（续上）

# 前端

## 分组模块

start.setTime(start.getTime() - 3600 \* 1000 \* 24 \* 7);

picker.$emit('pick', [start, end]);

}

}, {

text: '最近一个月',

onClick(picker) {

const end = new Date();

const start = new Date();

start.setTime(start.getTime() - 3600 \* 1000 \* 24 \* 30);

picker.$emit('pick', [start, end]);

}

}, {

text: '最近三个月',

onClick(picker) {

const end = new Date();

const start = new Date();

start.setTime(start.getTime() - 3600 \* 1000 \* 24 \* 90);

picker.$emit('pick', [start, end]);

}

}]

},

value6: '',

value7: '',

showEditModal: false,

taskStatusOptions: [],

suggestedGroupNames: [],

suggestedGroupLeaders: [],

wrappedSuggestedGroupLeader: '',

stages: [],

groupColumns: [

{

type: 'selection',

width: 60,

align: 'center'

},

{

title: '序号',

type: 'index',

width: 70,

align: 'center'

},

{

title: '队名',

align: 'center',

key: 'groupName'

},

{

title: '分组编号',

align: 'center',

width: 150,

key: 'arrangementId'

},

{

title: '组长',

align: 'center',

render: function (h, params) {

return h('span', {}, params.row.groupLeader.studentName)

}

},

{

title: '任务',

align: 'center',

render: function (h, params) {

return h('span', {}, vm.renderTask(params.row.dataMiningTask.taskName))

}

},

{

title: '建队时间',

align: 'center',

key: 'builtTime'

},

{

title: '组员数',

align: 'center',

key:'memberSize'

},

{

title: '任务状态',

align: 'center',

render: function (h, params) {

return h('Tag', {

props: vm.renderTaskStatusTag(params.row.taskStatus)

}, vm.findStatus(params.row.taskStatus));

}

},

{

title: '操作',

align: 'center',

width: 200,

render: (h, params) => {

return h('ButtonGroup', [

h('Button', {

props: {

icon:'ios-search',

size:'small'

},

on: {

click: () => {

this.handleCheck(params.index)

}

}

}),

h('Button', {

props: {

icon:'android-delete',

size:'small'

},

on: {

click: () => {

this.handleDelete(params.row,params.index);

}

}

}),

h('Button', {

props: {

icon:'edit',

size:'small'

},

on: {

click: () => {

this.handleEdit(params.index)

}

}

})

]);

}

},

],

total: null,

listLoading: false,

listQuery: {

groupName: "",

beginDate: '',

endDate: '',

leaderStudentId: '',

page: 0,

size: 20,

taskStatus: '',

sort: "builtTime,ASC",

},

detailTargetIndex: null,

temp: {

studentId: '',

studentName: '',

grade: '',

className: '',

profession: '',

status: {

statusId: 3,

chineseValue: '空闲',

englishValue: 'available',

},

favorite: {

favoriteId: 0,

chineseValue: '已收藏',

englishValue: 'favorite'

},

finishedTaskCount: 0

},

selectionIds: [],

selectedGroups: [],

groupList: [],

sortOptions:

[{label: '按建立时间升序', key: 'builtTime,ASC'},

{label: '按建立时间降序', key: 'builtTime,DESC'}],

multipleSelection: [],

gradeOptions: [],

professionOptions: [],

classNameOptions: [],

};

},

created() {

this.getGroupList();

this.getSuggestedGroupNames();

this.getSuggestedGroupLeaders();

this.getTaskStatusOptions();

this.getAllStudents();

},

methods: {

getAllStudents() {

let vm = this;

fetchStudentList({fetch: true}).then(res => {

vm.allStudents = res.content;

}).catch(error => {

});

},

findStatus(value){

for(let i = 0;i< this.taskStatusOptions.length;i++){

if (this.taskStatusOptions[i].value === value)

return this.taskStatusOptions[i].description;

}

},

handleClosed(){

this.showEditModal = false;

},

getSuggestedGroupNames() {

let vm = this;

getGroupNames().then(res => {

vm.suggestedGroupNames = res.map(r => {

return {

value: r,

name: r

}

});

})

},

getSuggestedGroupLeaders() {

let vm = this;

getGroupLeaders().then(res => {

vm.suggestedGroupLeaders = res.map(r => {

return {

value: r.studentId + ' - ' + r.studentName + ' - ' + r.className + ' - ' + r.profession,

info: r

}

});

}).catch(error => {

});

},

getTaskStatusOptions() {

let vm = this;

getTaskStatus().then((res) => {

vm.taskStatusOptions = res;

}).catch(error => {

});

},

groupNameSearch(queryString, cb) {

var suggestedGroupNames = this.suggestedGroupNames;

var results = queryString ? suggestedGroupNames.filter(this.createFilter(queryString)) : suggestedGroupNames;

cb(results);

},

createFilter(queryString) {

return (s) => {

var reg = new RegExp(queryString.toLowerCase());

return s.value.toLowerCase().match(reg);

};

},

groupLeaderSearch(queryString, cb) {

var suggestedGroupLeaders = this.suggestedGroupLeaders;

var results = queryString ? suggestedGroupLeaders.filter(this.createFilter(queryString)) : suggestedGroupLeaders;

cb(results);

},

handleLeaderFilter(item) {

this.listQuery.leaderStudentId = item.info.studentId;

},

handleSelectionChange(val) {

this.multipleSelection = val;

},

handleRowClicked(row, event, column) {

let lable = column.label;

if (lable === '操作' || lable === '算法配置') {

return;

}

this.$refs.studentTable.toggleRowSelection(row);

this.isDisplayFavoriteColumn = !this.isDisplayFavoriteColumn;

},

getGroupList() {

let that = this;

this.listLoading = true;

getGroupList(Object.assign({}, this.listQuery)).then(response => {

this.groupList = response.content;

this.total = response.totalElements;

this.listLoading = false;

}).catch(error => {

})

},

handleSizeChange(val) {

if (this.listQuery.size === val) {

return

}

this.detailTargetIndex = null;

this.listQuery.size = val;

this.getGroupList();

},

handleCurrentChange(val) {

if (this.listQuery.page === val - 1) {

return

}

this.detailTargetIndex = null;

this.listQuery.page = val - 1;

this.getGroupList();

},

handleSelectionChange(selections) {

this.selectedGroups = selections;

},

handTimeFilter(val) {

this.listQuery.beginDate = val[0];

this.listQuery.endDate = val[1];

},

handleFilter() {

this.getGroupList();

},

handleGroupLeaderFilter() {

this.listQuery.leaderStudentId = '';

},

handleEdit(index) {

this.detailTargetIndex = index;

//this.getRefStages(this.groupList[index].dataMiningTask.taskId);

this.getMembers(this.groupList[index].groupId, index);

this.showEditModal = true;

},

handleCheck(index) {

this.detailTargetIndex = index;

this.getRefStages(this.groupList[index].dataMiningTask.taskId);

this.getMembers(this.groupList[index].groupId, index);

let el = document.getElementById("group-view");

el.scrollIntoView();

},

handleDelete(group,index) {

let vm = this;

let wrapGroups = [];

wrapGroups.push(group.groupId);

this.$confirm('此操作将删除名为 ' + group.groupName + ' 的分组信息, 是否继续?', '确定删除', {

confirmButtonText: '确定',

cancelButtonText: '取消',

type: 'warning'

}).then(() => {

const loading = vm.$loading({

lock: true,

text: 'Loading',

spinner: 'el-icon-loading',

background: 'rgba(0, 0, 0, 0.7)'

});

deleteGroups(wrapGroups).then(() => {

vm.$message.success('删除成功');

vm.groupList.splice(index, 1);

loading.close();

}).catch(error => {

})

}).catch(() => {

this.$message.info('取消删除');

});

},

handleBatchDelete() {

let vm = this;

this.$confirm('此操作将删除已完成分组信息, 是否继续?', '确定删除', {

confirmButtonText: '确定',

cancelButtonText: '取消',

type: 'warning'

}).then(() => {

const loading = vm.$loading({

lock: true,

text: 'Loading',

spinner: 'el-icon-loading',

background: 'rgba(0, 0, 0, 0.7)'

});

deleteGroups(this.\_selectionIds).then(() => {

vm.$message.success('删除成功');

loading.close();

this.getGroupList();

}).catch(error => {

})

})

},

handleUpdateConfirm(groupDto) {

let vm = this;

updateGroup(groupDto).then(res => {

vm.groupList.splice(vm.detailTargetIndex,1, res);

vm.$message({

showClose: true,

message: '更新分组信息成功',

type: 'success'

});

vm.detailTargetIndex = null;

}).catch(error => {

});

},

getMembers(groupId, index) {

let vm = this;

let current = vm.groupList[index];

if(current.groupMembers ===undefined) {

getMembers(groupId).then(res => {

vm.$set(current, 'groupMembers', res);

}).catch(error => {

})

}

},

getRefStages(taskId){

let vm = this;

getRefStages(taskId).then(stages => {

vm.stages = stages;

}).catch(error => {

});

},

resetTemp() {

this.temp = {

studentId: '',

studentName: '',

grade: '',

className: '',

profession: '',

status: {

statusId: 3,

chineseValue: '空闲',

englishValue: 'available',

},

favorite: {

favoriteId: 0,

chineseValue: '已收藏',

englishValue: 'favorite'

},

finishedTaskCount: 0

};

},

renderTask(taskName) {

if (taskName === undefined || taskName === '') {

return '无';

}

else {

return taskName;

}

},

renderTaskStatusTag(taskStatus) {

let tagColor = '';

switch (taskStatus) {

case 1:

tagColor = 'yellow';

break;

case 2:

tagColor = '#EF6AFF';

break;

case 3:

tagColor = 'green';

break;

case 4:

tagColor = 'blue';

break;

case 5:

tagColor = 'red';

break;

case 6:

tagColor = '#25dc72';

break;

}

return {

type: 'dot',

color: tagColor

};

}

},

computed: {

fixPage() {

return this.listQuery.page + 1;

},

\_length() {

return this.groupList.length;

},

\_selectionIds() {

return this.selectedGroups.map(selection => selection.groupId);

},

\_wrappedDetailTarget() {

if (!this.\_length || this.detailTargetIndex === null) {

return null;

}

let wrap = [];

wrap.push(this.groupList[this.detailTargetIndex]);

return wrap;

},

\_updatingTargetGroup() {

return this.\_wrappedDetailTarget ? Object.assign({},this.\_wrappedDetailTarget[0]) : this.groupModel;

},

},

}

</script>

<style lang="scss">

.group-list-container {

margin-top: 90px;

}

.btn-import-container {

padding-right: 20px;

float: right;

.btn-item {

display: inline-block;

vertical-align: middle;

margin-bottom: 10px;

margin-left: 5px;

}

}

.test {

padding: 20px;

}

.title {

margin-bottom: 20px;

width: 22%;

font: bold 36px 微软雅黑;

}

div {

background-color: transparent;

margin: 0px auto;

}

</style>

# 后端

## 数据分析模块

package com.vero.dm.service.impl;

import static com.vero.dm.repository.specifications.ContainerSpecifications.findContainersByCollectionId;

import static com.vero.dm.util.PathUtils.concat;

import java.io.File;

import java.io.IOException;

import java.util.\*;

import org.apache.commons.io.FileUtils;

import org.springframework.beans.BeanUtils;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.data.domain.Page;

import org.springframework.data.domain.Pageable;

import org.springframework.http.converter.HttpMessageNotReadableException;

import org.springframework.scheduling.concurrent.ThreadPoolTaskExecutor;

import org.springframework.stereotype.Service;

import org.springframework.web.multipart.MultipartFile;

import com.vero.dm.model.\*;

import com.vero.dm.repository.dto.CollectionDto;

import com.vero.dm.service.DataSetCollectionService;

import com.vero.dm.service.constant.ResourcePath;

import com.vero.dm.util.PathUtils;

import lombok.extern.slf4j.Slf4j;

/\*\*

\* @author 刘祥德 qq313700046@icloud.com .

\* @date created in 1:34 2017/7/6.

\* @description

\* @modified by:

\*/

@Service

@Slf4j

public class DataSetCollectionServiceImpl extends AbstractBaseServiceImpl<DataSetCollection, Integer> implements DataSetCollectionService

{

private ThreadPoolTaskExecutor threadPoolTaskExecutor;

public DataSetCollection fectchAllDataSetContainer()

{

return null;

}

@Autowired

public void setThreadPoolTaskExecutor(ThreadPoolTaskExecutor threadPoolTaskExecutor) {

this.threadPoolTaskExecutor = threadPoolTaskExecutor;

}

@Override

public DataSetCollection findById(Integer id)

{

return collectionJpaRepository.findOne(id);

}

// public List<DataSetCollection> saveBatchCollection(Set<DataSetCollection> collections)

//// {

//// return collectionJpaRepository.save(collections);

//// }

@Override

public Page<DataSetCollection> getPageableCollection(Pageable pageable)

{

// Page<DataSetCollection> collections = collectionJpaRepository.findAll(pageable);

// collections.getContent().forEach(c -> c.setNumberOfWebHits(c.getNumberOfWebHits()));

// collectionJpaRepository.save(collections.getContent());

return collectionJpaRepository.findAll(pageable);

}

@Override

public DataSetContainer addDataSetContainer(Integer collectionId, MultipartFile multipartFile)

{

DataSetCollection collection = findById(collectionId);

String absolutePath = concat(collection.getDataSetFolderPath(),

multipartFile.getOriginalFilename());

DataSetContainer container = new DataSetContainer();

threadPoolTaskExecutor.execute(() -> {

try {

multipartFile.transferTo(new File(absolutePath));

} catch (IOException e) {

e.printStackTrace();

}

});

String originalFileName = multipartFile.getOriginalFilename();

String fileType = originalFileName.substring(originalFileName.lastIndexOf("."),

originalFileName.length());

log.debug("Uploaded file type is [{}]", fileType);

container.setFileName(multipartFile.getOriginalFilename());

container.setDataSetCollection(findById(collectionId));

container.setFilePath(absolutePath);

container.setFileType(fileType);

container.setSize(multipartFile.getSize());

containerJpaRepository.save(container);

return container;

}

public List<DataSetCollection> findCollectionsByIds(List<Integer> collectionIds)

{

return collectionJpaRepository.findAll(collectionIds);

}

@Override

public List<DataSetContainer> saveOrUpdateContainers(Integer collectionId,

List<Integer> containerIds)

{

// if (containerIds.isEmpty())

// {

// return null;

// }

// List<DataSetContainer> containers = containerDao.fetchContainers(containerIds);

// DataSetCollection collection = findById(collectionId);

// if (collection.getDataSets() == null)

// {

// collection.setDataSets(new LinkedHashSet<DataSetContainer>());

// }

// collection.getDataSets().addAll(containers);

// for (DataSetContainer container : containers)

// {

// container.setDataSetCollection(collection);

// containerDao.update(container);

// }

// return containers;

return null;

}

// public DataSetContainer removeDataSetContainer(String collectionId, String containerId)

// {

// DataSetContainer

// containerJpaRepository.delete(containerId);

// return null;

// }

@Override

public DataSetCollection saveCollection(CollectionDto collectionDto)

{

return collectionJpaRepository.save(convert(collectionDto));

}

@Override

public List<DataSetCollection> saveCollections(List<CollectionDto> collectionDtos)

{

List<DataSetCollection> dataSetCollections = new LinkedList<>();

collectionDtos.forEach(c -> dataSetCollections.add(convert(c)));

return collectionJpaRepository.save(dataSetCollections);

}

@Override

public DataSetCollection deleteByName(String collectionName)

{

// DataSetCollection collection = getCollectionByName(collectionName);

// collectionDao.deleteByName(collectionName);

// return collection;

return null;

}

@Override

public List<DataSetCollection> deleteBatch(List<Integer> collectionIds)

{

List<DataSetCollection> collections = findCollectionsByIds(collectionIds);

// List<String> setFilePaths = collectionJpaRepository.findAllDataSetsFilePaths(collectionIds);

collections.forEach( c -> {

try {

FileUtils.deleteDirectory(new File(c.getDataSetFolderPath()));

} catch (IOException e) {

e.printStackTrace();

}

});

collectionJpaRepository.delete(collections);

return collections;

}

@Override

public DataSetCollection deleteByCollectionId(Integer collectionId)

{

DataSetCollection collection = findById(collectionId);

File dataDir = new File(collection.getDataSetFolderPath());

try {

FileUtils.deleteDirectory(dataDir);

log.info("Delete [{}]:[{}] database index folder.", collection.getCollectionId(),

collection.getCollectionName());

} catch (IOException e) {

e.printStackTrace();

}

collectionJpaRepository.delete(collectionId);

return collection;

}

@Override

public DataSetCollection updateCollection(DataSetCollection collection)

{

return collectionJpaRepository.saveAndFlush(collection);

}

// @Override

// public DataSetCollection updateCollection(CollectionDto dataSetCollection)

// {

//// DataSetCollection collection = this.findById(dataSetCollection.getCollectionId());

//// BeanUtils.copyProperties(dataSetCollection, collection);

//// updateCollectionMultipleTypes(dataSetCollection, collection);

//// this.update(collection);

//// return collection;

// return null;

// }

@Override

public Page<DataSetContainer> getContainers(Integer collectionId, Pageable pageable)

{

DataSetCollection collection = findById(collectionId);

collection.setNumberOfWebHits(collection.getNumberOfWebHits() + 1);

collectionJpaRepository.save(collection);

return containerJpaRepository.findAll(findContainersByCollectionId(collectionId),pageable);

}

@Override

public List<String> getCollectionNames()

{

// return collectionDao.getSetNames();

return null;

}

@Override

public DataSetContainer relateContainer(Integer collectionId, Integer containerId)

{

// DataSetContainer container = containerDao.findById(containerId);

// return this.addDataSetContainer(collectionId, container);

return null;

}

@Override

public Map<String, List<?>> getOptions()

{

List<AreaType> areaTypeOptions = areaJpaRepository.findAll();

List<DataSetCharacteristic> dataSetCharOptions = setCharJpaRepository.findAll();

List<AttributeCharacteristic> attrCharOptions = attributeCharJpaRepository.findAll();

List<AssociatedTask> associatedTaskOptions = associatedTaskJpaRepository.findAll();

Map<String, List<?>> optionsMap = new LinkedHashMap<String, List<?>>();

optionsMap.put("dataSetCharOptions", dataSetCharOptions);

optionsMap.put("attrCharOptions", attrCharOptions);

optionsMap.put("associatedTaskOptions", associatedTaskOptions);

optionsMap.put("areaTypeOptions", areaTypeOptions);

return optionsMap;

}

public DataSetCollection convert(CollectionDto source)

{

List<DataSetDescription> descriptions = descriptionJpaRepository.findAll(

source.getDescriptionIds());

AreaType areaType = areaJpaRepository.findOne(source.getAreaId());

List<AttributeCharacteristic> attributeCharacteristics = attributeCharJpaRepository.findAll(

source.getAttributeCharIds());

List<AssociatedTask> associatedTasks = associatedTaskJpaRepository.findAll(

source.getAssociatedTaskIds());

List<DataSetCharacteristic> dataSetCharacteristics = setCharJpaRepository.findAll(

source.getDataSetCharIds());

Boolean isValidDtoParams = isValidDtoParams(areaType, attributeCharacteristics,

associatedTasks, dataSetCharacteristics);

if (!isValidDtoParams)

{

throw new HttpMessageNotReadableException(

"The provided request body is not readable!");

}

return convertResult(source, descriptions, areaType, attributeCharacteristics,

associatedTasks, dataSetCharacteristics);

}

private DataSetCollection convertResult(CollectionDto source,

List<DataSetDescription> descriptions,

AreaType areaType,

List<AttributeCharacteristic> attributeCharacteristics,

List<AssociatedTask> associatedTasks,

List<DataSetCharacteristic> dataSetCharacteristics)

{

if (source.getIsUpdated())

{

return updatedCollectionModel(source);

}

else

{

return newCollectionModel(source, descriptions, areaType, attributeCharacteristics,

associatedTasks, dataSetCharacteristics);

}

}

private boolean isValidDtoParams(AreaType areaType,

List<AttributeCharacteristic> attributeCharacteristics,

List<AssociatedTask> associatedTasks,

List<DataSetCharacteristic> dataSetCharacteristics)

{

return areaType != null && attributeCharacteristics != null && associatedTasks != null

&& dataSetCharacteristics != null;

}

private DataSetCollection newCollectionModel(CollectionDto source,

List<DataSetDescription> descriptions,

AreaType areaType,

List<AttributeCharacteristic> attributeCharacteristics,

List<AssociatedTask> associatedTasks,

List<DataSetCharacteristic> dataSetCharacteristics)

{

DataSetCollection newCollection = new DataSetCollection();

BeanUtils.copyProperties(source, newCollection);

newCollection.setArea(areaType);

newCollection.setAssociatedTasks(new LinkedHashSet<>(associatedTasks));

newCollection.setAttributeCharacteristics(new LinkedHashSet<>(attributeCharacteristics));

// newCollection.setDescriptions(new LinkedHashSet<>(descriptions));

descriptions.forEach(d -> d.setDataSetCollection(newCollection));

newCollection.setDataSetCharacteristics(new LinkedHashSet<>(dataSetCharacteristics));

collectionJpaRepository.save(newCollection);

String folderPath = concat(ResourcePath.COLLECTION\_PATH, newCollection.getCollectionId().toString());

newCollection.setDataSetFolderPath(PathUtils.getAbsolutePath(folderPath));

log.info("Create new data set base folder:[{}]", folderPath);

return newCollection;

}

private DataSetCollection updatedCollectionModel(CollectionDto source)

{

DataSetCollection collection = collectionJpaRepository.findOne(source.getCollectionId());

BeanUtils.copyProperties(source, collection);

return collection;

}

}

## 学生模块

package com.vero.dm.service.impl;

import static com.vero.dm.repository.specifications.StudentSpecifications.findLeisureStudents;

import static com.vero.dm.repository.specifications.StudentSpecifications.findStudentsWithParams;

import static com.vero.dm.util.PathUtils.concat;

import static com.vero.dm.util.PathUtils.getAbsolutePath;

import java.io.File;

import java.io.IOException;

import java.util.\*;

import java.util.stream.Collectors;

import com.vero.dm.exception.group.StudentNotFoundException;

import com.vero.dm.model.\*;

import org.apache.commons.io.FileUtils;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.data.domain.Page;

import org.springframework.data.domain.PageImpl;

import org.springframework.data.domain.Pageable;

import org.springframework.stereotype.Service;

import org.springframework.web.multipart.MultipartFile;

import com.vero.dm.exception.business.StudentIdDuplicatedException;

import com.vero.dm.exception.error.ExceptionCode;

import com.vero.dm.importer.core.ExcelExporter;

import com.vero.dm.importer.core.ExcelImporter;

import com.vero.dm.importer.core.ExcelModuleManager;

import com.vero.dm.repository.dto.StudentDto;

import com.vero.dm.service.StudentService;

import lombok.extern.slf4j.Slf4j;

/\*\*

\* @author XiangDe Liu qq313700046@icloud.com .

\* @version 1.5 created in

\*/

@Service

@Slf4j

public class StudentServiceImpl extends UserServiceImpl implements StudentService

{

@Autowired

private ExcelImporter<Student> studentExcelImporter;

@Autowired

private ExcelModuleManager excelModuleManager;

@Autowired

private ExcelExporter<Student> studentExcelExporter;

public final static String MAC\_DEFAULT\_ALGORITHM = "HmacSHA256";

public final static String SERCRET\_KEY\_DEFAULT\_ALGORITHM = "HMACSHA256";

@Override

public List<Student> importStudents(MultipartFile file)

{

String relativePath = concat("students", "import");

String dir = getAbsolutePath(relativePath);

String absolutePath = concat(dir, file.getOriginalFilename());

try

{

file.transferTo(new File(absolutePath));

}

catch (IOException e)

{

e.printStackTrace();

}

List<Student> students = studentExcelImporter.importFromExcel(new File(absolutePath));

checkStudentIdDuplication(students);

for (Student student : students)

{

// 初始化用户名密码皆为学号，密码为明文;

student.setPassword(student.getStudentId());

student.setUsername(student.getStudentId());

}

studentJpaRepository.save(students);

log.debug("Import Student Data Form Excel [{}].", file.getOriginalFilename());

return students;

}

@Override

public StudentDto findByUsername(String username) {

Student student = studentJpaRepository.findByUsername(username);

if (student == null) {

return null;

}

return StudentDto.build(student);

}

@Override

public StudentDto findByUserId(String userId) {

Student student = studentJpaRepository.findByUserId(userId);

if (student == null) {

return null;

}

return StudentDto.build(student);

}

private void checkStudentIdDuplication(List<Student> students)

{

List<String> existedStudentIds = getStudentIds();

StringBuilder message = new StringBuilder("[");

students.forEach(s -> {

if (existedStudentIds.contains(s.getStudentId()))

{

message.append(s.getStudentId()).append(",");

}

});

if (!message.toString().equals("["))

{

message.replace(message.length() - 1, message.length(), "]");

message.append(" Student Id is Duplicated.");

log.error(message.toString());

throw new StudentIdDuplicatedException(message.toString(),

ExceptionCode.StudentIdDuplicated);

}

}

@Override

public byte[] handleStudentExcelModuleDownload()

{

String filePath = excelModuleManager.getModulePath(Student.class);

File file = new File(filePath);

if (file.exists() && file.isFile())

{

try

{

return FileUtils.readFileToByteArray(file);

}

catch (IOException e)

{

e.printStackTrace();

}

}

return null;

}

@Override

public List<String> getStudentIds()

{

return studentJpaRepository.findAllStudentIds();

}

@Override

public List<Student> findAllStudents()

{

return studentJpaRepository.findAll();

}

@Override

public Student findStudentById(String id)

{

return studentJpaRepository.findOne(id);

}

@Override

public Student findByStudentId(String studentId)

{

return studentJpaRepository.findByStudentId(studentId);

}

@Override

public List<Student> findByStudentIds(List<String> ids)

{

return studentJpaRepository.findByStudentIds(ids);

}

@Override

public Page<Student> getStudentList(boolean fetch, Pageable pageable, String className,

String profession, String grade, String studentIdPrefix,

String studentName, Date beginDate, Date endDate)

{

if (fetch)

{

return new PageImpl<>(studentJpaRepository.findAll(findStudentsWithParams(className,

profession, grade, studentIdPrefix, studentName)));

}

else if (Objects.isNull(endDate) && Objects.isNull(beginDate))

{

return studentJpaRepository.findAll(

findStudentsWithParams(className, profession, grade, studentIdPrefix, studentName),

pageable);

}

else

{

return studentJpaRepository.findAll(findLeisureStudents(className, profession, grade,

studentIdPrefix, studentName, beginDate, endDate), pageable);

}

}

@Override

public List<Student> getAllLeisureStudents(Pageable pageable, String className,

String profession, String grade,

String studentIdPrefix, String studentName,

Date beginDate, Date endDate)

{

return studentJpaRepository.findAll(findLeisureStudents(className, profession, grade,

studentIdPrefix, studentName, beginDate, endDate));

}

@Override

public StudentDto deleteByStudentId(String studentId)

{

// Student student = studentDao.getStudentById(studentId);

// studentDao.deleteStudentById(studentId);

// return StudentDto.build(student);

return null;

}

public StudentDto save(Student student)

{

this.studentJpaRepository.save(student);

return new StudentDto(student);

}

@Override

public StudentDto update(StudentDto studentDto)

{

// Student student = studentDao.getStudentById(studentDto.getStudentId());

// BeanUtils.copyProperties(studentDto, student, "status", "favorite");

// studentDao.update(student);

// return new StudentDto(student);

return studentDto;

}

@Override

public StudentDto getStudent(String userId, String username) {

StudentDto stuByUserId = findByUserId(userId);

StudentDto stuByUsername = findByUsername(username);

if (Objects.isNull(stuByUserId) && Objects.isNull(stuByUsername)) {

throw new StudentNotFoundException("此用户不具有学生身份", ExceptionCode.StudentNotFound);

}

if (!Objects.isNull(stuByUserId)) {

return stuByUserId;

}

return stuByUsername;

}

@Override

public int markStudents(List<String> studentIds)

{

// return studentDao.markStudents(studentIds);

return 0;

}

@Override

public int unMarkStudents(List<String> studentIds)

{

// return studentDao.unMarkStudents(studentIds);

return 0;

}

@Override

public List<Student> deleteBatchByStudentIds(List<String> studentIds)

{

List<Student> students = studentJpaRepository.findByStudentIds(studentIds);

List<String> studentUserIds = students.stream().map(Student::getUserId).collect(Collectors.toList());

students.forEach(s -> {

s.setMiningGroups(null);

Set<DataMiningGroup> groups = s.getRuleMiningGroups();

groups.forEach(g -> g.setGroupLeader(null));

groupJpaRepository.save(groups);

});

//删除上交的挖掘记录

miningResultRepository.deleteMiningResultByMembers(studentUserIds);

studentJpaRepository.save(students);

studentJpaRepository.deleteBatchStudentsById(studentIds);

return students;

}

@Override

public FavoriteStatus getFavoriteStatusPersisted(Integer statusId)

{

// return studentDao.getFavoriteStatusPersisted(statusId);

return null;

}

@Override

public StudentStatus getStudentStatusPersisted(Integer statusId)

{

// return studentDao.getStudentStatus(statusId);

return null;

}

@Override

public Map<String, List<?>> getStudentPropertiesOptions()

{

Map<String, List<?>> options = new HashMap<>();

options.put("classNameOptions", studentJpaRepository.findClassNameOptions());

options.put("professionOptions", studentJpaRepository.findProfessionOptions());

options.put("gradeOptions", studentJpaRepository.findGradeOptions());

return options;

}

public List<Student> fetchStudentWithoutTasks()

{

return null;

}

}

package com.vero.dm.service.grouper;

import java.util.\*;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.stereotype.Component;

import org.springframework.util.ObjectUtils;

import org.springframework.util.StringUtils;

import com.vero.dm.exception.error.ExceptionCode;

import com.vero.dm.exception.group.StudentNotFoundException;

import com.vero.dm.exception.group.TasksEmptyException;

import com.vero.dm.model.DataMiningGroup;

import com.vero.dm.model.Student;

import com.vero.dm.model.Teacher;

import com.vero.dm.model.enums.MiningTaskStatus;

import com.vero.dm.repository.dto.DividingGroupInfo;

import com.vero.dm.repository.dto.GroupingConfigParams;

import com.vero.dm.service.GroupService;

import com.vero.dm.service.MiningTaskService;

import com.vero.dm.service.StudentService;

import com.vero.dm.service.TeacherService;

import lombok.Setter;

import lombok.extern.slf4j.Slf4j;

/\*\*

\* @author XiangDe Liu qq313700046@icloud.com .

\* @version 1.5 created in 13:52 2018/7/21.

\* @since data-mining-platform

\*/

@Slf4j

@Setter

@Component

public class MiningGrouper

{

private GroupService groupService;

private MiningTaskService taskService;

private StudentService studentService;

private TeacherService teacherService;

private GroupingStrategy groupingStrategy;

private Map<Integer, GroupingStrategy> strategies;

public MiningGrouper()

{

strategies = GroupingStrategyPool.createInstancePool(this);

}

@Autowired

public void setGroupService(GroupService groupService)

{

this.groupService = groupService;

}

@Autowired

public void setTaskService(MiningTaskService taskService)

{

this.taskService = taskService;

}

@Autowired

public void setStudentService(StudentService studentService)

{

this.studentService = studentService;

}

@Autowired

public void setTeacherService(TeacherService teacherService)

{

this.teacherService = teacherService;

}

// 默认简单分组方法

public DividingGroupInfo initDefaultGroupingStrategy(GroupingConfigParams params)

{

List<Student> candidates = getStudentsPrepareDivided(params);

List<String> taskIds = params.getSpecifiedTasks();

switchStrategy(params);

validateInputParams(candidates, taskIds);

checkStrategyConfiguration();

return groupingStrategy.doStrategy(params, candidates, params.getSpecifiedTasks());

}

private void switchStrategy(GroupingConfigParams params)

{

// 切换策略

this.setGroupingStrategy(this.strategies.get(params.getStrategyId()));

}

private void checkStrategyConfiguration()

{

if (groupingStrategy == null)

{

log.debug("The grouping strategy is not configured.User simple grouping strategy.");

groupingStrategy = new SimpleGroupingStrategy(this);

}

}

private void validateInputParams(List<Student> candidates, List<String> taskIds)

{

if (candidates == null || candidates.isEmpty())

{

String message = "找不到请求的学生分组列表";

log.error(message);

throw new StudentNotFoundException(message, ExceptionCode.StudentNotFound);

}

if (taskIds.isEmpty())

{

String message = "待分配的任务不能为空";

log.error(message);

throw new TasksEmptyException(message, ExceptionCode.TasksEmpty);

}

}

// private int nextGradient(GroupingConfigParams params) {

// Random random = new Random();

// return random.nextInt(params.getUpperBound() - params.getLowerBound() + 1) +

// params.getLowerBound();

// }

public DataMiningGroup buildGroup(String builderId, String taskId, String arrangementId,

List<Student> perGroupStudents)

{

DataMiningGroup group = new DataMiningGroup();

group.setGroupLeader(perGroupStudents.get(0));

if (StringUtils.isEmpty(taskId))

{

group.setDataMiningTask(null);

group.setTaskStatus(MiningTaskStatus.toBeAssigned);

}

else

{

group.setDataMiningTask(taskService.findById(taskId));

group.setTaskStatus(MiningTaskStatus.newTask);

}

Teacher teacherBuilder = teacherService.findTeacherById(builderId);

Student studentBuilder = studentService.findStudentById(builderId);

if (!ObjectUtils.isEmpty(teacherBuilder))

{

group.setTeacherBuilder(teacherBuilder);

}

if (!ObjectUtils.isEmpty(studentBuilder))

{

group.setStudentBuilder(studentBuilder);

}

group.setBuiltTime(new Date());

group.setGroupName("Group\_" + arrangementId);

group.setArrangementId(String.valueOf(arrangementId));

group.setGroupMembers(new LinkedHashSet<>(perGroupStudents));

return group;

}

private List<Student> getStudentsPrepareDivided(GroupingConfigParams params)

{

List<String> studentIds = params.getSpecifiedDividingStudents();

// 如果用户不指定要分组的学生，且不忽略当前有任务在身学生

if ((studentIds == null || studentIds.isEmpty()) && !params.getIsIgnoreArrangedTask())

{

// 前端传入的时间参数，要求在此时间端内，学生没有处于其他实践任务分组执行数据挖掘任务；

// 以任务的计划时间为准

// 获取符合要求的学生

return groupService.fetchStudentWithoutGroup(params.getBeginDate(),

params.getEndDate());

}

else if (studentIds != null && !studentIds.isEmpty())

{

return studentService.findByStudentIds(studentIds);

}

else if (params.getIsIgnoreArrangedTask())

{

// 全部学生参与分组

return studentService.findAllStudents();

}

return null;

}

public String cachePreview(List<DataMiningGroup> previewDefaultGroups)

{

String queryKey = UUID.randomUUID().toString();

groupService.setPreviewGroupCache(queryKey, previewDefaultGroups);

return queryKey;

}

}

## 分组模块

package com.vero.dm.service.impl;

import static com.vero.dm.repository.specifications.GroupSpecifications.groupSpec;

import java.util.\*;

import java.util.concurrent.ConcurrentHashMap;

import java.util.stream.Collectors;

import javax.persistence.EntityNotFoundException;

import org.springframework.beans.BeanUtils;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.data.domain.Page;

import org.springframework.data.domain.PageImpl;

import org.springframework.data.domain.Pageable;

import org.springframework.data.jpa.domain.Specification;

import org.springframework.stereotype.Service;

import org.springframework.util.StringUtils;

import com.vero.dm.exception.error.ExceptionCode;

import com.vero.dm.exception.group.PreviewGroupsNotFoundException;

import com.vero.dm.model.\*;

import com.vero.dm.model.enums.MiningTaskStatus;

import com.vero.dm.model.enums.ResultState;

import com.vero.dm.model.enums.StatusObject;

import com.vero.dm.model.enums.TaskProgressStatus;

import com.vero.dm.repository.dto.DataMiningGroupDto;

import com.vero.dm.repository.dto.DividingGroupInfo;

import com.vero.dm.repository.dto.GroupingConfigParams;

import com.vero.dm.repository.dto.StudentDto;

import com.vero.dm.service.GroupService;

import com.vero.dm.service.grouper.MiningGrouper;

import lombok.extern.slf4j.Slf4j;

/\*\*

\* @author XiangDe Liu qq313700046@icloud.com .

\* @version 1.5 created in

\*/

@Service

@Slf4j

public class GroupServiceImpl extends AbstractBaseServiceImpl<DataMiningGroup, String> implements GroupService

{

private Integer gradient = 12;

private MiningGrouper grouper;

/\*\*

\* 缓存分组列表

\*/

private Map<String, List<DataMiningGroup>> cacheGroups = new ConcurrentHashMap<>();

public void setGradient(Integer gradient)

{

this.gradient = gradient;

}

@Autowired

public void setGrouper(MiningGrouper grouper)

{

this.grouper = grouper;

}

@Override

public DataMiningGroup createGroup(DataMiningGroupDto groupDto)

{

DataMiningGroup group = new DataMiningGroup();

Teacher builder = teacherJpaRepository.findOne(groupDto.getBuilderId());

Student leader = studentJpaRepository.findByStudentId(groupDto.getLeaderId());

List<Student> members = studentJpaRepository.findByStudentIds(groupDto.getMemberIds());

DataMiningTask task = taskJpaRepository.findOne(groupDto.getTaskId());

if (!members.contains(leader))

{

log.error("The group leader must be one of the members.");

return null;

}

BeanUtils.copyProperties(groupDto, group);

updateTaskStatus(groupDto, group);

group.setDataMiningTask(task);

group.setTeacherBuilder(builder);

group.setGroupMembers(new LinkedHashSet<>(members));

group.setGroupLeader(leader);

group.setBuiltTime(new Date());

group = groupJpaRepository.save(group);

this.arrangeTask(group.getGroupId(), groupDto.getTaskId());

return group;

}

@Override

public DataMiningGroupDto updateGroup(DataMiningGroupDto groupDto)

{

DataMiningGroup group = groupJpaRepository.findOne(groupDto.getGroupId());

BeanUtils.copyProperties(groupDto, group);

this.configureGroupMembers(groupDto.getGroupId(), groupDto.getMemberIds());

updateTaskStatus(groupDto, group);

this.arrangeTask(groupDto.getGroupId(), groupDto.getTaskId());

this.updateLeader(groupDto.getLeaderId(), groupDto.getGroupId());

groupJpaRepository.saveAndFlush(group);

return DataMiningGroupDto.build(group);

}

@Override

public void setPreviewGroupCache(String queryKey, List<DataMiningGroup> groups)

{

this.cacheGroups.put(queryKey, groups);

}

private void updateTaskStatus(DataMiningGroupDto groupDto, DataMiningGroup group)

{

group.setTaskStatus(MiningTaskStatus.map.get(groupDto.getTaskStatus()));

DataMiningTask pre = group.getDataMiningTask();

if (!Objects.isNull(pre))

{

pre.setProgressStatus(TaskProgressStatus.toBeAssigned);

taskJpaRepository.save(pre);

}

if (!StringUtils.isEmpty(groupDto.getTaskId()))

{

if (group.getTaskStatus().equals(MiningTaskStatus.toBeAssigned))

{

group.setTaskStatus(MiningTaskStatus.newTask);

}

}

}

@Override

public Page<DataMiningGroupDto> fetchPageableGroups(Pageable pageable, String groupName,

Date beginDate, Date endDate,

String leaderStudentId,

MiningTaskStatus taskStatus, Boolean fetch)

{

Specification<DataMiningGroup> condition = groupSpec(groupName, beginDate, endDate,

leaderStudentId, taskStatus);

if (fetch)

{

return new PageImpl<>(DataMiningGroupDto.build(groupJpaRepository.findAll(condition)));

}

return new PageImpl<>(DataMiningGroupDto.build(

groupJpaRepository.findAll(condition, pageable).getContent()));

}

public Page<DataMiningGroup> fetchPageableGroups(Pageable pageable)

{

return groupJpaRepository.findAll(pageable);

}

// 默认简单分组方法

public DividingGroupInfo getDividingGroupInfo(GroupingConfigParams params)

{

return grouper.initDefaultGroupingStrategy(params);

}

@Override

public List<DataMiningGroupDto> sureDividingGroupRequest(String queryKey)

{

List<DataMiningGroup> groups = cacheGroups.get(queryKey);

if (groups == null || groups.isEmpty())

{

String message = "Could not query corresponding groups information.Make sure your have init divide strategy firstly.";

log.error(message);

throw new PreviewGroupsNotFoundException(message, ExceptionCode.PreviewGroupsNotFound);

}

groups.forEach(g -> {

DataMiningTask t = taskJpaRepository.findOne(g.getDataMiningTask().getTaskId());

t.setProgressStatus(TaskProgressStatus.assigned);

taskJpaRepository.save(t);

});

List<DataMiningGroup> createdGroups = groupJpaRepository.save(groups);

createdGroups.forEach(g -> buildMiningResultRecord(g.getDataMiningTask().getTaskId(), g.getGroupId()));

cacheGroups.remove(queryKey);

return DataMiningGroupDto.build(groups);

}

@Override

public List<Student> fetchStudentWithoutGroup(Date begin, Date end)

{

return studentJpaRepository.fetchStudentWithoutGroup(begin, end);

}

@Override

public DataMiningGroup fetchGroupDetails(String groupId)

{

return groupJpaRepository.findOne(groupId);

}

@Override

public List<DataMiningGroup> fetchGroupDetails(List<String> groupIds)

{

return groupJpaRepository.findAll(groupIds);

}

@Override

public DataMiningGroup deleteMiningGroupById(String groupId)

{

DataMiningGroup group = fetchGroupDetails(groupId);

cancelGroupConfiguration(group);

groupJpaRepository.delete(groupId);

return group;

}

private void cancelGroupConfiguration(DataMiningGroup group)

{

group.setGroupLeader(null);

group.setGroupMembers(null);

group.setStudentBuilder(null);

if (!Objects.isNull(group.getDataMiningTask()))

{

group.getDataMiningTask().setProgressStatus(TaskProgressStatus.toBeAssigned);

taskJpaRepository.save(group.getDataMiningTask());

}

clearResult(group);

group.setDataMiningTask(null);

groupJpaRepository.save(group);

}

public int clearResult(DataMiningGroup group)

{

if (Objects.isNull(group.getGroupMembers()))

{

return 0;

}

List<String> memberIds = group.getGroupMembers().stream().map(User::getUserId).collect(

Collectors.toList());

// 清除之前分组的挖掘记录8

return miningResultRepository.deleteMiningResultByMembers(memberIds);

}

@Override

public List<DataMiningGroup> deleteGroupBatch(List<String> groupIds)

{

List<DataMiningGroup> groups = fetchGroupDetails(groupIds);

groups.forEach(this::cancelGroupConfiguration);

groupJpaRepository.delete(groups);

return groups;

}

@Override

public StudentDto updateLeader(String studentId, String groupId)

{

DataMiningGroup group = fetchGroupDetails(groupId);

Student leader = studentJpaRepository.findByStudentId(studentId);

if (leader == null)

{

log.error("Could not found student request to be set leader.");

throw new EntityNotFoundException("找不到要被设为组长的学生信息");

}

if (!group.getGroupMembers().contains(leader))

{

throw new IllegalArgumentException("组长必须为该组的一名成员");

}

group.setGroupLeader(leader);

groupJpaRepository.save(group);

return StudentDto.build(group.getGroupLeader());

}

@Override

public List<StudentDto> fetchGroupMembers(String groupId)

{

Set<Student> students = fetchGroupDetails(groupId).getGroupMembers();

List<StudentDto> studentDtos = new LinkedList<>();

students.forEach(s -> studentDtos.add(StudentDto.build(s)));

return studentDtos;

}

@Override

public List<StudentDto> configureGroupMembers(String groupId, List<String> studentIds)

{

List<Student> students = studentJpaRepository.findByStudentIds(studentIds);

DataMiningGroup group = groupJpaRepository.findOne(groupId);

// if (!students.contains(group.getGroupLeader()))

// {

// group.setGroupLeader(null);

// }

group.setGroupMembers(new LinkedHashSet<>(students));

groupJpaRepository.save(group);

return StudentDto.build(students);

}

@Override

public List<String> fetchGroupNames()

{

return groupJpaRepository.findGroupNames();

}

@Override

public List<Student> fetchGroupLeaders()

{

return groupJpaRepository.findLeaders();

}

@Override

public DataMiningTask arrangeTask(String groupId, String taskId)

{

if (StringUtils.isEmpty(groupId) || StringUtils.isEmpty(taskId))

{

return null;

}

DataMiningTask task = taskJpaRepository.findOne(taskId);

DataMiningGroup group = fetchGroupDetails(groupId);

group.setDataMiningTask(task);

TaskProgressStatus status = task.getProgressStatus();

// 更新当前任务状态

Boolean isAssigned = status.equals(TaskProgressStatus.newCreate)

|| status.equals(TaskProgressStatus.toBeAssigned);

if (isAssigned)

{

task.setProgressStatus(TaskProgressStatus.assigned);

}

taskJpaRepository.saveAndFlush(task);

groupJpaRepository.saveAndFlush(group);

// 清除之前分组的挖掘记录

int deleteResults = clearResult(group);

log.info("删除[{}]先前的{}条结果记录.", group.getGroupName(), deleteResults);

buildMiningResultRecord(task.getTaskId(), groupId);

return task;

}

public void buildMiningResultRecord(String taskId, String groupId)

{

DataMiningTask task = taskJpaRepository.findOne(taskId);

Set<MiningTaskStage> stages = task.getStages();

List<MiningResult> results = new ArrayList<>();

DataMiningGroup g = groupJpaRepository.findOne(groupId);

for (MiningTaskStage stage : stages)

{

// 给每个组员、每个阶段分配一个挖掘结果记录

Set<Student> members = g.getGroupMembers();

for (Student member : members)

{

MiningResult result = new MiningResult();

result.setStage(stage);

result.setSubmitter(member);

result.setState(ResultState.noResult);

results.add(result);

}

}

miningResultRepository.save(results);

}

@Override

public MiningTaskStatus updateGroupStatus(String groupId, MiningTaskStatus newStatus)

{

DataMiningGroup group = fetchGroupDetails(groupId);

group.setTaskStatus(newStatus);

groupJpaRepository.save(group);

return newStatus;

}

@Override

public List<StatusObject> fetchStatusOptions()

{

return MiningTaskStatus.enum2Objects();

}

}

## 任务模块

package com.vero.dm.service.impl;

import static com.vero.dm.repository.specifications.ResultSpecifications.resultsSpec;

import static com.vero.dm.util.DownloadUtils.bigFileDownload;

import static com.vero.dm.util.DownloadUtils.generateTimestampZipFileName;

import static com.vero.dm.util.PathUtils.concat;

import static com.vero.dm.util.PathUtils.getAbsolutePath;

import static org.apache.tomcat.util.http.fileupload.FileUtils.forceDelete;

import java.io.File;

import java.io.IOException;

import java.util.\*;

import javax.servlet.http.HttpServletResponse;

import com.vero.dm.util.CompressUtil;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.data.domain.Page;

import org.springframework.data.domain.PageImpl;

import org.springframework.data.domain.Pageable;

import org.springframework.scheduling.concurrent.ThreadPoolTaskExecutor;

import org.springframework.stereotype.Service;

import org.springframework.util.StringUtils;

import org.springframework.web.multipart.MultipartFile;

import com.vero.dm.exception.error.ExceptionCode;

import com.vero.dm.exception.file.SetZipException;

import com.vero.dm.model.MiningResult;

import com.vero.dm.model.ResultRecord;

import com.vero.dm.model.Student;

import com.vero.dm.model.enums.ResultState;

import com.vero.dm.repository.dto.MiningResultDto;

import com.vero.dm.service.MiningResultService;

import com.vero.dm.service.constant.ResourcePath;

import lombok.extern.slf4j.Slf4j;

/\*\*

\* @author XiangDe Liu qq313700046@icloud.com .

\* @version 1.5 created in 22:39 2018/7/24.

\* @since data-mining-platform

\*/

@Service

@Slf4j

public class MiningResultServiceImpl extends AbstractBaseServiceImpl<MiningResult, Integer> implements MiningResultService

{

private ThreadPoolTaskExecutor threadPoolTaskExecutor;

@Autowired

public void setThreadPoolExecutor(ThreadPoolTaskExecutor threadPoolTaskExecutor)

{

this.threadPoolTaskExecutor = threadPoolTaskExecutor;

}

@Override

public MiningResult saveResult(MiningResult result)

{

return miningResultRepository.save(result);

}

@Override

public Page<MiningResultDto> findResults(String taskId, Integer stageId, Pageable pageable,

List<String> submitterIds, ResultState state,

boolean all)

{

if (all)

{

List<MiningResult> results = miningResultRepository.findAll(

resultsSpec(taskId, stageId, submitterIds, state));

return new PageImpl<>(MiningResultDto.build(results), pageable, results.size());

}

Page<MiningResult> page = miningResultRepository.findAll(

resultsSpec(taskId, stageId, submitterIds, state), pageable);

List<MiningResult> results = page.getContent();

return new PageImpl<>(MiningResultDto.build(results), pageable, page.getTotalElements());

}

@Override

public ResultRecord uploadResult(Integer resultId, MultipartFile resultFile)

{

MiningResult result = miningResultRepository.findOne(resultId);

if (Objects.isNull(result.getSubmitter()) || Objects.isNull(result.getStage())

|| Objects.isNull(result.getStage().getTask()))

{

return null;

}

Student submitter = result.getSubmitter();

String userPath = submitter.getStudentId() + '\_' + submitter.getStudentName();

String taskName = result.getStage().getTask().getTaskName();

Integer stageId = result.getStage().getOrderId();

String fileName;

if (StringUtils.isEmpty(resultFile.getOriginalFilename()))

{

fileName = resultFile.getName();

}

else

{

fileName = resultFile.getOriginalFilename();

}

String absolutePath = getAbsolutePath(

concat(ResourcePath.STUDENT\_PATH, userPath, taskName, String.valueOf(stageId)),

fileName);

log.info("[{}]开始上传文件，文件位于[{}]", submitter.getUsername(), absolutePath);

threadPoolTaskExecutor.execute(() -> {

try

{

File located = new File(Objects.requireNonNull(absolutePath));

resultFile.transferTo(located);

}

catch (IOException e)

{

e.printStackTrace();

}

});

ResultRecord resultRecord = new ResultRecord();

resultRecord.setChecked(false);

resultRecord.setFileName(fileName);

// resultRecord.setResult(result);

resultRecord.setPath(absolutePath);

resultRecord.setSize(resultFile.getSize());

resultRecord.setSubmittedDate(new Date());

resultRecordRepository.save(resultRecord);

//当前为状态为为提交的结果，置状态位为已提交

result.setState(ResultState.submitted);

Set<ResultRecord> resultRecords = result.getRecords();

if(Objects.isNull(resultRecords)){

resultRecords = new LinkedHashSet<>();

}

resultRecords.add(resultRecord);

result.setRecords(resultRecords);

miningResultRepository.save(result);

log.info(submitter.getUsername() + "上传了数据挖掘结果,位于[{}]", resultRecord.getPath());

return resultRecord;

}

@Override

public void downloadResults(List<Integer> recordIds, HttpServletResponse response)

{

List<ResultRecord> records = resultRecordRepository.findAll(recordIds);

String dir = getAbsolutePath(ResourcePath.STUDENT\_PATH);

List<String> filePaths = new ArrayList<>();

records.forEach(d -> filePaths.add(d.getPath()));

ArrayList<File> files = new ArrayList<>();

records.forEach(d -> files.add(new File(d.getPath())));

String zipPath = getAbsolutePath(concat(ResourcePath.ZIP\_PATH, ResourcePath.STUDENT\_PATH));

// 生成临时压缩文件

String zipFileName = UUID.randomUUID().toString() + ".zip";

// 输出的压缩文件路径

String zipFilePath = concat(zipPath, zipFileName);

try

{

// 进行文件压缩

// ZipUtil.zip(dir, zipPath, zipFileName, filePaths);

// ZipCompressor compressor = new ZipCompressor(zipFilePath, dir);

// compressor.zip(filePaths);

CompressUtil.zip(dir, zipFilePath,files,null, true );

bigFileDownload(response, zipFilePath, generateTimestampZipFileName("results\_"));

threadPoolTaskExecutor.execute(() -> {

try

{

// 删除临时创建的压缩文件

forceDelete(new File(zipFilePath));

log.info("删除临时压缩文件:[{}]", zipFilePath);

}

catch (IOException e)

{

throw new SetZipException("Could not delete temp zip files.",

ExceptionCode.ZipSetError);

}

});

List<MiningResult> results = miningResultRepository.findResultByRecords(recordIds);

//更新状态

results.forEach(r -> r.setState(ResultState.downloaded));

miningResultRepository.save(results);

}

catch (Exception e)

{

throw new SetZipException("Process zip operation error.", ExceptionCode.ZipSetError);

}

}

}

package com.vero.dm.service.impl;

import static com.vero.dm.repository.specifications.ResultSpecifications.resultsSpec;

import static com.vero.dm.repository.specifications.TaskSpecifications.tasksSpec;

import java.util.\*;

import org.springframework.beans.BeanUtils;

import org.springframework.data.domain.Page;

import org.springframework.data.domain.PageImpl;

import org.springframework.data.domain.Pageable;

import org.springframework.stereotype.Service;

import com.vero.dm.model.\*;

import com.vero.dm.model.enums.MiningTaskStatus;

import com.vero.dm.model.enums.ResultState;

import com.vero.dm.model.enums.StatusObject;

import com.vero.dm.model.enums.TaskProgressStatus;

import com.vero.dm.repository.dto.DataMiningGroupDto;

import com.vero.dm.repository.dto.MiningTaskDto;

import com.vero.dm.repository.dto.StudentDto;

import com.vero.dm.repository.dto.TaskStatistics;

import com.vero.dm.service.MiningTaskService;

import lombok.extern.slf4j.Slf4j;

/\*\*

\* @author XiangDe Liu qq313700046@icloud.com .

\* @version 1.5 created in

\*/

@Slf4j

@Service

public class MiningTaskServiceImpl extends AbstractBaseServiceImpl<DataMiningTask, String> implements MiningTaskService

{

@Override

public DataMiningTask findById(String id)

{

return taskJpaRepository.findOne(id);

}

@Override

public List<DataMiningTask> findAllTasks()

{

return taskJpaRepository.findAll();

}

@Override

public List<DataMiningTask> findByTaskIds(List<String> taskIds)

{

return taskJpaRepository.findAll(taskIds);

}

@Override

public List<String> findAllTaskNames()

{

return taskJpaRepository.findAllTaskNames();

}

@Override

public TaskStatistics findStatistics(String taskId)

{

DataMiningTask task = taskJpaRepository.findOne(taskId);

Set<MiningTaskStage> stages = task.getStages();

Iterator it = stages.iterator();

Map<Integer, Integer> submittedStatistics = new LinkedHashMap<>();

Map<Integer, Integer> noSubmittedStatistics = new LinkedHashMap<>();

Map<Integer, Integer> downloadedStatistics = new LinkedHashMap<>();

Map<Integer, Integer[]> stageToGroupSubmitted = new LinkedHashMap<>();

Map<ResultState, List<Integer>> stateToGroup = new LinkedHashMap<>();

for (int i = 0; i < ResultState.values().length; i++) {

stateToGroup.put(ResultState.values()[i], new ArrayList<>());

}

List<StudentDto> absentStudents;

while (it.hasNext())

{

MiningTaskStage stage = (MiningTaskStage)it.next();

List<MiningResult> submittedResults = miningResultRepository.findAll(

resultsSpec(taskId, stage.getStageId(), null, ResultState.submitted));

stateToGroup.get(ResultState.submitted).add(submittedResults.size());

submittedStatistics.put(stage.getStageId(), submittedResults.size());

List<MiningResult> noSubmittedResults = miningResultRepository.findAll(

resultsSpec(taskId, stage.getStageId(), null, ResultState.noResult));

stateToGroup.get(ResultState.noResult).add(noSubmittedResults.size());

noSubmittedStatistics.put(stage.getStageId(), noSubmittedResults.size());

List<MiningResult> downloadedResults = miningResultRepository.findAll(

resultsSpec(taskId, stage.getStageId(), null, ResultState.downloaded));

stateToGroup.get(ResultState.downloaded).add(downloadedResults.size());

downloadedStatistics.put(stage.getStageId(), downloadedResults.size());

Integer[] sizes = new Integer[]{submittedResults.size(), noSubmittedResults.size(), downloadedResults.size()};

stageToGroupSubmitted.put(stage.getStageId(), sizes);

}

List<Student> students = taskJpaRepository.findSpecializedStateStudents(taskId, ResultState.noResult);

Set<Student> uniqueStus = new HashSet<>(students);

absentStudents = StudentDto.build(new ArrayList<>(uniqueStus));

return new TaskStatistics(submittedStatistics, noSubmittedStatistics, downloadedStatistics,

absentStudents, stageToGroupSubmitted,stateToGroup);

}

@Override

public DataMiningTask saveOrUpdateMiningTask(MiningTaskDto miningTaskDto)

{

DataMiningTask task = new DataMiningTask();

BeanUtils.copyProperties(miningTaskDto, task);

List<DataMiningGroup> groups = groupJpaRepository.findAll(

miningTaskDto.getArrangeGroupIds());

List<DataSetCollection> collections = collectionJpaRepository.findAll(

miningTaskDto.getCollectionIds());

List<Algorithm> algorithms = this.algorithmJpaRepository.findAll(

miningTaskDto.getAlgorithmIds());

List<MiningGrammar> grammars = this.miningGrammarRepository.findAll(

miningTaskDto.getGrammarIds());

groups.forEach(g -> {

// 更新当前任务状态

g.setDataMiningTask(task);

g.setTaskStatus(MiningTaskStatus.newTask);

});

// 用户没有指定当前任务的执行分组,修改任务状态为“新创建”

if (groups.isEmpty())

{

task.setProgressStatus(TaskProgressStatus.newCreate);

}

else

{

task.setProgressStatus(TaskProgressStatus.assigned);

}

task.setAlgorithms(new LinkedHashSet<>(algorithms));

task.setGrammars(new LinkedHashSet<>(grammars));

task.setArrangedCollections(new LinkedHashSet<>(collections));

task.setGroups(new LinkedHashSet<>(groups));

task.setBuiltTime(new Date());

task.setPlannedStartTime(miningTaskDto.getPlannedTimeRange()[0]);

task.setPlannedFinishTime(miningTaskDto.getPlannedTimeRange()[1]);

this.taskJpaRepository.saveAndFlush(task);

this.groupJpaRepository.save(groups);

// 持久化任务阶段信息

Set<MiningTaskStage> stages = miningTaskDto.getStages();

stages.forEach(s -> {

Date[] deadline = s.getDeadline();

s.setTask(task);

s.setBegin(deadline[0]);

s.setEnd(deadline[1]);

});

this.taskStageJpaRepository.save(stages);

return task;

}

@Override

public DataMiningTask deleteByTaskId(String taskId)

{

DataMiningTask task = this.findById(taskId);

task.setAlgorithms(null);

task.setArrangedCollections(null);

task.setGroups(null);

taskJpaRepository.saveAndFlush(task);

taskJpaRepository.delete(taskId);

return task;

}

@Override

public List<DataMiningTask> deleteBatchTask(List<String> taskIds)

{

List<DataMiningTask> tasks = findByTaskIds(taskIds);

tasks.forEach(t -> {

Set<DataMiningGroup> groups = t.getGroups();

groups.forEach(g -> {

// 修改当前任务状态

g.setDataMiningTask(null);

g.setTaskStatus(MiningTaskStatus.toBeAssigned);

});

groupJpaRepository.save(groups);

t.setAlgorithms(null);

t.setArrangedCollections(null);

t.setGroups(null);

});

taskJpaRepository.delete(tasks);

return tasks;

}

@Override

public Page<MiningTaskDto> fetchTaskList(boolean fetch, String taskName, Date plannedBeginDate,

Date plannedEndDate, Date builtTimeBegin,

Date builtTimeEnd, String studentId,

Pageable pageable, TaskProgressStatus progressStatus,

Integer lowBound, Integer upperBound)

{

List<DataMiningTask> finalResult = new LinkedList<>();

List<DataMiningTask> taskLimit = taskJpaRepository.findByLinkedGroupsBound(lowBound,

upperBound);

if (fetch)

{

List<DataMiningTask> tasks = taskJpaRepository.findAll(

tasksSpec(taskName, plannedBeginDate, plannedEndDate, builtTimeBegin, builtTimeEnd,

progressStatus, studentId));

taskLimit.forEach(c -> {

if (tasks.contains(c))

{

finalResult.add(c);

}

});

}

else

{

Page<DataMiningTask> tasks = taskJpaRepository.findAll(

tasksSpec(taskName, plannedBeginDate, plannedEndDate, builtTimeBegin, builtTimeEnd,

progressStatus, studentId),

pageable);

List<DataMiningTask> content = tasks.getContent();

content.forEach(c -> {

if (taskLimit.contains(c))

{

finalResult.add(c);

}

});

}

return new PageImpl<>(MiningTaskDto.build(finalResult));

}

@Override

public Map<String, List<DataMiningGroupDto>> fetchInvolvedGroups(List<String> taskIds)

{

// return miningTaskDao.fetchInvolvedGroups(taskId);

Map<String, List<DataMiningGroupDto>> groups = new HashMap<>();

List<DataMiningTask> tasks = taskJpaRepository.findAll(taskIds);

tasks.forEach(t -> {

List<DataMiningGroup> miningGroups = groupJpaRepository.findByDataMiningTaskId(

t.getTaskId());

miningGroups.forEach(m -> m.getGroupMembers().size());

groups.put(t.getTaskId(), DataMiningGroupDto.build(miningGroups));

});

return groups;

}

@Override

public List<DataMiningGroup> removeInvolvedGroups(String taskId, List<String> groupIds)

{

List<DataMiningGroup> groups = groupJpaRepository.findAll(groupIds);

groups.forEach(g -> doArrangeTask(null, g));

groupJpaRepository.save(groups);

return groups;

}

@Override

public List<MiningTaskStage> fetchRefStages(String taskId)

{

return new LinkedList<>(taskJpaRepository.findOne(taskId).getStages());

}

@Override

public DataMiningGroup involveGroup(String taskId, String groupId)

{

DataMiningGroup group = groupJpaRepository.findOne(groupId);

DataMiningTask miningTask = taskJpaRepository.findOne(taskId);

doArrangeTask(miningTask, group);

groupJpaRepository.saveAndFlush(group);

return group;

}

@Override

public List<DataMiningGroup> involveGroups(String taskId, List<String> groupIds)

{

List<DataMiningGroup> groups = groupJpaRepository.findAll(groupIds);

DataMiningTask miningTask = taskJpaRepository.findOne(taskId);

groups.forEach(g -> doArrangeTask(miningTask, g));

groupJpaRepository.save(groups);

return groups;

}

private void doArrangeTask(DataMiningTask miningTask, DataMiningGroup g)

{

if (g.getDataMiningTask() == null)

{

g.setDataMiningTask(miningTask);

}

else

{

log.error(

"Group [{}] has arranged a task [].Please revoke their relationship firstly.",

g.getGroupName(), g.getDataMiningTask().getTaskName());

}

}

@Override

public List<Algorithm> fetchConfiguredAlgorithms(String taskId)

{

return new LinkedList<>(findById(taskId).getAlgorithms());

}

@Override

public List<Algorithm> configureAlgorithms(String taskId, List<Integer> algorithmIds)

{

DataMiningTask task = findById(taskId);

List<Algorithm> algorithms = algorithmJpaRepository.findAll(algorithmIds);

if (algorithmIds == null || algorithmIds.isEmpty())

{

task.setAlgorithms(new LinkedHashSet<>());

taskJpaRepository.save(task);

return new LinkedList<>();

}

task.setAlgorithms(new LinkedHashSet<>(algorithms));

taskJpaRepository.save(task);

return algorithms;

}

@Override

public List<DataSetCollection> configureMiningSets(String taskId, List<Integer> collectionIds)

{

List<DataSetCollection> collections = collectionJpaRepository.findAll(collectionIds);

DataMiningTask task = findById(taskId);

task.setArrangedCollections(new LinkedHashSet<>(collections));

taskJpaRepository.save(task);

return collections;

}

@Override

public List<DataSetCollection> fetchRefCollections(String taskId)

{

return new LinkedList<>(findById(taskId).getArrangedCollections());

}

@Override

public List<DataSetCollection> removeAllMiningSets(String taskId)

{

DataMiningTask task = findById(taskId);

List<DataSetCollection> collections = fetchRefCollections(taskId);

task.setArrangedCollections(null);

taskJpaRepository.save(task);

return collections;

}

@Override

public List<StatusObject> fetchProgressStatus()

{

return TaskProgressStatus.enum2Objects();

}

@Override

public Integer[] minAndMaxGroupNum()

{

Integer[] max\_min\_pair = new Integer[2];

if (findAllTasks().isEmpty())

{

max\_min\_pair[0] = 0;

max\_min\_pair[1] = 0;

return max\_min\_pair;

}

max\_min\_pair[0] = taskJpaRepository.findMaxGroupNum();

max\_min\_pair[1] = taskJpaRepository.findMinGroupNum();

return max\_min\_pair;

}

}