测试对象：EmployeeService类

测试人员：涂成武

测试日期：2020-05-11

1. 测试对象描述

该类为service层一个业务类，负责调用mapper层mybatis的接口进行数据的crud操作。该类对应的数据库表为employee表于empsalary表。

该类含有如下方法：

1. getEmployeeByPage()：该方法的作用为分页查询员工信息，该方法接受的参数有四个，分别为页数，页大小，查询条件(封装成Employee类型变量)，起始入职事件范围(以Date数组的形式传入)，返回值为如下所示的一个类型的变量：

public class RespPageBean {

private Long total;

private List<?> data;

}

其中total变量表示的是符合搜索条件的记录数，data变量封装了当前页的搜索结果。

1. addEmp()：该方法的作用为向employee表中插入一条数据，该方法在传入参数的某些字段未指定时，会根据默认规则自动计算，返回值为插入的记录数。
2. maxWorkID()：该方法返回employee表中所有记录的WorkID字段的最大值。
3. deleteEmpByEid()：该方法根据id删除employee表中某条特定记录，传入参数为id(封装为Integer变量)，返回值为删除的记录数。
4. updateEmp()：该方法根据id更新employee表中某条特定记录，传入参数为待更新的记录(封装为Employee类型的变量)，根据该变量的id进行更新，并且只更新变量中非空字段，返回值为更新的记录数。
5. addEmps()：该方法向employee表中添加一组记录，传入参数为一组记录(封装成一个List<Employee>)，然后将List中每个元素插入employee表中，返回值为插入的记录数。
6. getEmployeeByPageWithSalary()：该方法的作用为分页查询员工及其薪资信息，不设查询条件，该方法接受的参数有两个，分别为页数，页大小，返回值为RespPageBean类型(参见1)。
7. updateEmployeeSalaryById()：该方法的作用为更新empsalary中的记录，当原本的记录不存在时新建记录，当原本记录存在时删除原纪录新建另一条，返回值为涉及的记录条数，即不存在时返回1，存在时返回2。
8. getEmployeeById()：该方法的作用为根据id获取特定员工信息，传入参数为id(封装为Integer类)，返回为查询出的Employee类型变量。
9. 测试用例设计
10. 对于getEmployeeByPage方法，该方法有4个参数，根据数据库中的记录条数，可以为页大小和页数使用等价类方法，故我们设计了如下10个测试用例，由于逐个检验查询得到的数据开销过大，我们使用返回变量中的数据属性的size作为判断测试是否通过的依据。
11. 将所有记录放入一页，取出第一页，并且不设置搜索条件

参数为：1，617，new Employee(),null

预期结果为：617

1. 将所有记录放入一页，取出第二页，并且不设置搜索条件

参数为：2，617，new Employee()，null

预期结果为：0

1. 页大小为-1，且不设置搜索条件

参数为：2，-1，new Employee()，null

预期结果为：抛出BadSqlGrammarException类型异常

1. 页大小为10，取第一页，指定姓名为近似搜索条件，保证搜索结果不为空

参数为：1，10，condition = *new* Employee();condition.setName("江南一点雨");，null

预期结果为：10

1. 页大小为20，取最后一页，指定合同类型为劳动合同

参数为：25，20，condition=*new* Employee();condition.setEngageForm("劳动合同");，null

预期结果为：2

1. 页大小为15，取最后一页，指定politicId为3

参数为：12，15，condition=*new* Employee();condition.setPoliticId(3);，null

预期结果为：3

1. 页大小为10，取最后一页，指定jobLevelId为12

参数为：18，10，condition=*new* Employee();condition.setJobLevelId(12);，null

预期结果为：4

1. 页大小为8，取最后一页，指定departmentId为82

参数为：7，8 condition=*new* Employee();condition.setDepartmentId(82);，null

预期结果为：6

1. 页大小为12，取最后一页，指定posId为29

参数为：13，12，condition=*new* Employee();condition.setPosId(29);，null

1. 页大小为15，取最后一页，按合同起始日期来进行筛选

参数为：31，15，new Employee()，Date[] beginDateScope = *new* Date[2];  
DateConverter dateConverter=*new* DateConverter();  
beginDateScope[0]=dateConverter.convert("2017-12-15");  
beginDateScope[1]=dateConverter.convert("2018-01-05");

预期结果为：3

1. 对于addEmp()方法，该方法接受Employee类型的参数，根据该实体类与插入操作的执行过程，我们设计了如下3个测试用例：
2. 插入的记录的属性均为空

参数为：new Employee()

预期结果为：1(并且数据库中插入了一条除主键外均为空的记录)

1. 插入的记录的主键已经存在于数据库中

参数为：toAdd.setId(1941);

预期结果为：抛出DuplicateKeyException类型异常

1. 普通有效插入记录：

参数为：toAdd.setId(2000);  
toAdd.setName("涂成武");  
toAdd.setGender("男");  
toAdd.setBirthday(*new* Date());  
toAdd.setIdCard("4986151658149");  
toAdd.setWedlock("未婚");  
toAdd.setNationId(1);  
toAdd.setNativePlace("上海");  
toAdd.setPoliticId(3);  
toAdd.setEmail("tuchengwu19991218@qq.com");  
toAdd.setPhone("17621777283");  
toAdd.setAddress("龙水北路960弄12号404");  
toAdd.setDepartmentId(78);  
toAdd.setJobLevelId(15);  
toAdd.setPosId(33);  
toAdd.setEngageForm("劳务合同");  
toAdd.setTiptopDegree("本科");  
toAdd.setSpecialty("软件工程");  
toAdd.setSchool("同济大学");  
toAdd.setBeginDate(*new* Date());  
toAdd.setWorkState("在职");  
toAdd.setWorkID("00000065");  
toAdd.setConversionTime(*new* Date());  
toAdd.setNotWorkDate(*null*);  
toAdd.setBeginContract(*new* Date(110,0,24));  
toAdd.setEndContract(*new* Date(111,0,24));  
toAdd.setWorkAge(1);

预期结果为：1(数据库中插入了一条相应的记录)

1. 对于maxWorkID()方法，该方法没有参数，只需要设计如下一个测试用例即可：
2. 无参数，直接调用

参数：无

预期结果为：65

1. 对于deleteEmpById()方法，该方法接受一个整型作为主键，那么利用等价类方法，设计如下2个测试用例
2. 主键在数据表中

参数：2000

预期结果为：1(数据表中主键为2000的记录被删除)

前提条件：为了不对原始的记录产生影响，需要在执行测试用例之前，新建一个主键为2000的记录

1. 主键不在数据表中

参数：2001

预期结果为：0

1. 对于updateEmp()方法，该方法接受一个Employee类型变量，根据该update语句的执行过程，设计如下4个测试用例：
2. 更新主键在记录中，并且记录字段不均为空

参数：toUpdate=employeeService.getEmployeeById(1941);  
toUpdate.setWedlock("未婚");

预期结果为：1(数据表中主键为1941的记录被修改)

1. 更新主键不在记录中，并且记录字段均不为空

参数：toUpdate.setId(2500);(其中wedlock在a中已经设置)

预期结果为：0

1. 更新主键在记录中，但其余字段均为空

参数：toUpdate=*new* Employee();toUpdate.setId(1569);

预期结果为：抛出BadSqlGrammarException类型异常

1. 更新主键不在记录中，且其余字段均为空

参数：toUpdate=*new* Employee();toUpdate.setId(2569);

预期结果为：抛出BadSqlGrammarException类型异常

1. 对于addEmps()方法，接收参数为List，根据List的特点，设计如下2个测试用例：
2. 列表为空

参数：toAddList = *new* ArrayList<Employee>()

预期结果为：抛出BadSqlGrammarException类型异常

1. 列表不为空

参数：toAdd = *new* Employee();  
toAdd.setId(2100);  
toAdd.setName("涂成武");  
toAdd.setGender("男");  
toAdd.setBirthday(*new* Date());  
toAdd.setIdCard("4986151658149");  
toAdd.setWedlock("未婚");  
toAdd.setNationId(1);  
toAdd.setNativePlace("上海");  
toAdd.setPoliticId(3);  
toAdd.setEmail("tuchengwu19991218@qq.com");  
toAdd.setPhone("17621777283");  
toAdd.setAddress("龙水北路960弄12号404");  
toAdd.setDepartmentId(78);  
toAdd.setJobLevelId(15);  
toAdd.setPosId(33);  
toAdd.setEngageForm("劳务合同");  
toAdd.setTiptopDegree("本科");  
toAdd.setSpecialty("软件工程");  
toAdd.setSchool("同济大学");  
toAdd.setBeginDate(*new* Date());  
toAdd.setWorkState("在职");  
toAdd.setWorkID("00000065");  
toAdd.setConversionTime(*new* Date());  
toAdd.setNotWorkDate(*null*);  
toAdd.setBeginContract(*new* Date(110,0,24));  
toAdd.setEndContract(*new* Date(111,0,24));  
toAdd.setWorkAge(1);  
toAddList.add(toAdd);

预期结果为：1(数据表中插入了1条记录，记录详情与参数一致)

1. 对于getEmployeeByPageWithSalary()方法，设计策略与getEmployeeByPage()方法类似，设计得到如下3个测试用例：
2. 页大小为总记录数量，取出第一页

参数：1，617

预期结果：617

1. 页大小为总记录数量，取出第二页

参数：2，617

预期结果：0

1. 页大小为负数

参数：2，-1

预期结果为：抛出BadSqlGrammarException类型异常

1. 对于updateEmployeeSalaryById()方法，考虑到该表涉及到两个外键约束，设计得到如下4个测试用例：
2. eid存在于empsalary中，并且sid存在于salary中

参数：5，14

预期结果为：2(删除原本eid为5的记录，创建一条新的记录)

1. eid不存在empsalary中，但是sid不存在salary中

参数：5，1

预期结果为：抛出DataIntegrityViolationException类型异常

1. eid不存在empsalary中，sid存在于salary中

参数：48，14

预期结果为：1(新建一条记录)

1. eid不存在emplpoyee中

参数为：2000，13

预期结果为：抛出DataIntegrityViolationException类型异常

1. 对于getEmployeeById()方法，根据select语句的执行结果以及主键的唯一性，设计如下2个测试用例：
2. 主键在表中

参数：18

预期结果：与Employee expected=*new* Employee();  
expected.setId(18);  
expected.setName("王一亭");  
expected.setGender("男");  
expected.setBirthday(*new* Date(91, Calendar.*FEBRUARY*,1));  
expected.setIdCard("610144199102014569");  
expected.setWedlock("已婚");  
expected.setNationId(1);  
expected.setNativePlace("广东");  
expected.setPoliticId(6);  
expected.setEmail("zhangliming@qq.com");  
expected.setPhone("18979994478");  
expected.setAddress("广东珠海");  
expected.setDepartmentId(91);  
expected.setJobLevelId(15);  
expected.setPosId(33);  
expected.setEngageForm("劳动合同");  
expected.setTiptopDegree("高中");  
expected.setSpecialty("考古");  
expected.setSchool("清华大学");  
expected.setBeginDate(*new* Date(118, Calendar.*JANUARY*,1));  
expected.setWorkState("在职");  
expected.setWorkID("00000018");  
expected.setContractTerm((*double*) 7);  
expected.setConversionTime(*new* Date(118, Calendar.*APRIL*,1));  
expected.setBeginContract(*new* Date(118, Calendar.*JANUARY*,1));  
expected.setEndContract(*new* Date(125, Calendar.*JANUARY*,30));  
Employee result = employeeService.getEmployeeById(18);

相同

1. 主键不在表中

参数：2531

预期结果：null

1. 测试用例执行与结果
2. getEmployeeByPage方法

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 测试模块/对象 | | org.javaboy.vhr.service.EmployeeService#getEmployeeByPage | | | | |
| 是否有测试代码 | | 有 | 测试代码 | org.javaboy.vhr.service.EmployeeServiceTest  #getEmployeeByPage | | |
| 前提条件（如有） | | 无 | | | | |
| 测试用例序号 | 预期结果 | | | | 实际结果 | 备注 |
| 1 | 617 | | | | 617 |  |
| 2 | 0 | | | | 0 |  |
| 3 | BadSqlGrammarException | | | | BadSqlGrammarException |  |
| 4 | 10 | | | | 10 |  |
| 5 | 2 | | | | 2 |  |
| 6 | 3 | | | | 3 |  |
| 7 | 4 | | | | 4 |  |
| 8 | 6 | | | | 6 |  |
| 9 | 2 | | | | 2 |  |
| 10 | 3 | | | | 3 |  |

1. addEmp方法

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 测试模块/对象 | | org.javaboy.vhr.service.EmployeeService#addEmp | | | | |
| 是否有测试代码 | | 有 | 测试代码 | org.javaboy.vhr.service.EmployeeServiceTest  #addEmp | | |
| 前提条件（如有） | | 无 | | | | |
| 测试用例序号 | 预期结果 | | | | 实际结果 | 备注 |
| 1 | 1 | | | | 1 | 主键自增 |
| 2 | DuplicateKeyException | | | | DuplicateKeyException |  |
| 3 | 1 | | | | 1 |  |

1. maxWorkID方法

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 测试模块/对象 | | org.javaboy.vhr.service.EmployeeService#maxWorkID | | | | |
| 是否有测试代码 | | 有 | 测试代码 | org.javaboy.vhr.service.EmployeeServiceTest  #maxWorkID | | |
| 前提条件（如有） | | 无 | | | | |
| 测试用例序号 | 预期结果 | | | | 实际结果 | 备注 |
| 1 | 65 | | | | 65 |  |

1. deleteEmpByEid方法

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 测试模块/对象 | | org.javaboy.vhr.service.EmployeeService#deleteEmpByEid | | | | |
| 是否有测试代码 | | 有 | 测试代码 | org.javaboy.vhr.service.EmployeeServiceTest  #deleteEmpByEid | | |
| 前提条件（如有） | | 执行测试用例之前需要在employee表中插入一条主键为2000的记录 | | | | |
| 测试用例序号 | 预期结果 | | | | 实际结果 | 备注 |
| 1 | 1 | | | | 1 |  |
| 2 | 0 | | | | 0 |  |

1. updateEmp方法

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 测试模块/对象 | | org.javaboy.vhr.service.EmployeeService#updateEmp | | | | |
| 是否有测试代码 | | 有 | 测试代码 | org.javaboy.vhr.service.EmployeeServiceTest  #updateEmp | | |
| 前提条件（如有） | | 无 | | | | |
| 测试用例序号 | 预期结果 | | | | 实际结果 | 备注 |
| 1 | 1 | | | | 1 |  |
| 2 | 0 | | | | 0 |  |
| 3 | BadSqlGrammarException | | | | BadSqlGrammarException |  |
| 4 | BadSqlGrammarException | | | | BadSqlGrammarException |  |

1. addEmps方法

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 测试模块/对象 | | org.javaboy.vhr.service.EmployeeService#addEmps | | | | |
| 是否有测试代码 | | 有 | 测试代码 | org.javaboy.vhr.service.EmployeeServiceTest  #addEmps | | |
| 前提条件（如有） | | 无 | | | | |
| 测试用例序号 | 预期结果 | | | | 实际结果 | 备注 |
| 1 | BadSqlGrammarException | | | | BadSqlGrammarException |  |
| 2 | 1 | | | | 1 |  |

1. getEmployeeByPageWithSalary方法

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 测试模块/对象 | | org.javaboy.vhr.service.EmployeeService#  getEmployeeByPageWithSalary | | | | |
| 是否有测试代码 | | 有 | 测试代码 | org.javaboy.vhr.service.EmployeeServiceTest  #getEmployeeByPageWithSalary | | |
| 前提条件（如有） | | 无 | | | | |
| 测试用例序号 | 预期结果 | | | | 实际结果 | 备注 |
| 1 | 617 | | | | 617 |  |
| 2 | 0 | | | | 0 |  |
| 3 | BadSqlGrammarException | | | | BadSqlGrammarException |  |

1. updateEmployeeSalaryById方法

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 测试模块/对象 | | org.javaboy.vhr.service.EmployeeService#  updateEmployeeSalaryById | | | | |
| 是否有测试代码 | | 有 | 测试代码 | org.javaboy.vhr.service.EmployeeServiceTest  #updateEmployeeSalaryById | | |
| 前提条件（如有） | | 无 | | | | |
| 测试用例序号 | 预期结果 | | | | 实际结果 | 备注 |
| 1 | 2 | | | | 2 |  |
| 2 | DataIntegrityViolation  Exception | | | | DataIntegrityViolation  Exception |  |
| 3 | 1 | | | | 1 |  |
| 4 | DataIntegrityViolation  Exception | | | | DataIntegrityViolation  Exception |  |

1. getEmployeeById方法

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 测试模块/对象 | | org.javaboy.vhr.service.EmployeeService#getEmployeeById | | | | |
| 是否有测试代码 | | 有 | 测试代码 | org.javaboy.vhr.service.EmployeeServiceTest  #getEmployeeById | | |
| 前提条件（如有） | | 无 | | | | |
| 测试用例序号 | 预期结果 | | | | 实际结果 | 备注 |
| 1 | 见代码 | | | | 见代码 | FailNotEqual |
| 2 | Null | | | | Null |  |

1. failure原因分析及其追踪

在测试getEmployeeById方法时测试用例执行失败，经过检查，Employee实体类不仅包含数据库employee表中所有字段，并且还将表中外键引用的表的实体类作为属性包含在内，而在编写equals方法时使用了自动生成的方法，将其他实体类也纳入判断条件中，而在构造expected结果时并没有考虑。

Bug修复：修改Employee类的equals方法，改为只判断employee表中的字段值是否相等

@Override  
*public boolean* equals(Object o) {  
 *if* (*this* == o) *return true*;  
 *if* (o == *null* || getClass() != o.getClass()) *return false*;  
 Employee employee = (Employee) o;  
 *return* Objects.*equals*(id, employee.id) &&  
 Objects.*equals*(name, employee.name) &&  
 Objects.*equals*(gender, employee.gender) &&  
 Objects.*equals*(birthday, employee.birthday) &&  
 Objects.*equals*(idCard, employee.idCard) &&  
 Objects.*equals*(wedlock, employee.wedlock) &&  
 Objects.*equals*(nationId, employee.nationId) &&  
 Objects.*equals*(nativePlace, employee.nativePlace) &&  
 Objects.*equals*(politicId, employee.politicId) &&  
 Objects.*equals*(email, employee.email) &&  
 Objects.*equals*(phone, employee.phone) &&  
 Objects.*equals*(address, employee.address) &&  
 Objects.*equals*(departmentId, employee.departmentId) &&  
 Objects.*equals*(jobLevelId, employee.jobLevelId) &&  
 Objects.*equals*(posId, employee.posId) &&  
 Objects.*equals*(engageForm, employee.engageForm) &&  
 Objects.*equals*(tiptopDegree, employee.tiptopDegree) &&  
 Objects.*equals*(specialty, employee.specialty) &&  
 Objects.*equals*(school, employee.school) &&  
 Objects.*equals*(beginDate, employee.beginDate) &&  
 Objects.*equals*(workState, employee.workState) &&  
 Objects.*equals*(workID, employee.workID) &&  
 Objects.*equals*(contractTerm, employee.contractTerm) &&  
 Objects.*equals*(conversionTime, employee.conversionTime) &&  
 Objects.*equals*(notWorkDate, employee.notWorkDate) &&  
 Objects.*equals*(beginContract, employee.beginContract) &&  
 Objects.*equals*(endContract, employee.endContract) &&  
 Objects.*equals*(workAge, employee.workAge) ;  
}

再次执行测试用例，测试通过

1. 测试结果分析

根据上表与后续的修改，我们可以看出实际结果与预期结果完全一致，说明该类的功能符合我们对其的预期，表明该单元通过了单元测试。