**第二章 继承MP**

**2.1 创建测试表**

|  |
| --- |
| SET FOREIGN\_KEY\_CHECKS=0;  -- ----------------------------  -- Table structure for tb\_employee  -- ----------------------------  DROP TABLE IF EXISTS `tb\_employee`;  CREATE TABLE `tb\_employee` (  `id` int(11) NOT NULL AUTO\_INCREMENT,  `last\_name` varchar(255) DEFAULT NULL,  `email` varchar(255) DEFAULT NULL,  `gender` varchar(255) DEFAULT NULL,  `age` int(11) DEFAULT NULL,  PRIMARY KEY (`id`)  ) ENGINE=InnoDB AUTO\_INCREMENT=5 DEFAULT CHARSET=utf8;  -- ----------------------------  -- Records of tb\_employee  -- ----------------------------  INSERT INTO `tb\_employee` VALUES ('1', 'tom', 'tom@atguigu.com', '1', '22');  INSERT INTO `tb\_employee` VALUES ('2', 'jerry', 'jerry@atguigu.com', '0', '25');  INSERT INTO `tb\_employee` VALUES ('3', 'black', 'black@atguigu.com', '1', '30');  INSERT INTO `tb\_employee` VALUES ('4', 'white', 'white@atguigu.com', '0', '35'); |

**2.2 创建javabean**

创建普通maven项目。。。该步骤省略

创建javabean Employee

|  |
| --- |
| package cn.huangwei.entity;  */\*\*  \* java bean  \*  \* `id` int(11) NOT NULL AUTO\_INCREMENT,  \* `last\_name` varchar(255) DEFAULT NULL,  \* `email` varchar(255) DEFAULT NULL,  \* `gender` varchar(255) DEFAULT NULL,  \* `age` int(11) DEFAULT NULL,  \*  \* 注意：定义javabean时，成员变量所使用的类型使用包装类型  \* 因为每个基本类型都有一个默认值；所以对于一个初始对象来说，明明没有值的也会出现默认值  \* 如果设置为包装类，那么只有一个null值。  \*/* public class Employee {  private Integer id;  private String lastName;  private String email;  private Integer gender;  private Integer age;   public Integer getId() {  return id;  }   public void setId(Integer id) {  this.id = id;  }   public String getLastName() {  return lastName;  }   public void setLastName(String lastName) {  this.lastName = lastName;  }   public String getEmail() {  return email;  }   public void setEmail(String email) {  this.email = email;  }   public Integer getGender() {  return gender;  }   public void setGender(Integer gender) {  this.gender = gender;  }   public Integer getAge() {  return age;  }   public void setAge(Integer age) {  this.age = age;  }   @Override  public String toString() {  return "Employee{" +  "id=" + id +  ", lastName='" + lastName + '\'' +  ", email='" + email + '\'' +  ", gender=" + gender +  ", age=" + age +  '}';  } } |

**2.3 依赖配置**

（1）在pom.xml中加入对MP、spring】连接池、mysql驱动等依赖

注意：mybatis和mybatis-spring依赖请勿加入项目配置，以免引起版本冲突，mybatis-plus会帮你自动维护。

|  |
| --- |
| <?xml version="1.0" encoding="UTF-8"?> <project xmlns="http://maven.apache.org/POM/4.0.0"  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">  <modelVersion>4.0.0</modelVersion>   <groupId>cn.huangwei</groupId>  <artifactId>mp01</artifactId>  <version>1.0-SNAPSHOT</version>    <dependencies>   <!--mp依赖-->  <dependency>  <groupId>com.baomidou</groupId>  <artifactId>mybatis-plus</artifactId>  <version>3.1.0</version>  </dependency>   <!--junit-->  <dependency>  <groupId>junit</groupId>  <artifactId>junit</artifactId>  <version>4.12</version>  </dependency>   <!--log4j-->  <dependency>  <groupId>org.apache.logging.log4j</groupId>  <artifactId>log4j</artifactId>  <version>2.10.0</version>  </dependency>   <!--c3p0连接池-->  <dependency>  <groupId>com.mchange</groupId>  <artifactId>c3p0</artifactId>  <version>0.9.5.2</version>  </dependency>   <!--mysql-->  <dependency>  <groupId>mysql</groupId>  <artifactId>mysql-connector-java</artifactId>  <version>5.1.47</version>  </dependency>   <!--spring-->  <dependency>  <groupId>org.springframework</groupId>  <artifactId>spring-context</artifactId>  <version>5.0.11.RELEASE</version>  </dependency>   <dependency>  <groupId>org.springframework</groupId>  <artifactId>spring-orm</artifactId>  <version>5.0.11.RELEASE</version>  </dependency>   </dependencies> </project> |

（2）加入Mybatis的全局配置文件mybatis-config.xml

|  |
| --- |
| <?xml version="1.0" encoding="utf-8" ?> <!DOCTYPE configuration PUBLIC "-//mybatis.org//DTD Config 3.0//EN"  "http://mybatis.org/dtd/mybatis-3-config.dtd"> <configuration>       </configuration> |

（3）加入log4j的配置文件log4j.xml

|  |
| --- |
| <?xml version="1.0" encoding="UTF-8" ?> <!DOCTYPE log4j:configuration SYSTEM "log4j.dtd">  <log4j:configuration debug="true" xmlns:log4j='http://jakarta.apache.org/log4j/' >  <appender name="STDOUT" class="org.apache.log4j.ConsoleAppender">  <param name="Encoding" value="UTF-8" />  <layout class="org.apache.log4j.PatternLayout">  <param name="ConversionPattern" value="%-5p %d{MM-dd HH:mm:ss, SSS} %m (%F:%L) \n" />  </layout>  </appender>  <logger name="java.sql">  <level value="debug"/>  </logger>   <logger name="org.apache.ibatis">  <level value="info"/>  </logger>   <root>  <level value="debug" />  <appender-ref ref="STDOUT"/>  </root>  </log4j:configuration> |

（4）数据库的链接信息db.properties

|  |
| --- |
| jdbc.driver=com.mysql.jdbc.Driver jdbc.url=jdbc:mysql://localhost:3306/mp jdbc.username=root jdbc.password=123 |

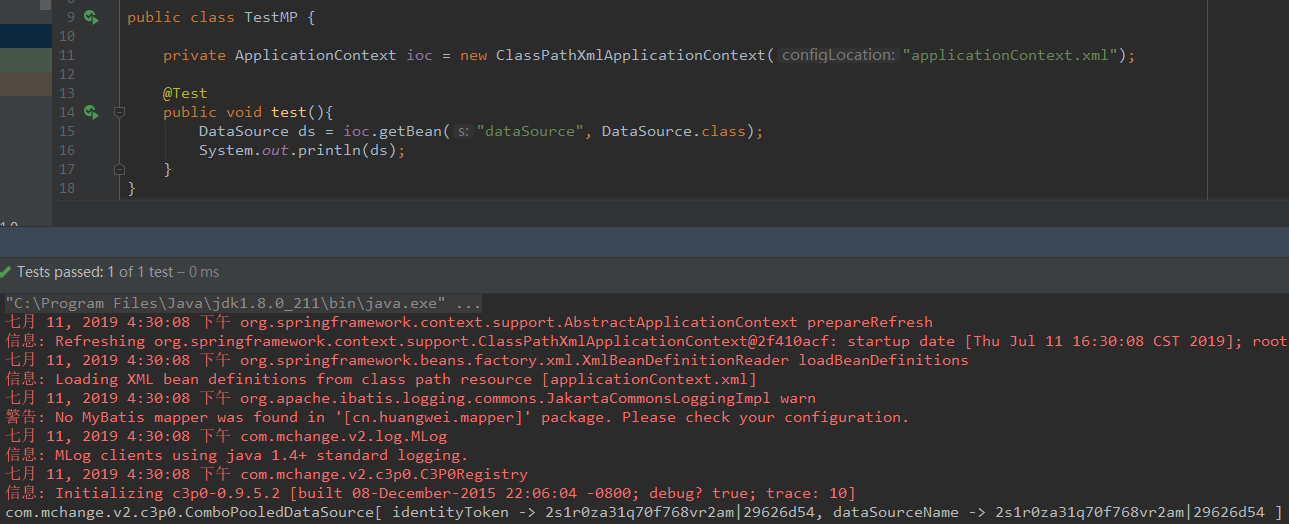
（5）spring的配置文件 applicationContext.xml

|  |
| --- |
| <?xml version="1.0" encoding="UTF-8"?> <beans xmlns="http://www.springframework.org/schema/beans"  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  xmlns:context="http://www.springframework.org/schema/context"  xmlns:tx="http://www.springframework.org/schema/tx"  xsi:schemaLocation="http://www.springframework.org/schema/beans  http://www.springframework.org/schema/beans/spring-beans.xsd  http://www.springframework.org/schema/context  http://www.springframework.org/schema/context/spring-context.xsd  http://www.springframework.org/schema/tx  http://www.springframework.org/schema/tx/spring-tx.xsd ">   <!--数据源-->  <context:property-placeholder location="classpath:db.properties" />  <bean id="dataSource" class="com.mchange.v2.c3p0.ComboPooledDataSource">  <property name="driverClass" value="${jdbc.driver}"></property>  <property name="jdbcUrl" value="${jdbc.url}"></property>  <property name="user" value="${jdbc.username}"></property>  <property name="password" value="${jdbc.password}"></property>  </bean>   <!--事务管理器-->  <bean id="dataSourceTransactionManager" class="org.springframework.jdbc.datasource.DataSourceTransactionManager">  <property name="dataSource" ref="dataSource"></property>  </bean>   <!--基于注解的事务管理-->  <tx:annotation-driven transaction-manager="dataSourceTransactionManager" />   <!--配置sqlSessionFactoryBean-->  <bean id="sqlSessionFactoryBean" class="org.mybatis.spring.SqlSessionFactoryBean">  <!--数据源-->  <property name="dataSource" ref="dataSource"></property>  <property name="configLocation" value="classpath:mybatis-config.xml"></property>   <!--别名处理-->  <property name="typeAliasesPackage" value="cn.huangwei.entity"></property>   </bean>   <!--配置mybatis扫描mapper接口路径-->  <bean class="org.mybatis.spring.mapper.MapperScannerConfigurer">  <property name="basePackage" value="cn.huangwei.mapper"></property>  </bean> </beans> |

**2.4 测试**

测试Spring-Mybatis的环境，保证可行

|  |
| --- |
| package cn.huangwei.test;  import org.junit.Test; import org.springframework.context.ApplicationContext; import org.springframework.context.support.ClassPathXmlApplicationContext;  import javax.sql.DataSource;  public class TestMP {   private ApplicationContext ioc = new ClassPathXmlApplicationContext("applicationContext.xml");   @Test  public void test(){  DataSource ds = ioc.getBean("dataSource", DataSource.class);  System.*out*.println(ds);  } } |



2.6 集成MP

（1）mybatis-plus的继承非常简单，对于Spring，我们仅仅需要把Mybatis自带的SqlSessionFactoryBean替换成MP自带的即可

<bean id="sqlSessionFactoryBean" class="com.baomidou.mybatisplus.extension.spring.MybatisSqlSessionFactoryBean">

**第三章 入门HelloWorld**

**3.1 通用CRUD**

（1）提出问题：

假设我们已经存在一张tb\_employee表，且已经有了实体类Employee，实现tb\_employee表的CRUD操作我们需要做什么呢？

（2）实现方式：

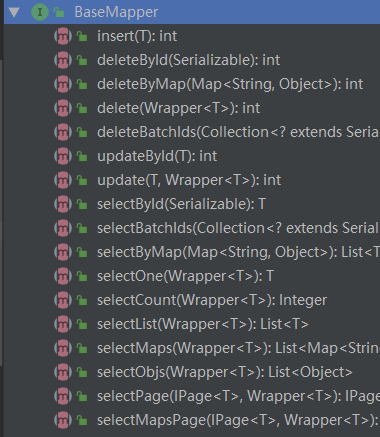
基于mybatis：需要编写EmployeeMapper接口，并手动编写CRUD方法，提供EmployeeMapper.xml映射文件，并手动编写每个方法对应的SQL语句

基于MP：

只需要创建EmployeeMapper接口并继承BaseMapper接口，这就是使用MP需要完成的所有操作，甚至不需要创建sql映射文件

|  |
| --- |
| package cn.huangwei.mapper;  import cn.huangwei.entity.Employee; import com.baomidou.mybatisplus.core.mapper.BaseMapper;  */\*\*  \* mapper接口  \* 基于mybatis：需要在mapper中编写CRUD相关方法，以及Mapper对应文件及sql语句  \*  \* 基于MP：让xxMapper接口继承BaseMapper接口  \* BaseMapper<T>:泛型指定的就是当前Mapper接口所操作的实体类类型  \*/* public interface EmployeeMapper extends BaseMapper<Employee> {  } |

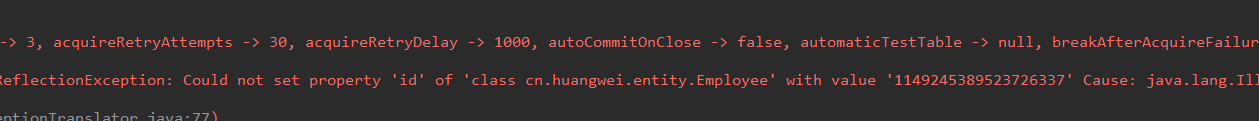
因为baseMapper定义了基本的Crud操作



**3.2插入操作**

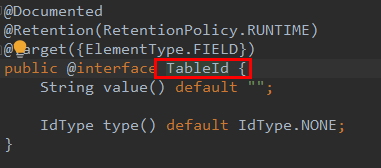
|  |
| --- |
| @Test public void testCommonInsert(){  Employee employee = new Employee();  employee.setLastName("MP");  employee.setEmail("mp@huangwei.com");  employee.setAge(22);  int result = employeeMapper.insert(employee);  System.*out*.println("result:" + result); } |

运行发现出现下面异常



MP支持4种主键策略：自增，用户输入，全局唯一自动填充（默认配置），全局唯一自动填充。

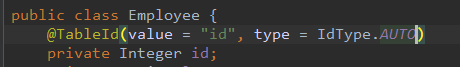
如果代码中没有设定主键策略，那么就会随机生成某个主键，故需要指定逐渐策略在id主键上加上@TableId注解



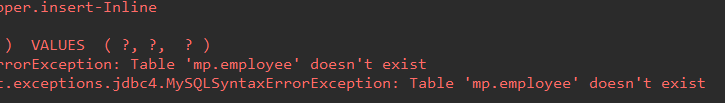
type就是主键策略，value表示如果表中主键列与成员变量id一致，可省略，否则需要填写表中主键列的列名



所以进行以下修改



修改之后重新运行出现新的异常：



出现的原因是Mybatis-Plus 会默认使用实体类名称到数据库中找对应的表；数据库中没有employee这张表，只有tb\_employee表，所以出现这个异常

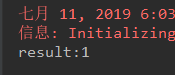
此时需要介绍另一个注解@TableName



主要用value指定表名，如果是自定义的复杂结构，那么就需要用resultMap进行xml中id的指定。

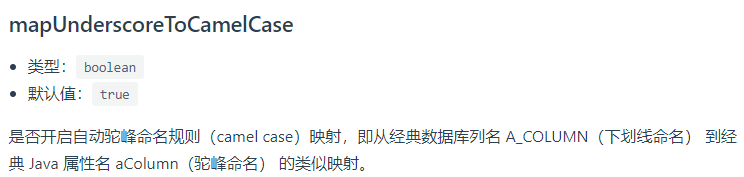
添加@tableName修改

然后运行，结果如下图，成功运行



这里mybatis-plus将employee的lastName属性映射到了表中的last\_name属性列，这个是怎么配置的呢？

**Configuration下的**



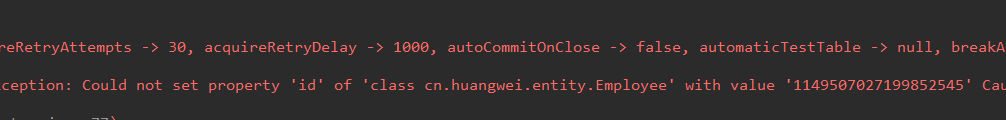
此版本3.1统一使用该属性进行驼峰命名的映射，已经将db-column-underline去除

**3.2.1MP全局配置**

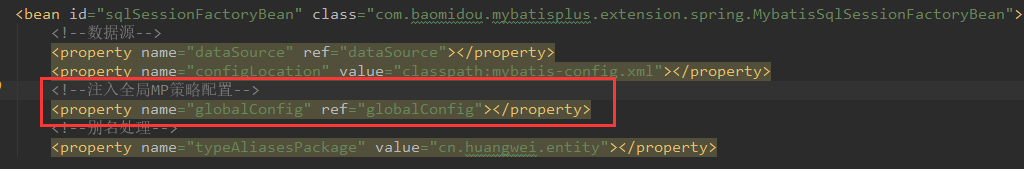
**（1）MP全局主键配置**

|  |
| --- |
| <!--定义MybatisPlus全局配置文件--> <bean id="globalConfig" class="com.baomidou.mybatisplus.core.config.GlobalConfig">  <!--全局主键策略-->  <property name="dbConfig">  <bean class="com.baomidou.mybatisplus.core.config.GlobalConfig.DbConfig">  <!--主键规则-->  <property name="idType" value="AUTO"/>  </bean>  </property> </bean> |

此时注释掉Employee中的idType设置，重新运行

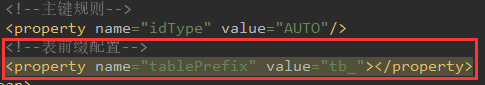


又出现该异常，原因是配置的全局配置还没有注入到SqlSessionfactroy中，还需进行如下操作



此时插入成功。

（2）全局表前缀配置



测试成功通过。

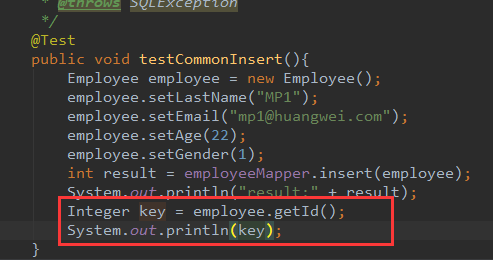
（3）@TableField注解

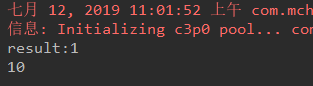


如果没有配置驼峰命名，那么需要该注解的value属性执行实体成员变量lastName->last\_name; 如果你在实体中添加了一个salary成员变量，但是数据库中没有该属性列，那么由于该注解exist属性默认为true，即数据库中存在salary字段，但是实际中没有该字段，因此会出错，可以将exist设置为false；

**3.2.2 获取插入数据的id值**

mybatis需要在insert标签中设定useGenerateKeys=”true” keyProperty=”id”，才可以返回对应插入数据的主键，而**MP中自动将主键值写到实体类中**。





上述插入过程中，如果我们将age和gender两个值不作设置，那么在insert的SQL语句中，就不会出现age和gender相关属性列，如果想要它仍然有属性列，那么需要使用其他方法。

baseMapper中有个insertAllColumn方法，该方法不管有没有设置属性，都会将所有要插入的属性显示到SQL中，这个针对MP2.3，不适用与MP3.1。

**3.3 查询操作**

|  |
| --- |
| T selectById(Serializable id); 通过id进行实体类查询  List<T> selectBatchIds(@Param("coll") Collection<? extends Serializable> idList); 通过多个id进行查询，传入集合框架，主要是list  List<T> selectByMap(@Param("cm") Map<String, Object> columnMap);  通过map封装条件查询，使用的是columnMap，key中使用数据表列名，不是对象属性名  下面这些方法由于涉及到Wrapper，留到后面条件构造器进行解释  T selectOne(@Param("ew") Wrapper<T> queryWrapper);  Integer selectCount(@Param("ew") Wrapper<T> queryWrapper);  List<T> selectList(@Param("ew") Wrapper<T> queryWrapper);  List<Map<String, Object>> selectMaps(@Param("ew") Wrapper<T> queryWrapper);  List<Object> selectObjs(@Param("ew") Wrapper<T> queryWrapper);  IPage<T> selectPage(IPage<T> page, @Param("ew") Wrapper<T> queryWrapper);  IPage<Map<String, Object>> selectMapsPage(IPage<T> page, @Param("ew") Wrapper<T> queryWrapper); |

**3.4 MP启动注入SQL原理分析**

（1）问题：xxMapper继承了BaseMapper<T>, BaseMapper中提供了通用的CRUD方法，方法来源与BaseMapper，有方法就必须有SQL，因为Mybatis最终还是需要通过SQL语句操作数据。前置知识：MyBatis框架注入SQL原理的执行流程

（2）透过现象看本质。

**现象：**

1.employee Mapper是一个代理对象，MapperProxy。

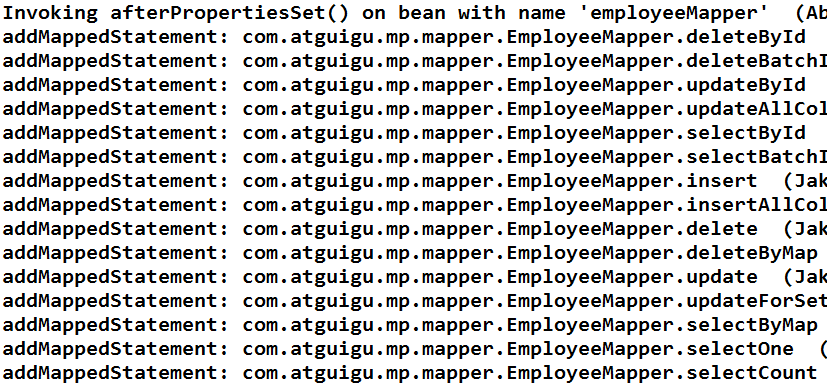
2.MapperProxy中有个sqlSession, sqlSession中包含了sqlSessionFactory。

3. sqlSessionFactory中有一个Configuration对象

4.Confiuration中包含了所有定义好的MappedStatements，每个MappedStatement都表示Mapper接口的一个方法和Mapper映射文件，其中有个SqlSource，里面就存着我们的sql语句

5.MP在启动时候就会分析xxxMapper中的方法，并且将对应的sql语句处理好，保存到configuration对象中的mappedStatements中。

**本质：**



以后有时间再分析

**第四章：条件构造器AbstractWrapper**

**4.1 AbstractWrapper简介**

（1）实体包装器，主要用于处理sql拼接，排序，实体参数查询等

（2）注意：方法入参中的R column均表示数据库字段,当R具体类型为String时则为**数据库字段名**(字段名是数据库关键字的自己用转义符包裹!)!**而不是实体类数据字段名**!!!,

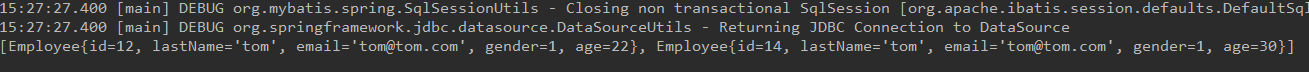


**4.2 使用条件构造器进行查询操作**

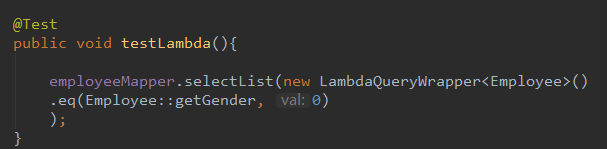
分页查询，年龄为18-50之间性别为男的名字tom的用户

|  |
| --- |
| */\*\*  \* 使用条件构造器进行查询操作  \*/* @Test public void testEW() {  List<Employee> employees = employeeMapper.selectList(new QueryWrapper<Employee>()  .between("age", 18, 50)  .eq("gender", 1)  .eq("last\_name", "tom")  );  System.*out*.println(employees); } |

显示结果

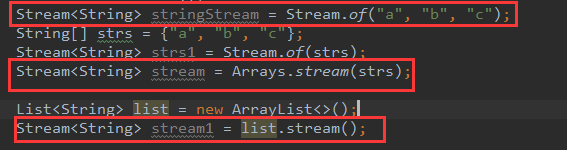


有关Boolean condition参数，如果是true表明该条件作为条件查询，否则不作为条件查询

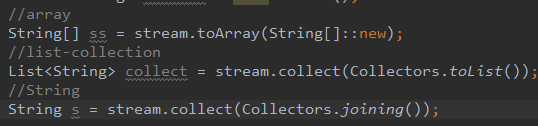


**第五章Stream与lambda表达式操作**

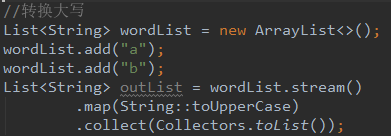
1.常用Stream用法



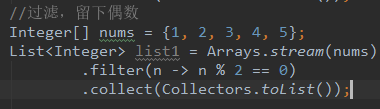
2.Stream转换为其他数据结构

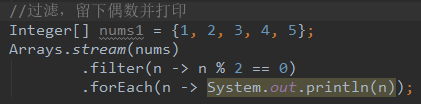


3.Stream的转换：map、flatMap（一对多情况）



4.Stream的过滤filter





5.分组，groupingBy与partitioningBy