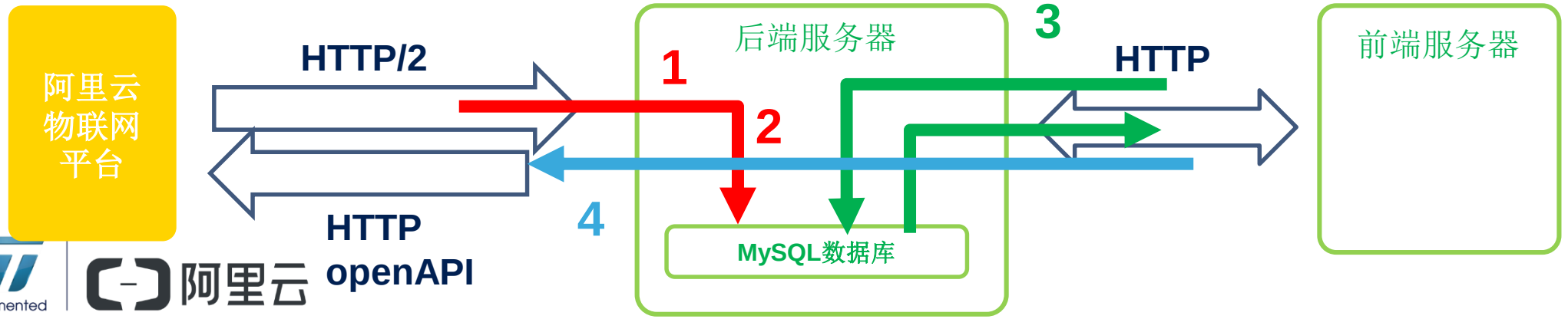
本节要实现的部分



- 阿里云IoT平台侧配置服务端订阅
- 数据库设计
  - 表格设计
  - 上行消息写入数据库
  - 根据前端请求，查询数据库
- 和阿里云IoT平台的通信
  - 处理上行 @ HTTP/2客户端
  - 处理下行 @ 调用openAPI
- 和前端服务器交互接口编写
  - 跨域请求

后端项目例程．功能

- （通过Navicat建立数据库表）
- 【1】接收来自阿里云IoT平台推送的节点设备上报消息
- 【2】写入数据库（创建数据记录）
- 【3】接收前端服务器的数据库查询操作，并返回对应信息
- 【4】接收前端服务器命令，转发给阿里云IoT平台



数据库的设计

- 和前端服务器的交互API定义
  - GET/api/v1/device/clearAlarm
    （设备报警状态取消）
  - GET/api/v1/device/listDeviceName
    （查询所有设备名）
  - GET/api/v1/device/queryAlarmHistoryLogs
    （查询一段报警数据）
  - GET/api/v1/device/queryDeviceProp
    （获取设备属性）
  - GET/api/v1/device/queryDevicePropHistoryLogs
    （查询一段时间温湿度数据）
  - GET/api/v1/device/setDeviceProperty
    （修改方式设备报警阀值）

- 三个表格

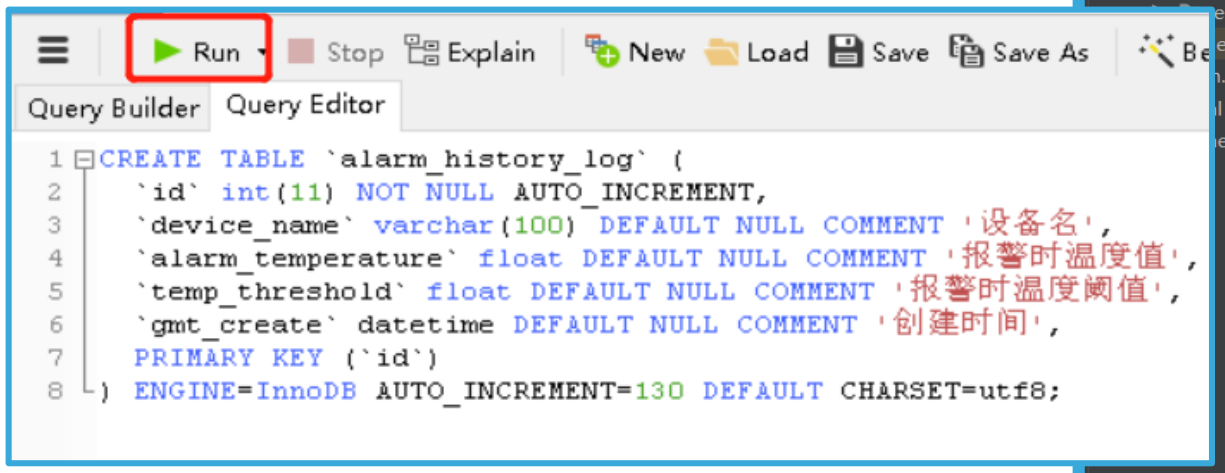
device_info
id
device_name
current_temperature
current_humidity
temp_threshold
alarm_status
last_alarm_date
last_online_date
connect_status
gmt_update

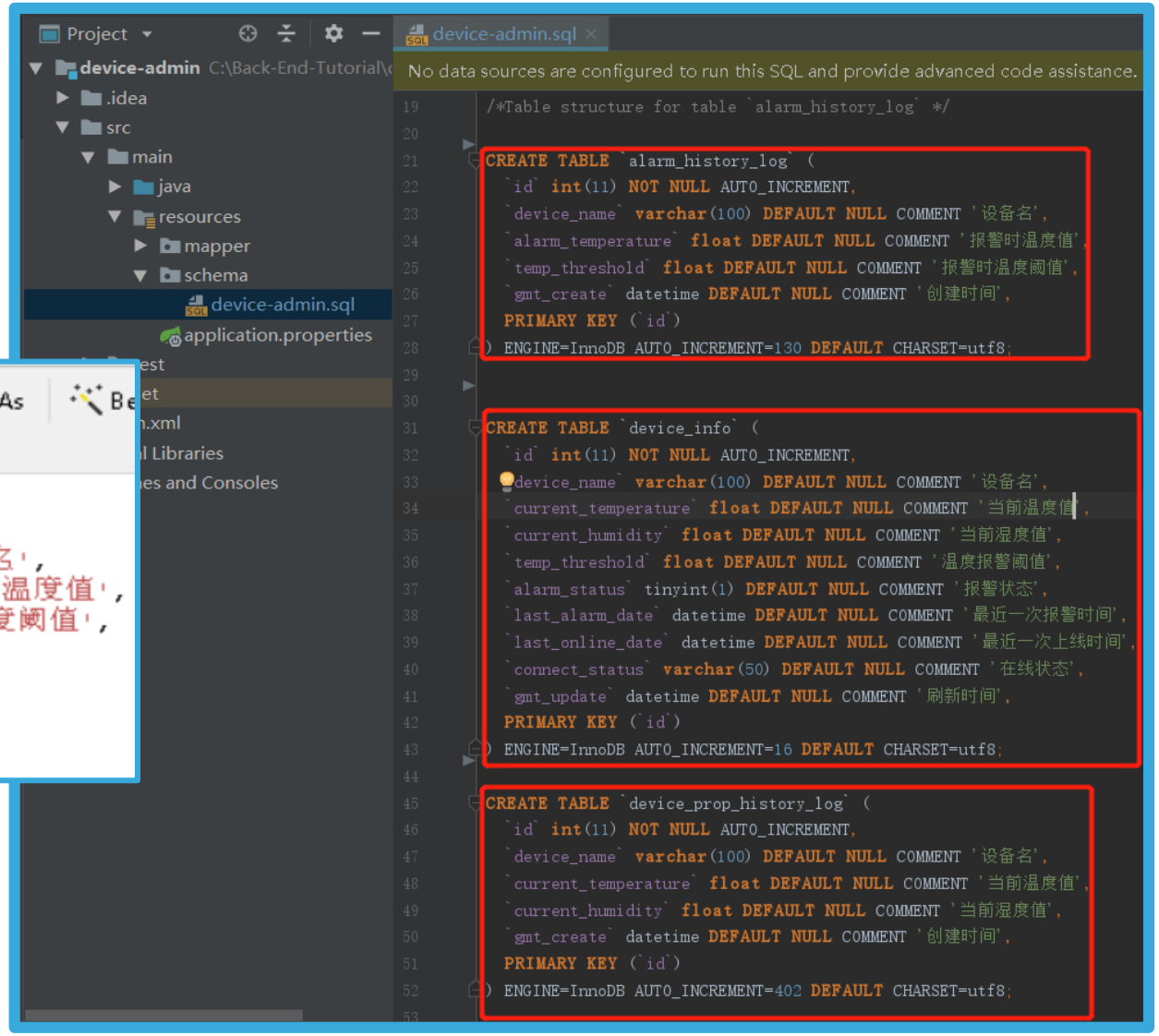
alarm_history_log
id
device_name
alarm_temperature
temp_threshold
gmt_create

device_prop_history_log
id
device_name
current_temperature
current_humidity
gmt_creat

数据库：使用Navicat建表

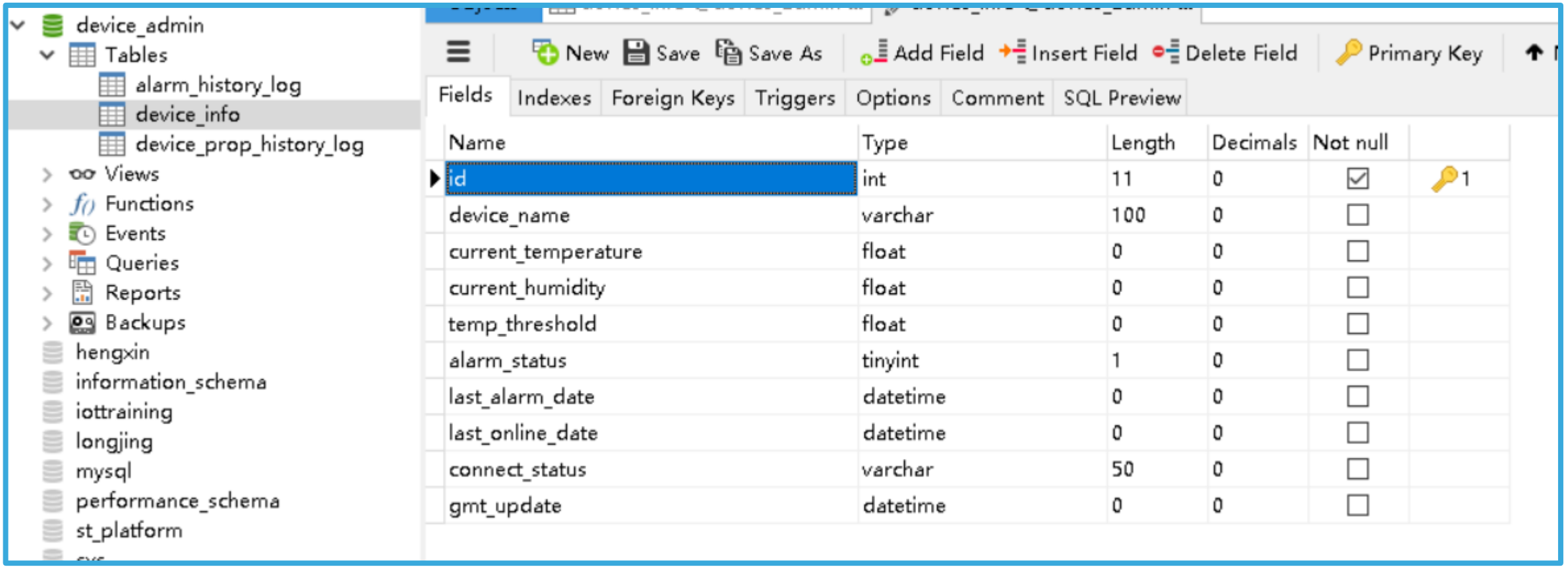
- 使用Navicat建立MySQL数据库
  - 复制建表语句到Navicat，并执行
  - 分别建立三个表





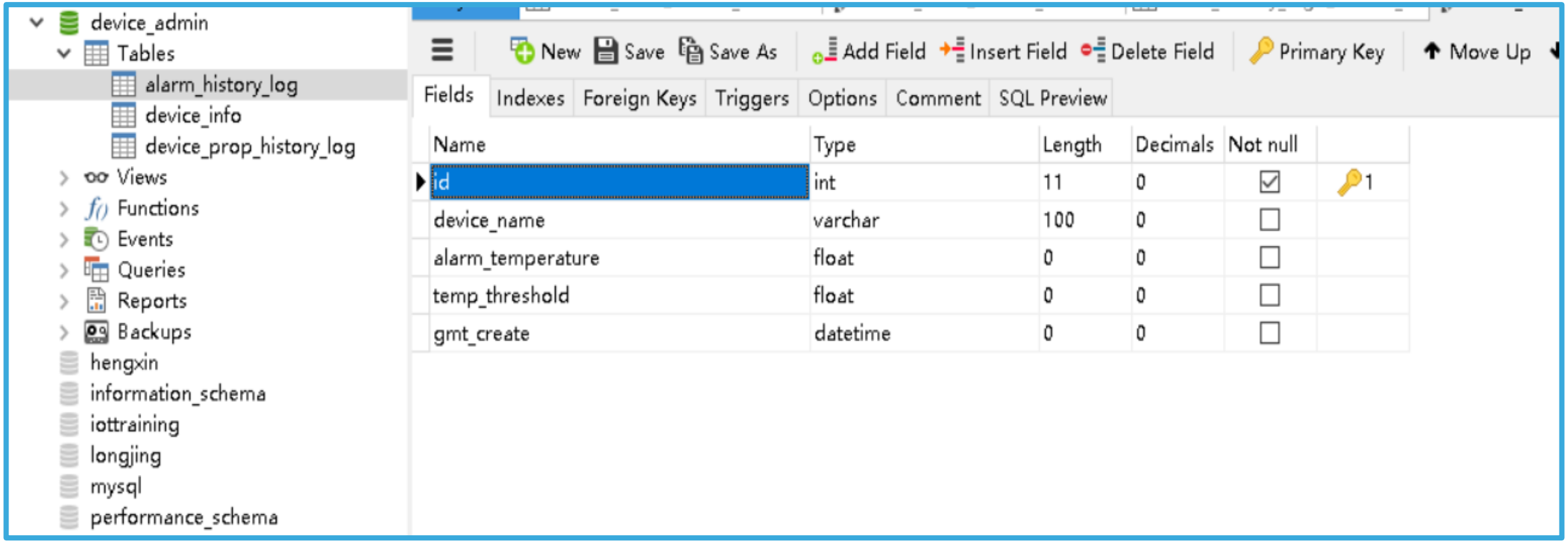
数据库：使用Navicat建表

- 表device_info记录设备当前状态：温度、湿度、报警阈值，连接状态等



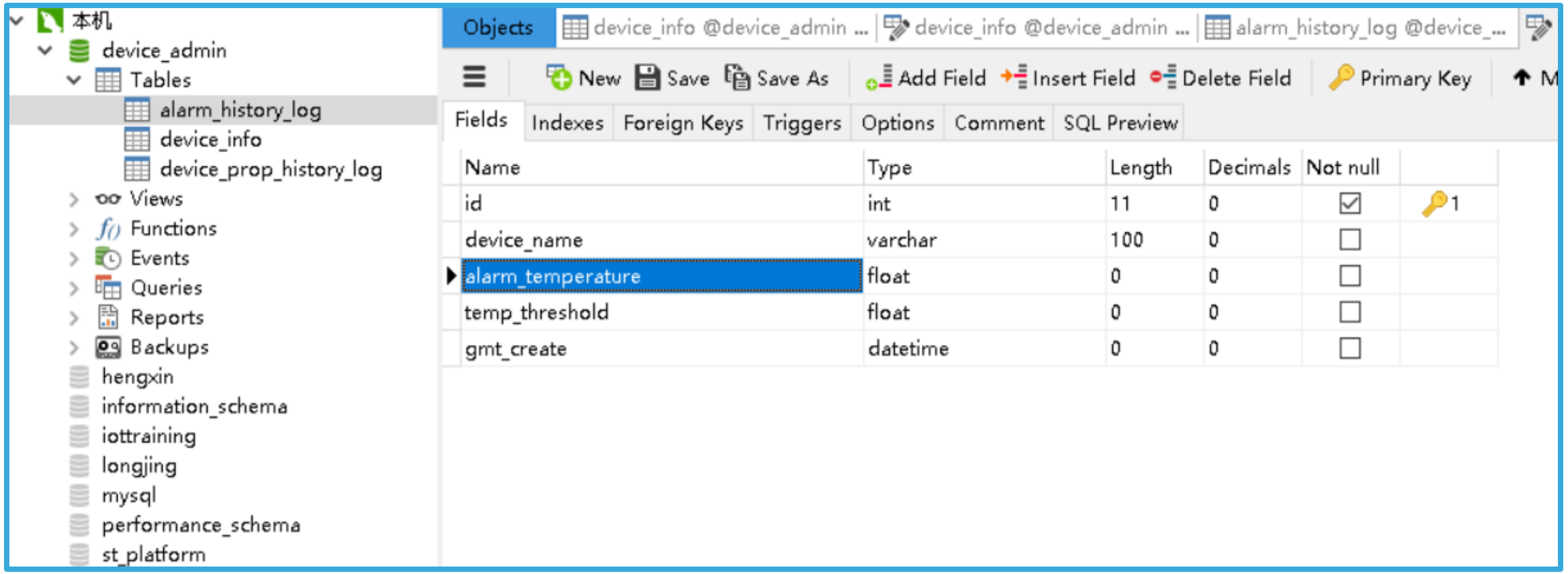
数据库：使用Navicat建表

- 表device_prop_history_log记录设备温湿度值历史数据

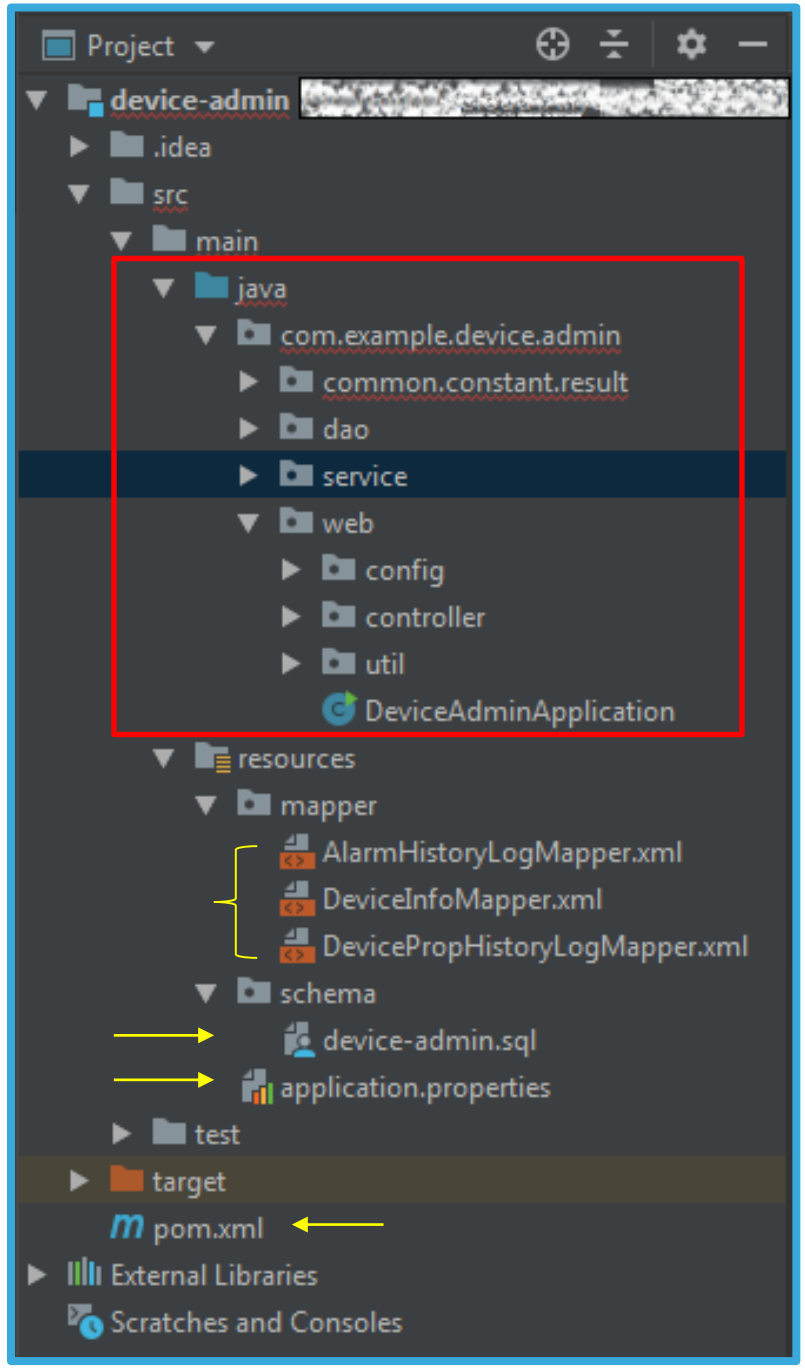


数据库：使用Navicat建表

- 表alarm_history_log记录设备报警温度及温度阈值



后端项目例程 . 结构



- 项目结构
  - Java文件
    - common：主要存放应用中常量，异常，工具类等基础类
    - dao：数据操作层，主要存放Dao类
    - service：主要存放业务处理类（service）
    - web：提供HTTP接口供前端调用，同时也是工程启动入口
      - DeviceAdminApplication.java
  - 其他辅助文件
    - 配置文件 @ application.properties
    - Mapper文件
    - Maven工程配置文件 @ pom.xml
    - 数据库测试文件 @ device-admin.sql

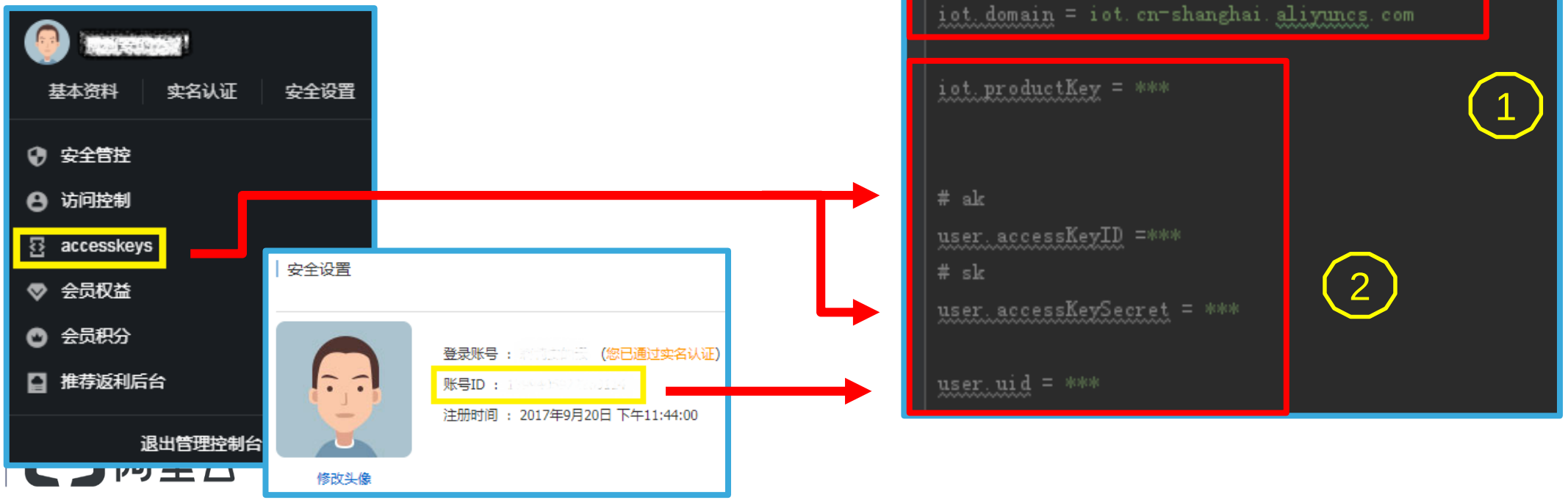
Maven工程配置

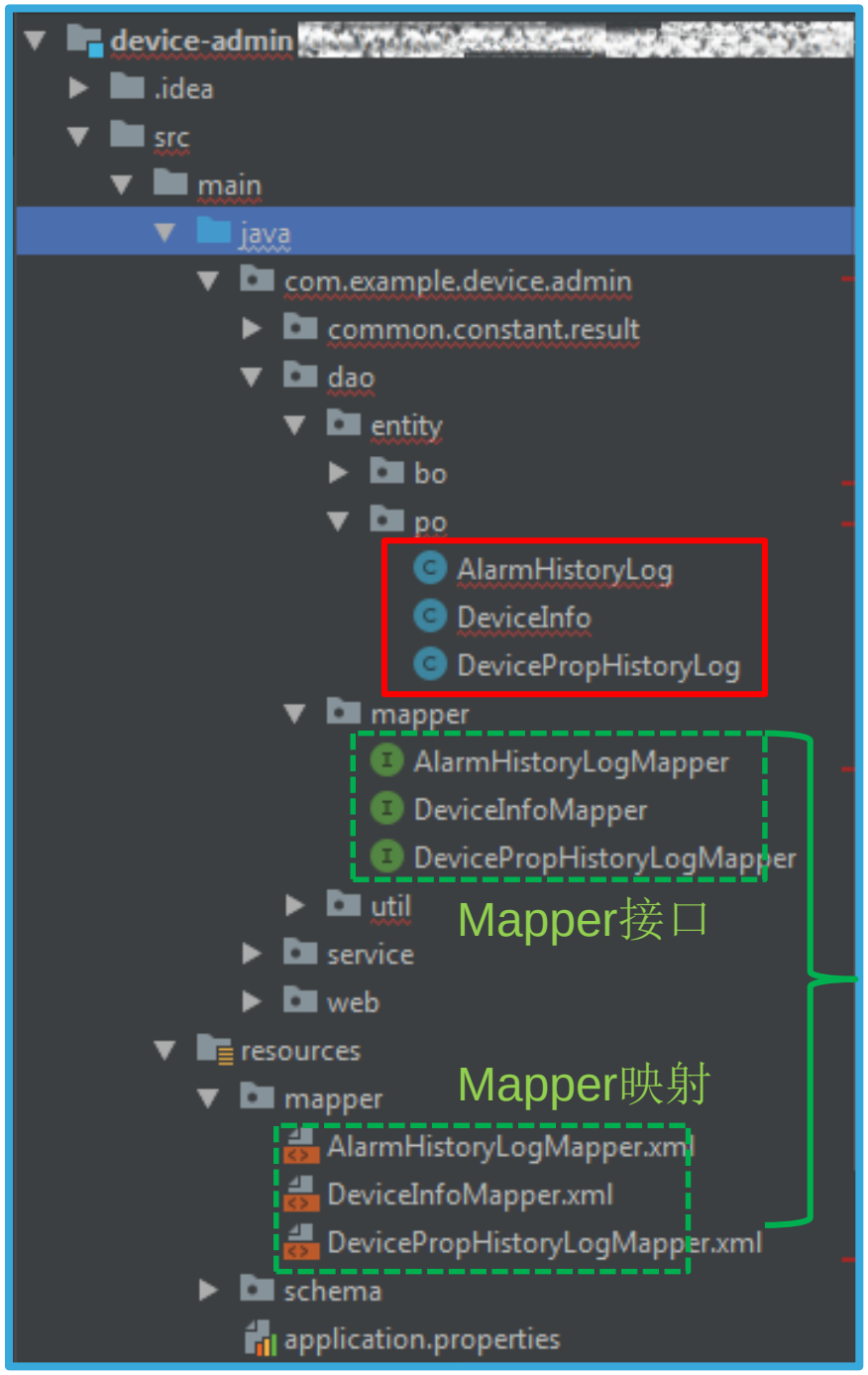
- 打包方式：jar
- 父工程：spring-boot-starter-parent
  - 是Spring boot的父依赖，当前项目就是Spring boot项目了
- 【新增】依赖关系

依赖的jar包	本项目要用到它的什么功能
aliyun-java-sdk-core	阿里云java 底层SDK
aliyun-java-sdk-iot	用户服务器调用云端openAPI，以实现产品管理、设备管理、Topic管理等功能（主要是下行操作）
iot-client-message	用户服务器通过HTTP/2通道，接收物联网平台推送过来的设备消息（主要是上行操作）

应用配置文件

- 路径：src/main/resource/application.properties
- 【新增】对阿里云IoT平台访问的配置
  - 阿里云IoT平台的账号信息
  - 阿里云IoT平台访问区域信息
  - 阿里云IoT平台上要访问的产品信息





操作数据库：创建Java类

- 创建**Java类**和所建数据表结构一一对应
  - 三个类 Vs. 三个表
    - AlarmHistoryLog类
    - DeviceInfo类
    - DevicePropHistoryLog类
  - 抽象出表结构和可操作的方法
- **Mapper文件**的使用：Mapper接口 Vs. Mapper映射
  - 使用默认已经封装好的方法
  - 手动映射：把mapper.xml中SQL语句和mapper.java中的方法对应起来

操作数据库：Mapper的使用

AlarmHistoryLogMapper.java

```java
public interface AlarmHistoryLogMapper extends BaseMapper<AlarmHistoryLog> {

}
```

AlarmHistoryLogMapper.xml

```xml
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE mapper PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN" "http://mybatis.org/dtd/mybatis-3-mapper.dtd" >
<mapper namespace="com.example.device.admin.dao.mapper.AlarmHistoryLogMapper"></mapper>
```

DevicePropHistoryLogMapper.java

```java
public interface DevicePropHistoryLogMapper extends BaseMapper<DevicePropHistoryLog> {

}
```

DevicePropHistoryLogMapper.xml

```xml
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE mapper PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN" "http://mybatis.org/dtd/mybatis-3-mapper.dtd" >
<mapper namespace="com.example.device.admin.dao.mapper.DevicePropHistoryLogMapper"></mapper>
```

DeviceInfoMapper.java

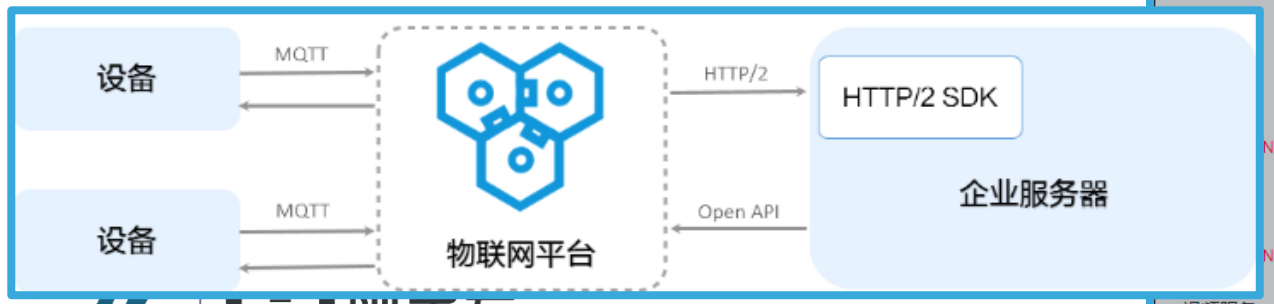
```java
public interface DeviceInfoMapper extends BaseMapper<DeviceInfo> {

    public int insertOrUpdate(DeviceInfo deviceInfo);
}
```

DeviceInfoMapper.xml

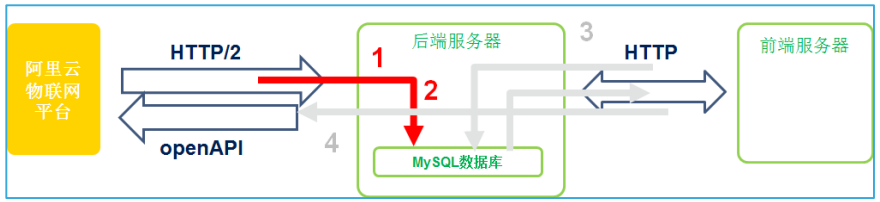
```xml
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE mapper PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN" "http://mybatis.org/dtd/mybatis-3-mapper.dtd">
<mapper namespace="com.terabits.device.admin.dal.mapper.DeviceInfoMapper">
    <insert id="insertOrUpdate" parameterType="com.example.device.admin.dao.entity.po.DeviceInfo">
        INSERT INTO device_info(device_name,current_temperature,current_humidity,temp_threshold)
        VALUES(#{deviceName},#{currentTemperature},#{currentHumidity},#{tempThreshold}) ON DUPLICATE KEY UPDATE
                device_name=#{deviceName},
                current_temperature=#{currentTemperature},
                current_humidity=#{currentHumidity},
                temp_threshold=#{tempThreshold}
    </insert>
</mapper>
```

服务器端订阅消息

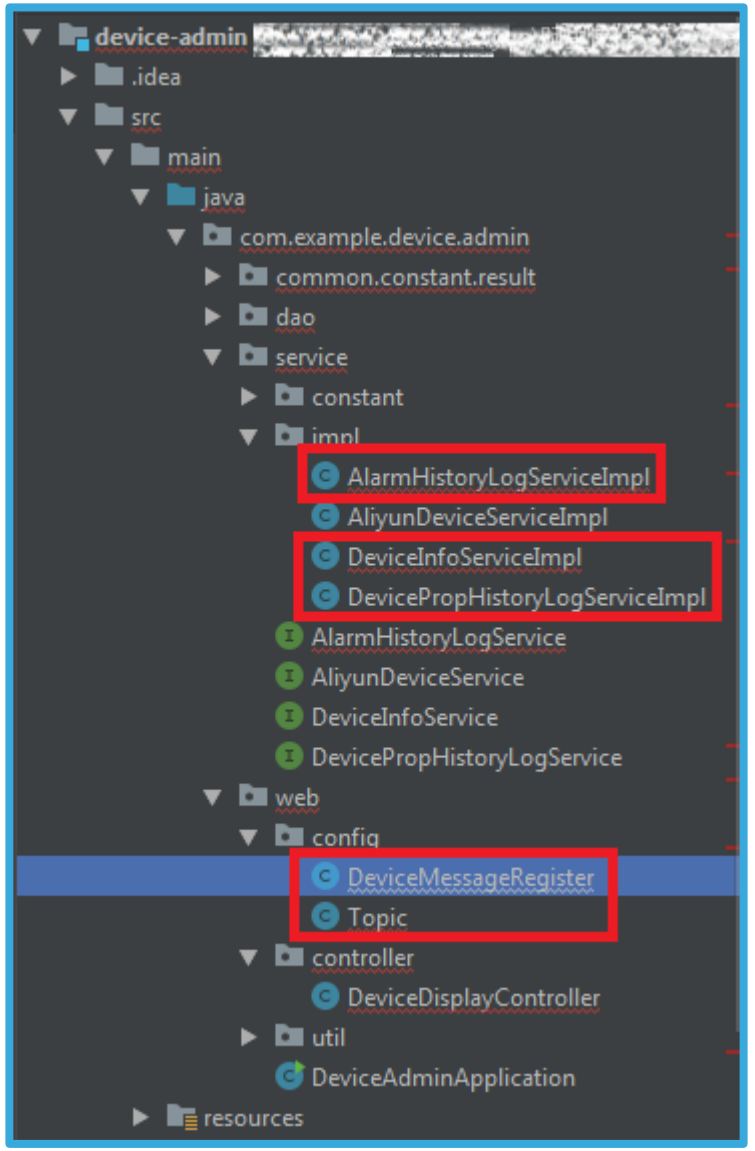
- 服务端可直接订阅产品消息
- 阿里云IoT平台通过HTTP/2通道进行消息流转
- HTTP/2 SDK
  - 提供身份验证、topic订阅、消息发送、消息接收等能力
  - 适用于平台和服务器间大量数据流转



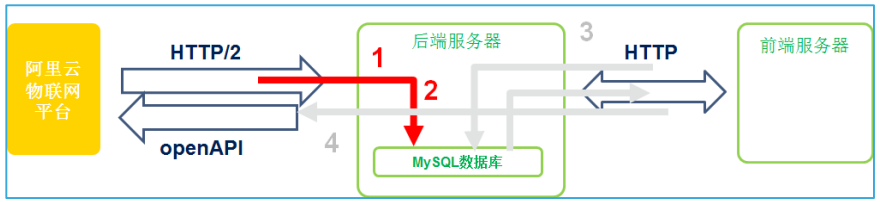
- 登录阿里云IoT平台/控制台设置
- 选择推送的消息类型
  - 设备上报消息：所有具有发布权限的topic
  - 设备状态变化通知：上线、下线
  - 设备生命周期变更：设备创建、删除等



后端服务器应用开发 . 上行



- 创建DeviceMessageRegister类，实现InitializingBean接口；在spring初始化bean的时候，会自动调用afterPropertiesSet方法
- 建立和阿里云IoT平台的连接
  - 基于用户的阿里云 AccessKey 进行身份认证
- 订阅主题
- 监听，通过SDK消息回调获得数据
  - messageID、topic、payload、generateTime、qos
- 根据不同主题，操作数据库



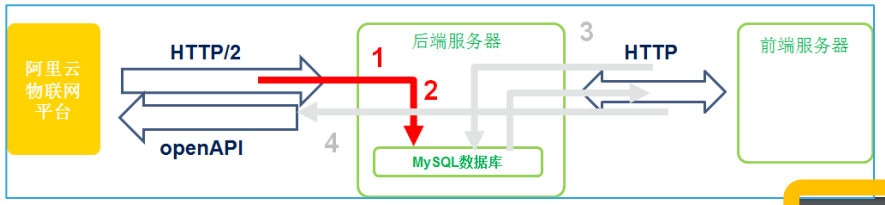
后端服务器应用开发 . 上行

- 连接前的身份认证
  - **accessKey** 即账号的 AccessKey ID
  - **accessSecret** 即 AccessKey ID对应的 AccessKey Secret
  - 选择**安全设置**查看账号 ID
  - **regionId**为所在物联网平台服务地域

DeviceMessageRegister.java

```
// endPoint: "https://${uid}.iot-as-http2.${region}.aliyuncs.com"
String endPoint = "https://" + uid + ".iot-as-http2." + regionId + ".aliyuncs.com";

// 连接配置
Profile profile = Profile.getAccessKeyProfile(endPoint, regionId, accessKey, accessSecret);

// 构造客户端
MessageClient client = MessageClientFactory.messageClient(profile);
client.setMessageListener(new MessageCallback() {
    @Override
    public Action consume(MessageToken messageToken) {
        return null;
    }
});
```

后端服务器应用开发．上行



- 服务器关心的topic
  - 设备状态上报

getDeviceStatusTopic()

/as/mqtt/status/{pk}/#

  - (高级)设备属性上报

getDevicePropertyPostTopic()

/sys/{pk}/{dn}/thing/event/property/post

  - (高级)设备事件上报

getDeviceEventPostTopic()

/sys/{pk}/{dn}/thing/event/$(tls.event.identifer)/post

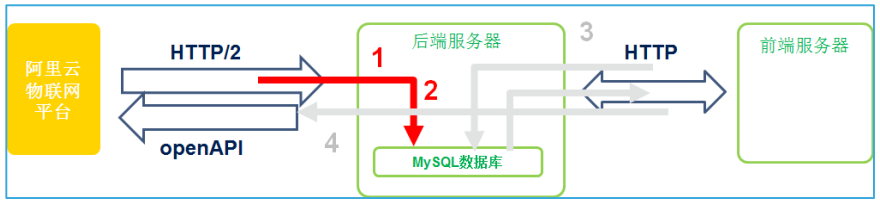
  - 设备生命周期上报
  - .....

Topic.java

```
19    // 设备状态上报  /as/mqtt/status/{productKey}/#
20        public String getDeviceStatusTopic(){
21            return new StringBuffer().append("/as/mqtt/status/").append(productKey).append("/#").toString();
22        }
23
24        //  高级版设备属性上报 sys/{productKey}/{deviceName}/thing/event/property/post
25        public String getDevicePropertyPostTopic(){
26            return new StringBuffer().append("/").append(productKey).append("/#").append("/thing/event/property/post").toString();
27        }
28
29        // 高级版设备事件上报 /sys/{productKey}/${deviceName}/thing/event/${tsl.event.identifer}/post
30        public String getDeviceEventPostTopic(){
31            return new StringBuffer().append("/").append(productKey).append("/#").append("/thing/event").append("/#").toString();
32        }
33
34        // 设备设备生命周期上报 /{productKey}/${deviceName}/thing/lifecycle
35        public String getDeviceLifecyclePostTopic(){
36            return new StringBuffer().append("/").append(productKey).append("/#").append("/thing/lifecycle").toString();
37        }
39        //基础版温湿度上报 /{productKey}/#/tempHumUpload
40        public String getTempHumUploadTopic() {
41            return new StringBuffer().append("/").append(productKey).append("/#").append("/tempHumUpload").toString();
42        }
43        //基础版温度报警上报  /{productKey}/#/tempAlarm
44        public String getTempAlarmTopic() {
45            return new StringBuffer().append("/").append(productKey).append("/#").append("/tempAlarm").toString();
46        }
47    }
```

life.augmented

阿里云



后端服务器应用开发 . 上行

- 监听，设置消息接收接口
  - 建立连接时，需要提供消息接收接口，用于处理未设置回调的消息

DeviceMessageRegister.java

```
87        //属性上报  topic:   /productKey/#/thing/event/property/post
88        client.setMessageListener(topic.getDevicePropertyPostTopic(), new MessageCallback() {
89            @Override
90            public Action consume(MessageToken messageToken) {
91                Message m = messageToken.getMessage();
92                logger.info("receive 属性上报" + new String(messageToken.getMessage().getPayload()));
93                DevicePropertyPost devicePropertyPost = JSON.parseObject(new String(m.getPayload()), DeviceProper
94                Boolean isUpdateDeviceProperty = deviceInfoService.updateOrInsertDeviceProperty(devicePropertyPos
95                Boolean isUpdateDevicePropHistoryLog = devicePropHistoryLogService.insertDevicePropHistoryLog(dev
96                if (isUpdateDeviceProperty && isUpdateDevicePropHistoryLog) {
97                    return Action.CommitSuccess;
98                }
99                return Action.CommitFailure; }});
```

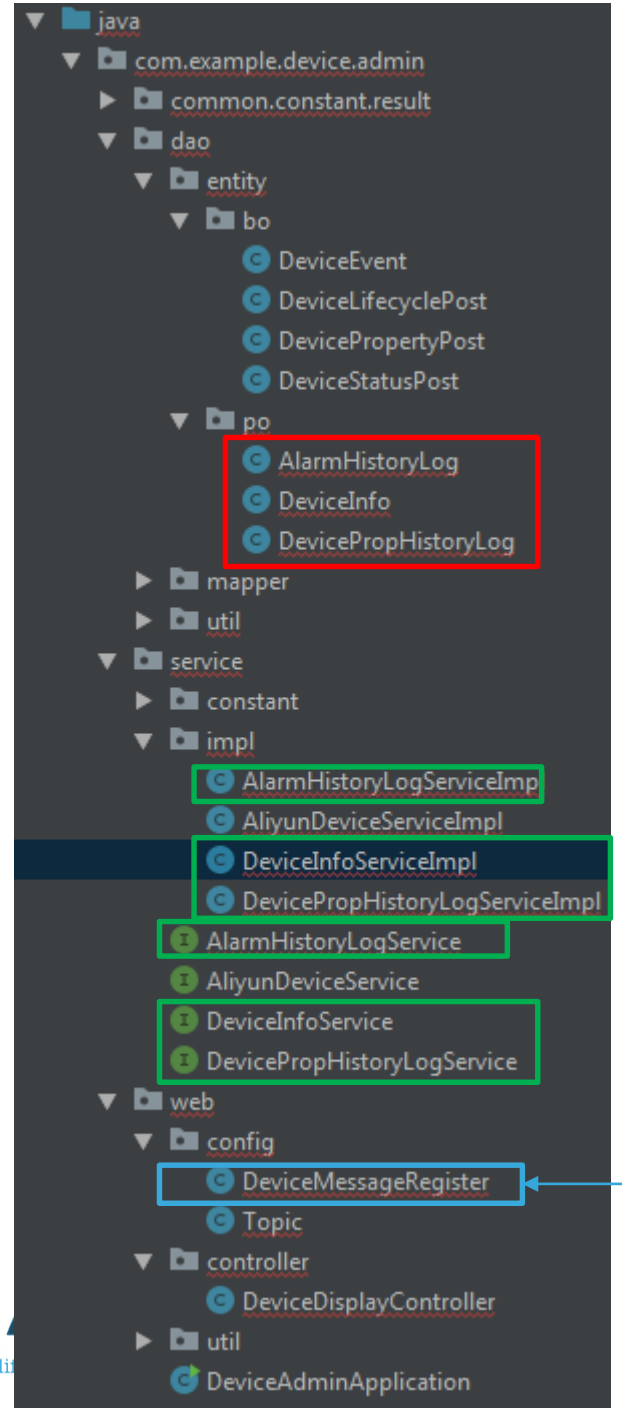
DeviceMessageRegister.java

```
203       client.connect(messageToken -> {
204           Message m = messageToken.getMessage();
205           byte[] payload = m.getPayload();
206           System.out.println("receive message from " + m);
207           System.out.println("m.getTopic() = " + m.getTopic());
208           return MessageCallback.Action.CommitSuccess;
209       });
```

- 根据不同主题，通过service层，操作数据库
  - client.setMessageListener (topic.xxxxxx(), new MessageCallback()

```
//更新设备信息表
Boolean isUpdateDeviceProperty = deviceInfoService.updateOrInsertDeviceProperty(devicePropertyPost);
//更新设备历史温湿度表
Boolean isUpdateDevicePropHistoryLog = devicePropHistoryLogService.insertDevicePropHistoryLog(devicePropertyPost);
```

上行数据，写入数据库



web 层

接口 → 子类
实现抽象方法

service 层

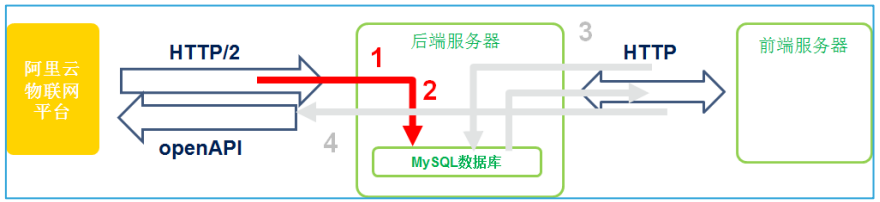
mapper下面的方法
←→ SQL语句

dao 层

po下面三个java表类

<DeviceMessageRegister.java>
afterPropertiesSet()
→ **deviceInforService**.xxx
→ **devicePropHistoryLogService**.xxx
→ **alarmHistoryLogService**.xxx

DeviceInforServieImpl.java
→ **deviceInforMapper**.xxx
DevicePropHistoryLogServiceImpl.java
→ **devicePropHistoryLogMapper**.xxx
AlarmHistoryLogServiceImpl.java
→ **alarmHistoryLogMapper**.xxx



alarm_history_log
id
device_name
alarm_temperature
temp_threshold
gmt_create

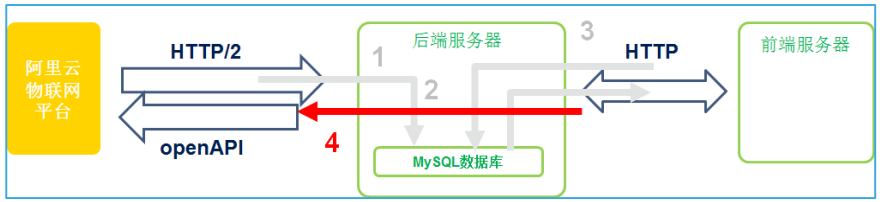
interface	抽象方法	
AlarmHistoryLogService	listAlarmHistoryLogs	
	insertAlarmHistoryLogByAliyunQuery	
	insertAlarmHistoryLogByMysqlQuery	
class	**覆写interface里的抽象方法**	
AlarmHistoryLogServiceImpl	listAlarmHistoryLogs	alarmHistoryLogMapper.selectByExample(…)
	insertAlarmHistoryLogByAliyunQuery	alarmHistoryLogMapper.insert(…)
	insertAlarmHistoryLogByMysqlQuery	

device_prop_history_log
id
device_name
current_temperature
current_humidity
gmt_creat

interface	抽象方法	
DevicePropHistoryLogService	listDevicePropHistoryLogs	
	insertDevicePropHistoryLog	
class	**覆写interface里的抽象方法**	
DevicePropHistoryLogServiceImpl	listDevicePropHistoryLogs	devicePropHistoryLogMapper.selectByExample(…)
	insertDevicePropHistoryLog	devicePropHistoryLogMapper.insert(…)

device_info
id
device_name
current_temperature
current_humidity
temp_threshold
alarm_status
last_alarm_date
last_online_date
connect_status
gmt_update

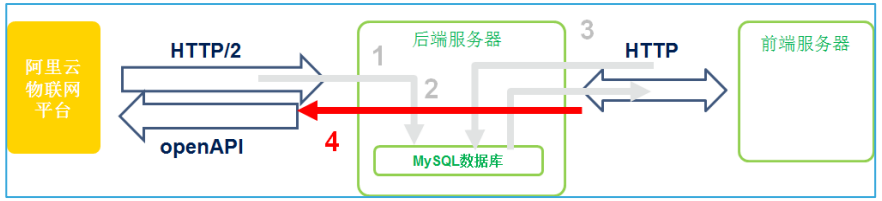
interface	抽象方法	
DeviceInfoService	updateStatus	
	getDeviceInfoByDeviceName	
	listDeviceNames	
	updateOrInsertDeviceProperty	
	insertOrDelete	
	updateAlarmStatus	
	updateTempTheshold	
class	**覆写interface里的抽象方法**	
DeviceInfoServiceImpl	updateStatus	deviceInfoMapper. insertOrUpdateStatus (…)
	getDeviceInfoByDeviceName	deviceInfoMapper.selectOne(…)
	listDeviceNames	deviceInfoMapper.selectAll(…)
	updateOrInsertDeviceProperty	deviceInfoMapper. insertOrUpdateDeviceProperty (…)
	insertOrDelete	deviceInfoMapper. insert(…) deviceInfoMapper.delete(…)
	updateAlarmStatus	deviceInfoMapper.updateByExampleSelective(…)
	updateTempTheshold	



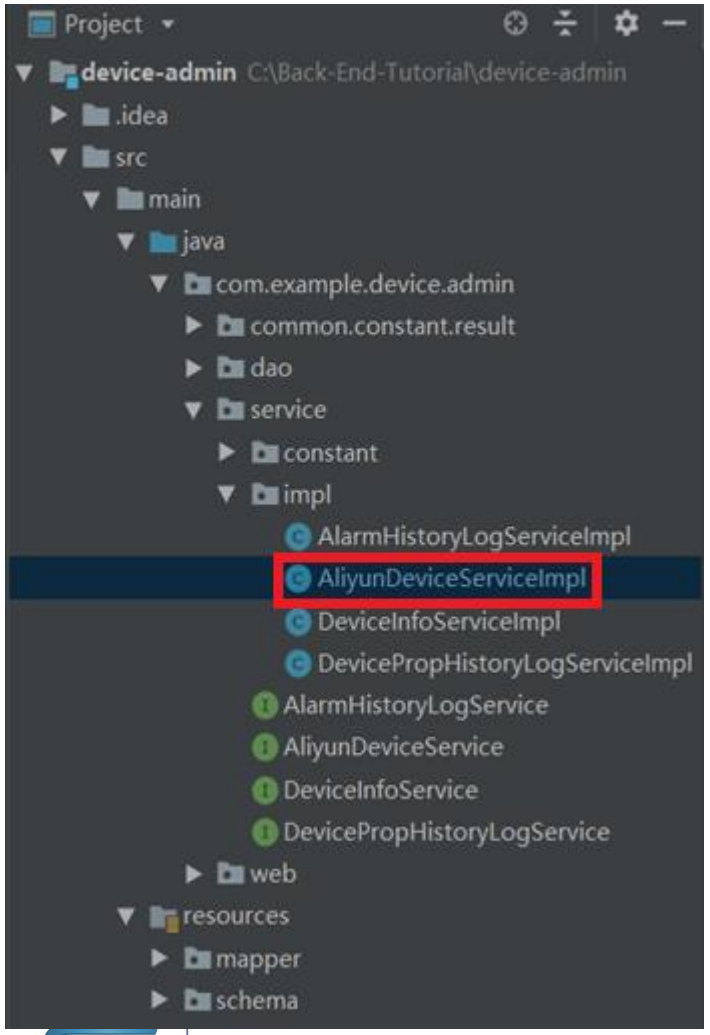
后端服务器应用开发．下行

- 调用云端openAPI，可进行产品管理、设备管理、topic管理、数据流转规则设置、消息通信等
  - 通过向服务端地址发送HTTP(S) GET/POST请求，按照API接口说明封装，进行API调用
  - API功能列表，及调用方式
- SDK下载：**aliyun-java-sdk-iot/**
  - 在项目中由maven负责依赖jar包的自动下载
- SDK的使用
  - 初始化SDK客户端
  - 封装request
  - 调用getAcsResponse发出request并收取response

API列表	本项目使用的	
产品管理		
设备管理	SetDeviceProperty,	设置设备属性
	QueryDevicePro,	查询设备属性
	InvokeThingService	调用设备的服务
分组管理		
topic管理		
消息通信	Pub	向指定主题发消息



后端服务器应用开发．下行



- 建立连接 @ getClient()

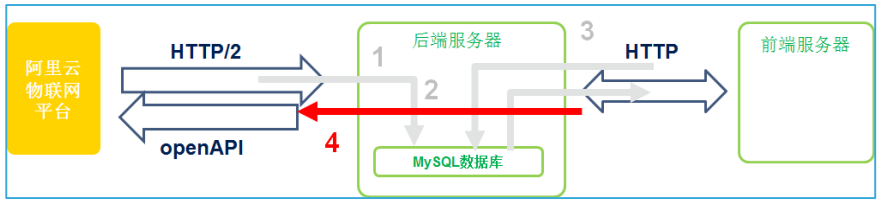
```
IClientProfile profile = DefaultProfile.getProfile(regionId, accessKeyID, accessKeySecret);
DefaultProfile.addEndpoint(regionId, regionId, productCode,
        domain);
// 初始化client
client = new DefaultAcsClient(profile);
```

- 以封装的PubRequest类调用getAcsResponse方法

```
String topic ="/"+productKey+"/"+deviceName+"/"+identifier;
PubRequest request = new PubRequest();
request.setProductKey(productKey);
request.setTopicFullName(topic);
request.setMessageContent(Base64.encodeBase64String(msg.getBytes()));
request.setAcceptFormat(FormatType.JSON);
request.setQos(1);
PubResponse acsResponse =client.getAcsResponse(request);
```

- 以封装的invokeThingService类调用getAcsResponse方法
- 以封装的setDeviceProperty类调用getAcsResponse方法
- 以封装的queryDevicePro类调用getAcsResponse方法

life.augmented 阿里云

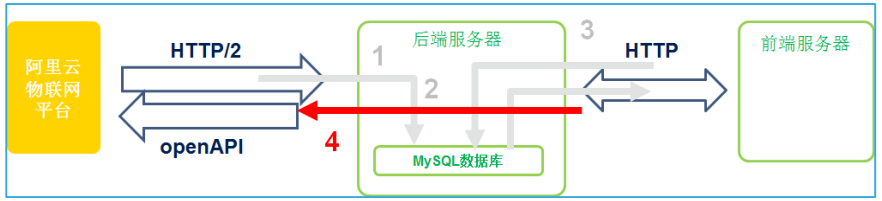


后端服务器应用开发．下行

- 前端浏览器下发【指令】给节点设备

下发的指令	与前端交互的API	与Iot平台交互的openAPI	
		基础版	高级版
设置温度阈值	setDeviceProperty	pub	setDeviceProperty
解除报警状态	clearAlarm	pub	invokeThingService

openAPI列表	本项目使用到的		
产品管理			
设备管理	setDeviceProperty,	设置设备属性	向高级版设备下发控制命令：设置温度阈值
	queryDevicePro,	查询设备属性	收到报警事件的上行消息后，会主动查询设备当前的温度值，并存入数据库
	invokeThingService	调用设备的服务	向高级版设备下发控制命令：解除警报
分组管理			
topic管理			
消息通信	pub	向指定主题发消息	向基础版设备下发两个控制命令



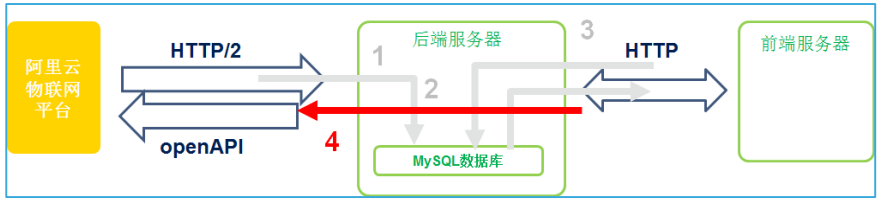
接收”设置阈值”命令并转发

- 透传方式，设置设备温度阈值
  - DeviceDisplayController.java
  - BaseResp setDeviceProperty(String deviceName, Float temThreahold)

```
//透传方式修改设备报警阈值 如果是基础版，请启用此段代码 如果是高级版，请注释此段代码
@RequestMapping(value = "/api/v1/device/setDeviceProperty",method = RequestMethod.GET)
public @ResponseBody BaseResp setDeviceProperty(String deviceName, Float tempThreshold) {
    Boolean b = null;
    try {
        b = aliyunDeviceService.pub(deviceName,  identifier: "tempThresholdSet",
                Character.toString((char)tempThreshold.floatValue()));
        if(b){
            deviceInfoService.updateTempThreshold(deviceName,tempThreshold);
        }
    } catch (ClientException e) {
        e.printStackTrace();
    }
    Map<String,Boolean> map=new HashMap();
    map.put("isSuccess",b);
    return new BaseResp<>(ResultStatus.SUCCESS,map); }
```

- Alink方式，设置设备温度阈值
  - DeviceDisplayController.java
  - BaseResp setDeviceProperty(String deviceName, Float temThreahold)

```
//Alink方式修改设备报警阈值，如果是高级版，请启用此段代码 如果是基础版，请注释此段代码
@RequestMapping(value = "/api/v1/device/setDeviceProperty",method = RequestMethod.GET)
public @ResponseBody BaseResp setDeviceProperty(String deviceName, Float tempThreshold) {
    Boolean b = null;
    try {
        b = aliyunDeviceService.setDeviceProperty(deviceName, "TempThreshold", tempThreshold);
    } catch (ClientException e) {
        e.printStackTrace();
    }
    Map<String,Boolean> map=new HashMap();
    map.put("isSuccess",b);
    return new BaseResp<>(ResultStatus.SUCCESS,map); }
```



接收”解除警报”命令并转发

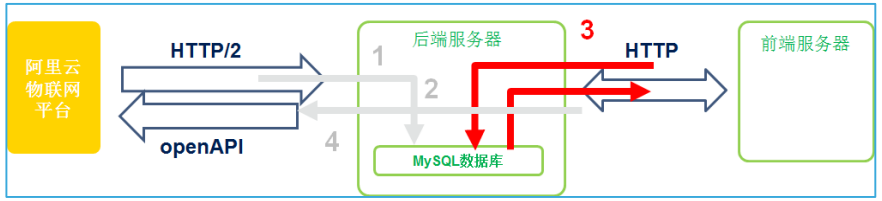
- 透传方式，解除设备报警状态
  - DeviceDisplayController.java
  - BaseResp clearAlarm(String deviceName)

```
//透传方式设备报警状态取消 如果是基础版，请启用此段代码 如果是高级版，请注释此段代码
@RequestMapping(value = "/api/v1/device/clearAlarm",method = RequestMethod.GET)
public @ResponseBody BaseResp clearAlarm(String deviceName) {
    Boolean b = null;
    try {
        b = aliyunDeviceService.pub(deviceName, identifier: "clearAlarm", Character.toString((char)1));
        if(b){
            deviceInfoService.updateAlarmStatus(deviceName, b: false);
        }
    } catch (ClientException e) {
        e.printStackTrace();
    }
    Map<String,Boolean> map=new HashMap();
    map.put("isSuccess",b);
    return new BaseResp<>(ResultStatus.SUCCESS,map); }
```

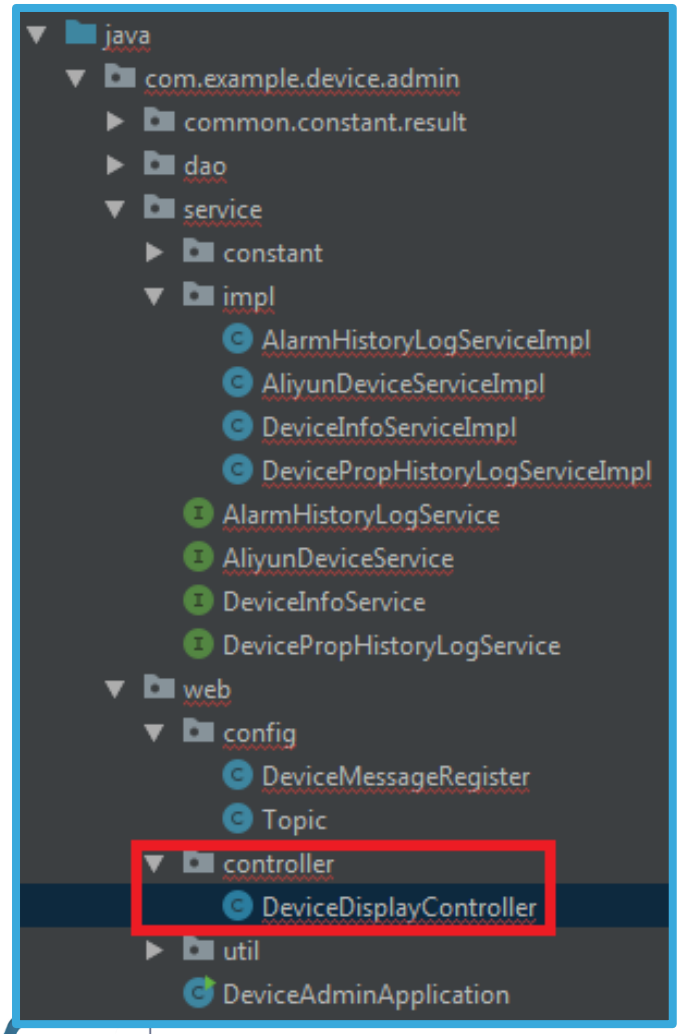
- Alink方式，解除设备报警状态
  - DeviceDisplayController.java
  - BaseResp clearAlarm(String deviceName)

```
/*
  //Alink方式设备报警状态取消 如果是高级版，请启用此段代码 如果是基础版，请注释此段代码
  @RequestMapping(value = "/api/v1/device/clearAlarm",method = RequestMethod.GET)
   public @ResponseBody BaseResp clearAlarm(String deviceName) {
       Boolean b = null;
       try {
           b = aliyunDeviceService.invokeThingService(deviceName, "ClearAlarm", "{}");
           if(b){
               deviceInfoService.updateAlarmStatus(deviceName,false);
           }
       } catch (ClientException e) {
           e.printStackTrace();
       }
       Map<String,Boolean> map=new HashMap();
       map.put("isSuccess",b);
       return new BaseResp<>(ResultStatus.SUCCESS,map);}
   */
```

life.augmented | 阿里云



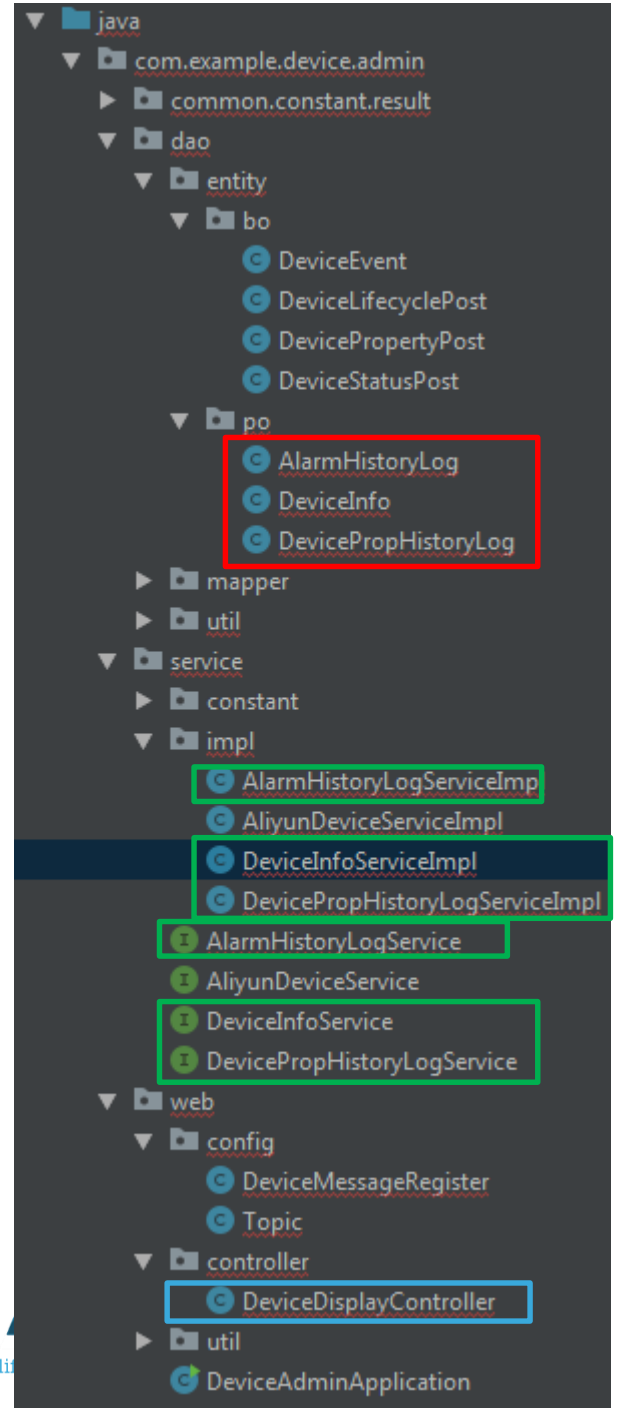
响应前端请求，查询数据库



- DeviceDisplayController.java
  - 和前端服务器的交互API定义
  - GET/api/v1/device/listDeviceName
    （查询所有设备名）
  - GET/api/v1/device/queryAlarmHistoryLogs
    （查询一段报警数据）
  - GET/api/v1/device/queryDeviceProp
    （获取设备属性）
  - GET/api/v1/device/queryDevicePropHistoryLogs
    （查询一段时间温湿度数据）
  - GET/api/v1/device/setDeviceProperty
    （修改方式设备报警阀值）
  - GET/api/v1/device/clearAlarm
    （设备报警状态取消）

查询数据库

响应前端请求，查询数据库



web 层

接口 → 子类
实现抽象方法

service 层

mapper下面的方法
←→ SQL语句

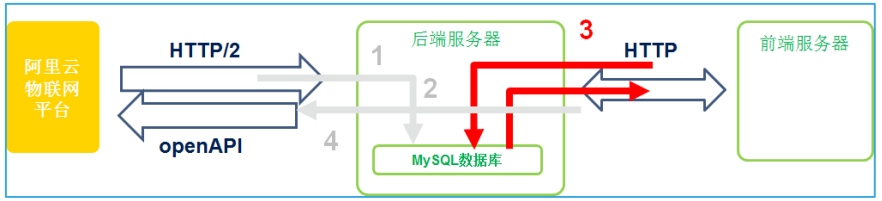
dao 层

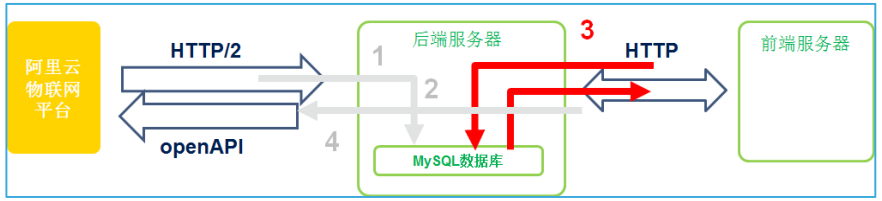
<DeviceDisplayController.java>
→ **deviceInforService**.xxx
→ **devicePropHistoryLogService**.xxx
→ **alarmHistoryLogService**.xxx

DeviceInforServieImpl.java
→ **deviceInforMapper**.xxx

DevicePropHistoryLogServiceImpl.java
→ **devicePropHistoryLogMapper**.xxx

AlarmHistoryLogServiceImpl.java
→ **alarmHistoryLogMapper**.xxx





响应前端请求，查询数据库

	获取设备属性	查询所有设备名	查询温湿度历史数据	查询报警事件历史数据
地址映射	/api/v1/device/queryDeviceProp	/api/v1/device/listDeviceName	/api/v1/device/queryDevicePropHistoryLogs	/api/v1/device/queryAlarmHistoryLogs
Web/方法	queryDeviceProp	listDeviceNames	queryDevicePropHistoryLogs	queryAlarmHistoryLogs
Service/接口	DeviceInfoService		devicePropHistoryLogService	alarmHistoryLogService
Service/实现	DeviceInfoService**Impl**		devicePropHistoryLogService**Impl**	alarmHistoryLogService**Impl**
方法覆写	getDeviceInfoByDeviceName	listDeviceNames	listDevicePropHistoryLogs	listAlarmHistoryLogs
Dao/mapper	deviceInfoMapper		DevicePropHistoryLogMapper	alaramHistoryLogMapper
继承**mapper**	BaseMapper			
封装的方法	selectOne	selectAll	selectByExample	selectByExample

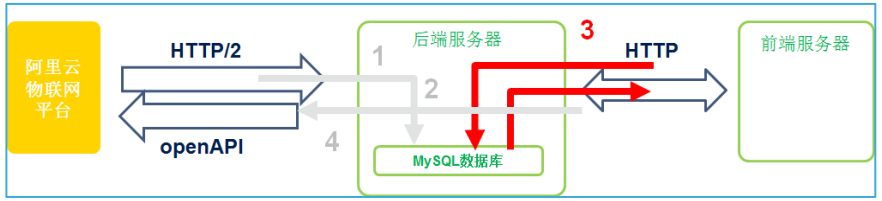
举例：获取设备属性

DeviceDisplayController.java

```
42      //获取设备属性(当前温湿度及报警阀值及设备在线状态)
43      @RequestMapping(value = "/api/v1/device/queryDeviceProp",method = RequestMethod.GET)
44      public @ResponseBody  BaseResp<DeviceInfo> queryDeviceProp(String deviceName) {
45          DeviceInfo deviceInfo=deviceInfoService.getDeviceInfoByDeviceName(deviceName);
46          return new BaseResp<>(ResultStatus.SUCCESS,deviceInfo);
47      }
```

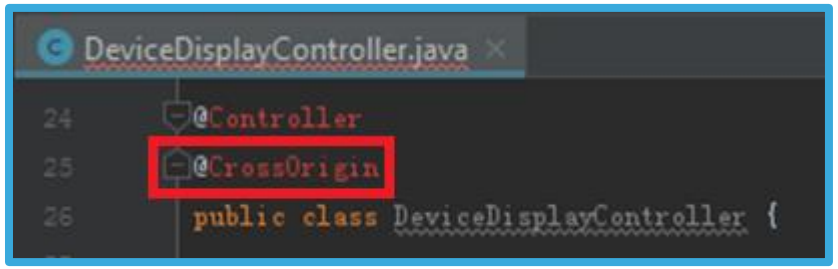
DeviceInfoServiceImpl.java

```
51      @Override
52      public DeviceInfo getDeviceInfoByDeviceName(String deviceName) {
53          DeviceInfo deviceInfo=new DeviceInfo();
54          deviceInfo.setDeviceName(deviceName);
55          DeviceInfo deviceInfo1 = deviceInfoMapper.selectOne(deviceInfo);
56          return deviceInfo1;
57      }
```



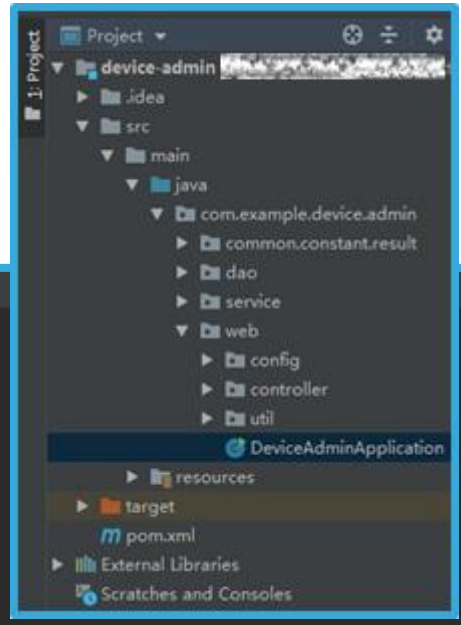
前后端交互：跨域请求

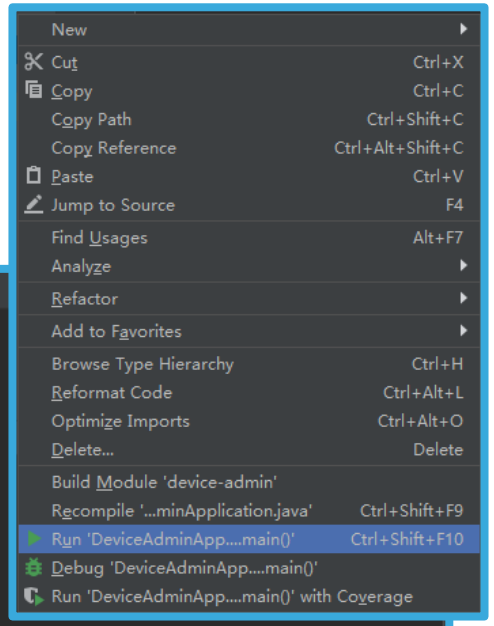
- 跨域请求
  - 发起请求的域，与该请求指向的资源所在的域不一样
  - “域”：协议 + 域名 + 端口号
- 浏览器处于安全考虑，对跨域请求的限制
- 跨域解决方法 – CORS (Cross-region resource sharing)
  - 新增一组HTTP首部字段，运行服务端声明哪些源站有权限访问哪些资源
  - 允许浏览器向声明了CORS的跨域服务器发出HTTPRequest请求
- Spring配置CORS
  - 使用@CrossOrigin注解，标注在需要被配置的类或方法前



（本地）运行IDEA项目

- 选中DeviceAdminApplication.java
- 鼠标右键 → 菜单中选择 “Run”

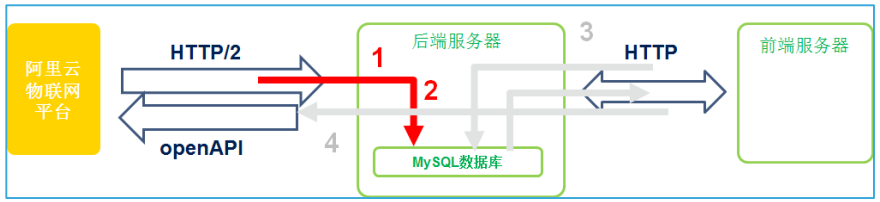




开始启动……

约5启动完毕

和阿里云IoT连接成功



本地测试.与IoT平台

- 节点设备连接阿里云IoT平台，定期上报数据（设备属性）
- 查询数据库内容

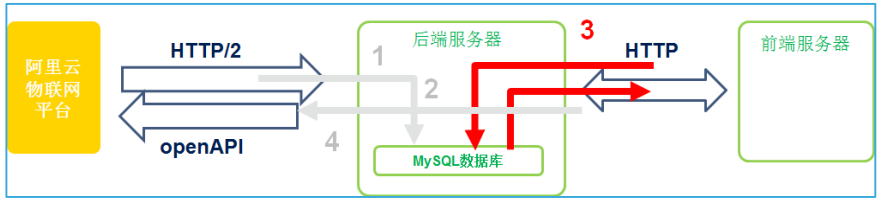
IDEA运行窗口：20:49:06 接到“设备状态变化上报”

```
receive msg, messageId:1113785565527945216, data size: 256
receive 设备状态变化上报{"lastTime":"2019-04-04 20:49:06.599","utcLastTime":"2019-04-04T12:49:06.599Z","clientIp":"112.65.61.75","utcTime":"2019-04-04T12:49:06.614Z","time":"2019-04-04 20:49:0
AppHikariCP - Starting...
AppHikariCP - Start completed.
==>  Preparing: SELECT id,device_name,current_temperature,current_humidity,temp_threshold,alarm_status,connect_status,gmt_update,last_alarm_date,last_online_date FROM device_info WHERE ( ( dev
==> Parameters: 2019CT_case2_node001(String)
<==      Total: 1
==>  Preparing: UPDATE device_info SET device_name = ?,connect_s
==> Parameters: 2019CT_case2_node001(String), 在线(String), 2019                                                                    p), 2960(Integer)
receive msg, messageId:1113785568682056192, data size: 327
receive 属性上报{"deviceType":"CustomCategory","iotId":"UeUyHwfqHb110wTX4kpF000100","productKey":"a1zPqJ3NYxJ","gmtCreate":1554382147353,"deviceName":"2019CT_case2_node001","items":{"CurrentHu
==>  Preparing: INSERT INTO device_info(device_name,current_temperature,current_humidity,temp_threshold) VALUES(?,?,?,?) ON DUPLICATE KEY UPDATE device_name=?, current_temperature=?, current_h
```

收到“属性上报”，准备写数据库

查数据库

```
mysql> select id,device_name,g                                                    device_info
+------+----------------------+---------------------+
| id   | device_name          | gmt_update          |
+------+----------------------+---------------------+
|   16 | smartthermometer     | 2019-04-03 17:39:33 |
| 2899 | 2019CT_case1_node001 | 2019-04-03 16:21:36 |
| 2960 | 2019CT_case2_node001 | 2019-04-04 20:49:06 |
+------+----------------------+---------------------+
3 rows in set (0.00 sec)
```

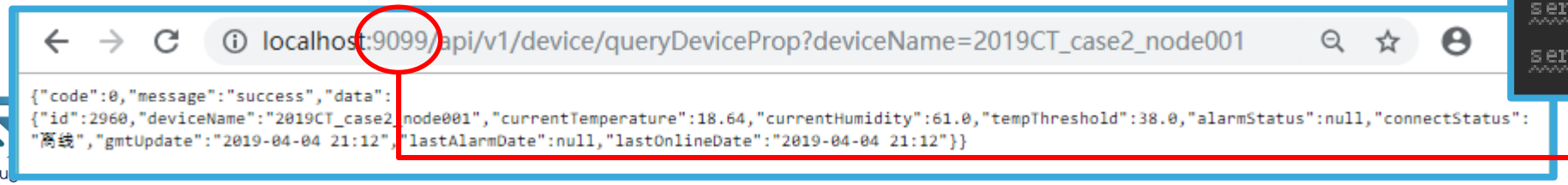


本地测试.与前端交互

- 可通过浏览器模拟测试前端发来的get请求

和前端服务器的交互API	
GET/api/v1/device/listDeviceName	查询所有设备名
GET/api/v1/device/queryDeviceProp	获取设备属性
GET/api/v1/device/queryDevicePropHistoryLogs	查询一段时间温湿度数据
GET/api/v1/device/queryAlarmHistoryLogs	查询一段报警数据
GET/api/v1/device/setDeviceProperty	修改方式设备报警阀值
GET/api/v1/device/clearAlarm	设备报警状态取消

- 举例：查询所有设备名、查询2019CT_case2_node001的设备属性



第四章.第二节 内容简介

- 认识后端框架
  - Mysql数据库介绍
  - MyBatis框架
  - SpringBoot框架
  - Maven介绍
- 初始化并运行第一个后端项目
  - 第一个springboot项目详解
  - 从无到有搭建第一个springboot项目
- 应用系统开发
  - 存储数据库设计
  - 配置服务端订阅
  - 应用系统与iot平台连接
  - 调用阿里云物联网接口
  - 接口编写
  - 跨域请求
- 应用调试与部署
  - 项目打包
  - 项目部署

项目打包与部署

IDEA IDE内项目运行
（开发电脑上本地运行）

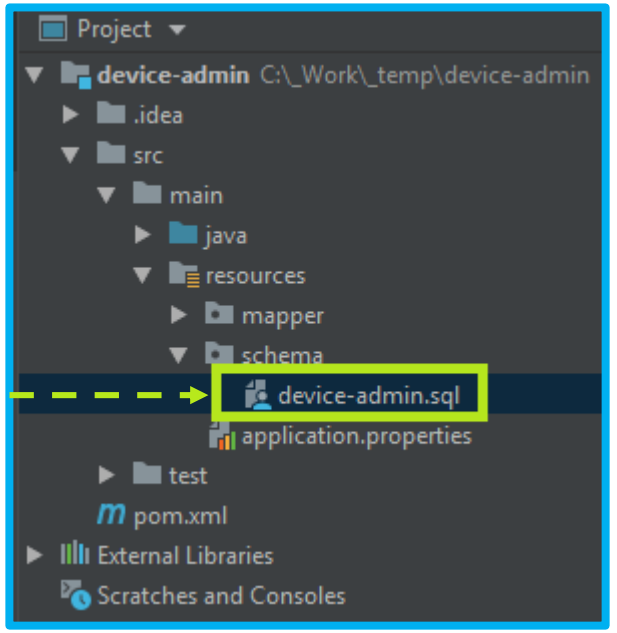
生成jar包运行
（开发电脑上本地运行）

修改数据库配置，重新生成jar包

上传至公网服务器

在公网服务器上启动

在服务器上手动建立
device_admin数据库和
三个表
(参照device-admin.sql)

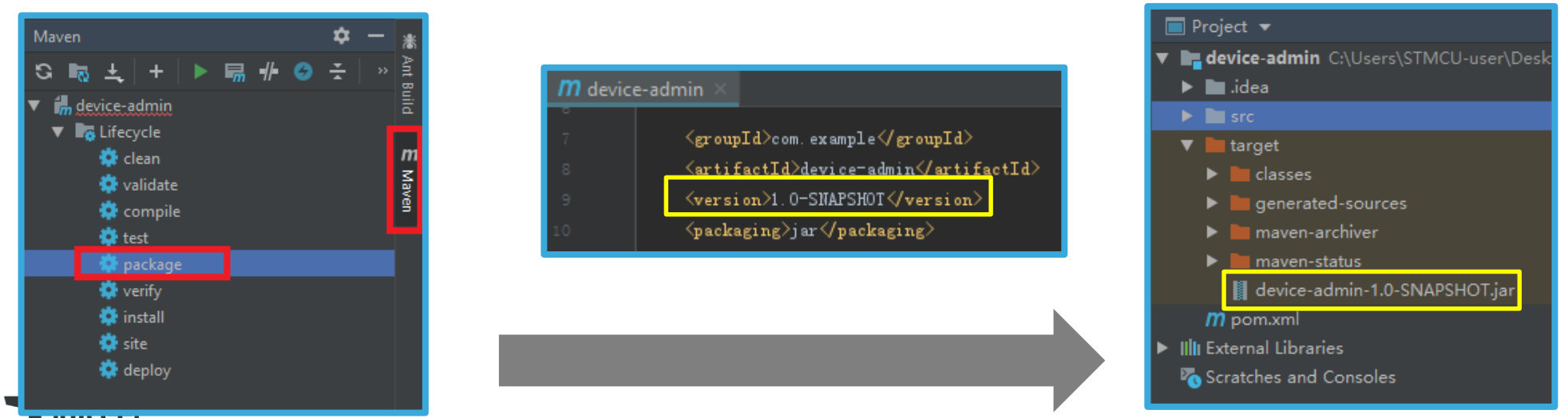


项目打包

- 【新增】依赖插件

依赖的插件包	本项目要用到它的什么功能
spring-boot-maven-plugin	将SpringBoot应用打包为可执行的jar或war文件

- 项目打包
  - 可在<pom.xml>中配置jar包名称：1.0-SNAPSHOP



项目（本地）部署：Windows环境

- 切换到jar包目录，命令行运行
  - java -jar device-admin-1.0-SNAPSHOT.jar

```
C:\Users\STMCU-user\Desktop\Back-End-Tutorial\device-admin\target>java -jar device-admin-1.0-SNAPSHOT.jar

  .   ____          _            __ _ _
 /\\ / ___'_ __ _ _(_)_ __  __ _ \ \ \ \
( ( )\___ | '_ | '_| | '_ \/ _` | \ \ \ \
 \\/  ___)| |_)| | | | | || (_| |  ) ) ) )
  '  |____| .__|_| |_|_| |_\__, | / / / /
 =========|_|==============|___/=/_/_/_/
 :: Spring Boot ::        (v2.1.1.RELEASE)

2019-04-04 21:34:49.118  INFO 8600 --- [           main] c.e.d.admin.web.DeviceAdminApplication   : Starting DeviceAdminApplication v1.0-SNAPSHOT on STMCU with PID 8600 (C:\Users\STMCU-user\Desktop\Back-End-Tutorial\device-admin\target\device-admin-1.0-SNAPSHOT.jar started by STMCU-user in C:\Users\STMCU-user\Desktop\Back-End-Tutorial\device-admin\target)
```

```
2019-04-04 21:34:56.474  INFO 8600 --- [           main] o.s.b.w.embedded.tomcat.TomcatWebServer  : Tomcat started on port(s): 9099 (http) with context path ''
2019-04-04 21:34:56.479  INFO 8600 --- [           main] c.e.d.admin.web.DeviceAdminApplication   : Started DeviceAdminApplication in 7.931 seconds (JVM running for 8.571)
2019-04-04 21:35:05.705  INFO 8600 --- [client-thread-0] c.a.o.iot.api.http2.IotHttp2Client       : update connection count, current count 1, expected count 8
2019-04-04 21:35:05.705  INFO 8600 --- [client-thread-0] c.a.o.i.a.h.c.i.ConnectionManagerImpl    : connecting to 1399405922263114.iot-as-http2.cn-shanghai.aliyuncs.com/139.196.67.176:443
2019-04-04 21:35:05.707  INFO 8600 --- [ntLoopGroup-2-2] c.a.o.i.a.h.netty.NettyHttp2Initializer : init http2 handler. enable SSL : true
2019-04-04 21:35:05.720  INFO 8600 --- [ntLoopGroup-2-2] c.a.o.iot.api.http2.IotHttp2Client       : connection status changed, connection: bcc4b09d, status: CREATING
2019-04-04 21:35:05.722  INFO 8600 --- [client-thread-0] c.a.o.i.a.h.c.i.ConnectionManagerImpl    : connecting to 1399405922263114.iot-as-http2.cn-shanghai.aliyuncs.com/139.196.67.176:443
```

部署到公网服务器

- 修改数据库配置(用户名、密码)，重新生成jar包
- 上传到公网服务器
  - 工具：gitbash/scp
- 远程登录公网服务器，工具：xshell

application.properties

```
#连接数据库url
spring.datasource.url=jdbc:mysql://localhost:3306/device_admin?useUnicode=true\
  &characterEncoding=UTF-8&autoReconnect=true\
  &useSSL=false&zeroDateTimeBehavior=convertToNull&serverTimezone=Asia/Shanghai
#连接数据库用户名
spring.datasource.username=root
#连接数据库密码
spring.datasource.password=***
#设置com.example.device.admin包下日志级别debug
logging.level.com.example.device.admin=DEBUG
```

1

```
STMCU-user@STMCU MINGW64 ~/Desktop/Back-End-Tutorial/device-admin/target
$ scp device-admin-1.0-SNAPSHOT.jar root@[redacted]:/root
root@[redacted]'s password:
device-admin-1.0-SNAPSHOT.jar                 100%   33MB   1.0MB/s   00:33
```

2

```
Xshell 6 (Build 0121)
Copyright (c) 2002 NetSarang Computer, Inc. All rights reserved.

Type `help' to learn how to use Xshell prompt.
[C:\~]$

Connecting to 47.94.214.229:22...
Connection established.
To escape to local shell, press 'Ctrl+Alt+]'.

WARNING! The remote SSH server rejected X11 forwarding request.
Last login: Thu Apr  4 21:55:55 2019 from 183.193.161.110

Welcome to Alibaba Cloud Elastic Compute Service !

[root@iz2zeczwmztkrnzjqhn0dqz ~]# ls
defrag~  defraz~  device-admin-1.0-SNAPSHOT.jar  mosquitto.log.1884  nvm
[root@iz2zeczwmztkrnzjqhn0dqz ~]#
```

3

life.augmented

阿里云

在公网服务器上创建数据库和表

- 创建数据库和表，工具：xshell

  - 启动服务器上数据库服务
    - mysql –u root -p
  - 创建数据库
    - create database device_admin
  - 选择自己的数据库
    - use device_admin
  - 建立三个数据表
    - 直接使用device_admin.sql

    中语句即可

```
[root@iz2zeczwmztkrnzjqhn0dqz ~]# mysql -u root -p
Enter password:
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 68
Server version: 5.7.22 MySQL Community Server (GPL)

Copyright (c) 2000, 2018, Oracle and/or its affiliates. All rights reserved.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql> show databases;
+--------------------+
| Database           |
+--------------------+
| information_schema |
| device_admin       |
| mysql              |
| performance_schema |
| sttrainmanager     |
| sys                |
| yunserver          |
+--------------------+
7 rows in set (0.00 sec)
```

```
mysql> use device_admin
Reading table information for completion of table and column names
You can turn off this feature to get a quicker startup with -A

Database changed
mysql> show tables;
+-------------------------+
| Tables_in_device_admin  |
+-------------------------+
| alarm_history_log       |
| device_info             |
| device_prop_history_log |
+-------------------------+
3 rows in set (0.00 sec)

mysql>
```

在公网服务器上运行

- 启动jar包运行

```
[root@iz2zeczwmztkrnzjqhn0dqz ~]# java -jar device-admin-4.0-SNAPSHOT.jar

  .   ____          _            __ _ _
 /\\ / ___'_ __ _ _(_)_ __  __ _ \ \ \ \
( ( )\___ | '_ | '_| | '_ \/ _` | \ \ \ \
 \\/  ___)| |_)| | | | | || (_| |  ) ) ) )
  '  |____| .__|_| |_|_| |_\__, | / / / /
 =========|_|==============|___/=/_/_/_/
 :: Spring Boot ::        (v2.1.1.RELEASE)

2019-04-08 14:11:12.420  INFO 20910 --- [           main] c.e.d.admin.web.DeviceAdminApplication   : Starting DeviceAdminApplication v4.0-SNAPSHOT on
 iz2zeczwmztkrnzjqhn0dqz with PID 20910 (/root/device-admin-4.0-SNAPSHOT.jar started by root in /root)
2019-04-08 14:11:12.423 DEBUG 20910 --- [           main] c.e.d.admin.web.DeviceAdminApplication   : Running with Spring Boot v2.1.1.RELEASE, Spring
v5.1.3.RELEASE
2019-04-08 14:11:12.425  INFO 20910 --- [           main] c.e.d.admin.web.DeviceAdminApplication   : No active profile set, falling back to default p
rofiles: default
2019-04-08 14:11:13.549  WARN 20910 --- [           main] t.m.s.mapper.ClassPathMapperScanner      : No MyBatis mapper was found in '[com.example.dev
ice.admin.web]' package. Please check your configuration.
2019-04-08 14:11:13.557  WARN 20910 --- [           main] o.m.s.mapper.ClassPathMapperScanner      : No MyBatis mapper was found in '[com.example.dev
ice.admin.web]' package. Please check your configuration.
2019-04-08 14:11:14.427  INFO 20910 --- [           main] o.s.b.w.embedded.tomcat.TomcatWebServer  : Tomcat initialized with port(s): 9099 (http)
2019-04-08 14:11:14.456  INFO 20910 --- [           main] o.apache.catalina.core.StandardService   : Starting service [Tomcat]
2019-04-08 14:11:14.456  INFO 20910 --- [           main] org.apache.catalina.core.StandardEngine  : Starting Servlet Engine: Apache Tomcat/9.0.13
2019-04-08 14:11:14.470  INFO 20910 --- [           main] o.a.catalina.core.AprLifecycleListener   : The APR based Apache Tomcat Native library which
 allows optimal performance in production environments was not found on the java.library.path: [/usr/java/packages/lib/amd64:/usr/lib64:/lib64:/lib:/
usr/lib]
2019-04-08 14:11:14.561  INFO 20910 --- [           main] o.a.c.c.C.[Tomcat].[localhost].[/]       : Initializing Spring embedded WebApplicationConte
xt
2019-04-08 14:11:14.561  INFO 20910 --- [           main] o.s.web.context.ContextLoader            : Root WebApplicationContext: initialization compl
eted in 2067 ms
2019-04-08 14:11:15.466  INFO 20910 --- [           main] c.a.o.i.a.h.c.i.ConnectionManagerImpl    : [ConnectionManagerImpl]initialize netty
2019-04-08 14:11:15.639  INFO 20910 --- [           main] c.a.o.i.a.h.c.i.ConnectionManagerImpl    : connecting to 1399405922263114.iot-as-http2.cn-s
hanghai.aliyuncs.com/106.15.83.13:443
```

2秒左右启动完成

在公网服务器上运行

- 启动jar包运行

```
2019-04-08 14:11:26.429  INFO 20910 --- [ntLoopGroup-2-4] c.a.o.iot.api.http2.IotHttp2Client       : receive setting, connection: 10936960, subscript
ion count : 8
2019-04-08 14:11:26.433  INFO 20910 --- [ntLoopGroup-2-3] c.a.o.iot.api.http2.IotHttp2Client       : connection status changed, connection: 293266ae,
 status: CREATED
2019-04-08 14:11:26.455  INFO 20910 --- [ntLoopGroup-2-4] c.a.o.iot.api.http2.IotHttp2Client       : connection status changed, connection: 10936960,
 status: CREATED
2019-04-08 14:11:26.474  INFO 20910 --- [ntLoopGroup-2-3] c.a.o.iot.api.http2.IotHttp2Client       : connection status changed, connection: 293266ae,
 status: AUTHORIZED
2019-04-08 14:11:26.491  INFO 20910 --- [ntLoopGroup-2-4] c.a.o.iot.api.http2.IotHttp2Client       : connection status changed, connection: 10936960,
 status: AUTHORIZED
```

和阿里云IoT平台连接已被授权建立

在公网服务器上运行

- 收到设备信息（上线、属性上报），更新数据库

```
2019-04-08 14:15:02.976  INFO 20910 --- [ntLoopGroup-2-2] c.a.o.i.a.m.impl.MessageClientImpl       : receive msg, messageId:1115135947848053248, data
 size: 256
2019-04-08 14:15:02.978  INFO 20910 --- [ient-receiver-0] egister$$EnhancerBySpringCGLIB$$c766e467 : receive 设备状态变化上报{"lastTime":"2019-04-08
14:15:02.836","utcLastTime":"2019-04-08T06:15:02.836Z","clientIp":"112.65.61.64","utcTime":"2019-04-08T06:15:02.848Z","time":"2019-04-08 14:15:02.848
","productKey":"a1zPqJ3NYxJ","deviceName":"2019CT_case2_node001","status":"online"}
2019-04-08 14:15:03.048  INFO 20910 --- [ient-receiver-0] com.zaxxer.hikari.HikariDataSource       : AppHikariCP - Starting...
2019-04-08 14:15:03.212  INFO 20910 --- [ient-receiver-0] com.zaxxer.hikari.HikariDataSource       : AppHikariCP - Start completed.
2019-04-08 14:15:03.218 DEBUG 20910 --- [ient-receiver-0] c.e.d.a.d.m.D.selectOneByExample         : ==>  Preparing: SELECT id,device_name,current_te
mperature,current_humidity,temp_threshold,alarm_status,connect_status,gmt_update,last_alarm_date,last_online_date FROM device_info WHERE ( ( device_n
ame = ? ) )
2019-04-08 14:15:03.242 DEBUG 20910 --- [ient-receiver-0] c.e.d.a.d.m.D.selectOneByExample         : ==> Parameters: 2019CT_case2_node001(String)
2019-04-08 14:15:03.258 DEBUG 20910 --- [ient-receiver-0] c.e.d.a.d.m.D.selectOneByExample         : <==      Total: 1
2019-04-08 14:15:03.262 DEBUG 20910 --- [ient-receiver-0] .e.d.a.d.m.D.updateByPrimaryKeySelective : ==>  Preparing: UPDATE device_info SET device_na
me = ?,connect_status = ?,gmt_update = ?,last_online_date = ? WHERE id = ?
2019-04-08 14:15:03.264 DEBUG 20910 --- [ient-receiver-0] .e.d.a.d.m.D.updateByPrimaryKeySelective : ==> Parameters: 2019CT_case2_node001(String), 在
线(String), 2019-04-08 14:15:03.26(Timestamp), 2019-04-08 14:15:03.26(Timestamp), 16(Integer)
2019-04-08 14:15:03.268 DEBUG 20910 --- [ient-receiver-0] .e.d.a.d.m.D.updateByPrimaryKeySelective : <==    Updates: 1
2019-04-08 14:15:03.562  INFO 20910 --- [ntLoopGroup-2-2] c.a.o.i.a.m.impl.MessageClientImpl       : receive msg, messageId:1115135950716955648, data
 size: 327
2019-04-08 14:15:03.563  INFO 20910 --- [ient-receiver-1] egister$$EnhancerBySpringCGLIB$$c766e467 : receive 属性上报{"deviceType":"CustomCategory","
iotId":"UeUyHwfqHbll0wTX4kpF000100","productKey":"a1zPqJ3NYxJ","gmtCreate":1554704103527,"deviceName":"2019CT_case2_node001","items":{"CurrentHumidit
y":{"value":52,"time":1554704103532},"CurrentTemperature":{"value":26.86,"time":1554704103532},"TempThreshold":{"value":38,"time":1554704103532}}}
2019-04-08 14:15:03.574 DEBUG 20910 --- [ient-receiver-1] c.e.d.a.d.m.D.insertOrUpdate             : ==>  Preparing: INSERT INTO device_info(device_n
ame,current_temperature,current_humidity,temp_threshold) VALUES(?,?,?,?) ON DUPLICATE KEY UPDATE device_name=?, current_temperature=?, current_humidi
ty=?, temp_threshold=?
2019-04-08 14:15:03.575 DEBUG 20910 --- [ient-receiver-1] c.e.d.a.d.m.D.insertOrUpdate             : ==> Parameters: 2019CT_case2_node001(String), 26
.86(Float), 52.0(Float), 38.0(Float), 2019CT_case2_node001(String), 26.86(Float), 52.0(Float), 38.0(Float)
2019-04-08 14:15:03.580 DEBUG 20910 --- [ient-receiver-1] c.e.d.a.d.m.D.insertOrUpdate             : <==    Updates: 2
2019-04-08 14:15:03.582 DEBUG 20910 --- [ient-receiver-1] c.e.d.a.d.m.D.insert                     : ==>  Preparing: INSERT INTO device_prop_history_
log ( id,device_name,current_temperature,current_humidity,gmt_create ) VALUES( ?,?,?,?,? )
2019-04-08 14:15:03.583 DEBUG 20910 --- [ient-receiver-1] c.e.d.a.d.m.D.insert                     : ==> Parameters: null, 2019CT_case2_node001(Strin
g), 26.86(Float), 52.0(Float), 2019-04-08 14:15:03.527(Timestamp)
2019-04-08 14:15:03.588 DEBUG 20910 --- [ient-receiver-1] c.e.d.a.d.m.D.insert                     : <==    Updates: 1
```

在公网服务器上运行

- 查看数据库里表的内容（来自阿里IoT平台的HTTP/2推送）

```
mysql> select id,device_name, current_temperature, current_humidity from device_info;
+----+---------------------+---------------------+------------------+
| id | device_name         | current_temperature | current_humidity |
+----+---------------------+---------------------+------------------+
| 16 | 2019CT_case2_node001 |               28.06 |               49 |
+----+---------------------+---------------------+------------------+
1 row in set (0.00 sec)

mysql> select id,device_name, current_temperature, current_humidity from device_prop_history_log;
+-----+---------------------+---------------------+------------------+
| id  | device_name         | current_temperature | current_humidity |
+-----+---------------------+---------------------+------------------+
| 402 | 2019CT_case2_node001 |               25.45 |               54 |
| 403 | 2019CT_case2_node001 |               25.49 |               54 |
| 404 | 2019CT_case2_node001 |               25.53 |               54 |
| 405 | 2019CT_case2_node001 |               25.53 |               54 |
| 406 | 2019CT_case2_node001 |               25.78 |               54 |
| 407 | 2019CT_case2_node001 |                25.8 |               53 |
| 408 | 2019CT_case2_node001 |               25.84 |               53 |
| 409 | 2019CT_case2_node001 |               25.84 |               53 |
| 410 | 2019CT_case2_node001 |               25.88 |               53 |
| 411 | 2019CT_case2_node001 |               25.86 |               53 |
| 412 | 2019CT_case2_node001 |                25.9 |               53 |
| 413 | 2019CT_case2_node001 |                25.9 |               53 |
```

在公网服务器上运行

- 不挂断开启程序
  - 使用nohup java -jar device-admin-web-0.0.1-SNAPSHOT.jar & 命令后台不挂断运行执行程序

```
[root@iz2zeczwmztkrnzjqhn0dqz ~]# nohup java -jar device-admin-5.0-SNAPSHOT.jar &
[1] 25555
[root@iz2zeczwmztkrnzjqhn0dqz ~]# nohup: ignoring input and appending output to 'nohup.out
```

1

- 关闭程序
  - netstat –pan |grep portno命令查看端口占用情况
  - kill pid命令 终止进程


```
Xshell 6 (Build 0121)
Copyright (c) 2002 NetSarang Computer, Inc. All rights reserved.

Type `help' to learn how to use Xshell prompt.
[C:\~]$

Connecting to 47.94.214.229:22...
Connection established.
To escape to local shell, press 'Ctrl+Alt+]'.

WARNING! The remote SSH server rejected X11 forwarding request.
Last login: Wed Apr 10 14:53:41 2019 from 112.65.48.88

Welcome to Alibaba Cloud Elastic Compute Service !

[root@iz2zeczwmztkrnzjqhn0dqz ~]# netstat -pan | grep 9099
tcp        0      0 0.0.0.0:9099            0.0.0.0:*               LISTEN      25555/java
tcp        0      0 172.16.252.218:9099     112.65.48.88:18783      ESTABLISHED 25555/java
[root@iz2zeczwmztkrnzjqhn0dqz ~]# kill 25555
```

3

life.augmented

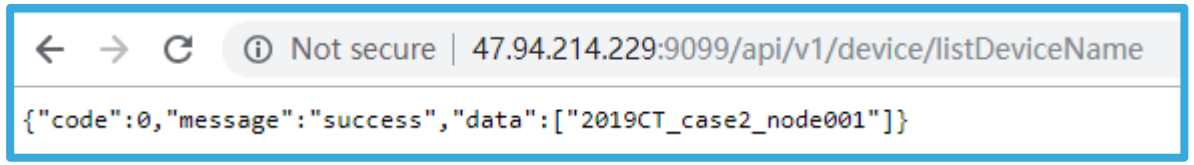
阿里云

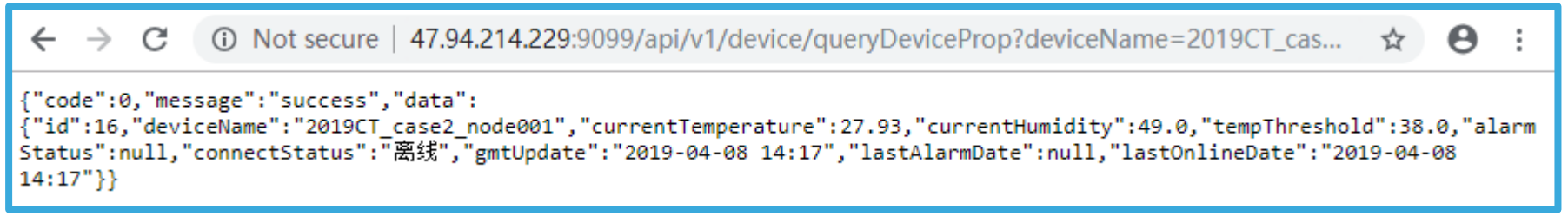
在公网服务器上运行

- 公网服务器开通服务器9099端口

在公网服务器上运行

- 浏览器访问





下节预告

- 第一节：综合软件架构介绍
  - 软件架构，知识结构梳理
- 第二节：后端服务开发
  - 后端框架，后端项目：和阿里云IoT平台对接，操作数据库，和前端应用对接
- 第三节：前端服务开发体验
  - 系统应用前端开发详解