

# day03-学习计划和进度

经过前面的努力，我们已经完成了《我的课程表》相关的功能的基础部分，不过还有功能实现的并不完善。还记得昨天给大家的练习题吗？《查询我正在学习的课程》，在原型图中有这样的一个需求：



我们需要在查询结果中返回已学习课时数、正在学习的章节名称。虽然我们在learning\_lesson表中设计了两个字段：

- learned\_sections：已学习章节数
- latest\_learn\_time：最近学习时间

但是，这几个字段默认都是空或0，我们该如何得知用户到底学习了几节？最近一次学习是什么时候？最近一次学习的是第几章节呢？

以上的问题归纳下来，就是一个**学习进度统计**问题，这在在线教育、视频播放领域是一个非常常见的问题。因此，学会了解决这套解决方案，你就能游刃有余的应对相关行业的类似问题了。

大家在学习这套解决方案的同时，也可以增强下面的能力：

- 需求分析和表设计能力
- 复杂SQL的编写能力
- 处理高并发写数据库的能力

## 1.分析产品原型

大部分人的学习自律性是比较差的，属于“买了就算会了”的状态。如果学员学习积极性下降，学习结果也会不尽人意，从而产生挫败感。导致购买课程的欲望也会随之下降，形成恶性循环，不利于我们卖课。

所以，我们推出学习计划的功能，让学员制定一套学习计划，每周要学几节课。系统会做数据统计，每一周计划是否达标，达标后给予奖励，未达标则提醒用户，达到督促用户持续学习的目的。

用户学习效果好了，产生了好的结果，就会有继续学习、购买课程的欲望，形成良性循环。因此，学习计划、学习进度统计其实是学习辅助中必不可少的环节。

## 1.1.分析业务流程

我们从两个业务点来分析：

- 学习计划
- 学习进度统计

### 1.1.1.学习计划

在我的课程页面，可以对有效的课程添加学习计划：



学习计划就是简单设置一下用户每周计划学习几节课：

创建计划

选择课程

请选择

每周学习章节数

预计学习完成时间

预计 2022年7月10日 完成

取消

创建

这个在昨天的数据库设计中已经有对应的字段了，只不过功能尚未完成。

有了计划以后，我们就可以在我的课程页面展示用户计划的完成情况，提醒用户尽快学习：

本周实际学习小节数量

本周计划学习小节量

本周学习积分

本周计划

1/15

积分奖励

25

课程	本周进度	课程进度	学习时间	操作
课程一MySQL入门	0/5	12/52	2022-7-10	去学习
课程二SQLsevers入门	1/5	12/52	2022-7-10	去学习
课程三Oracle入门	0/5	12/52	2022-7-10	去学习

某课程本周已经学习小节数量

某课程本周计划学习小节数量

黑马程序员-研究院

可以看到，在学习计划中是需要统计用户“已经学习的课时数量”的。那么我们该如何统计用户学了多少课时呢？

### 1.1.2.学习进度统计

要统计学习进度，需要先弄清楚用户学习的方式，学习的内容。在原型图《课程学习页-录播课-课程学习页-目录》中，可以看到学习课程的原型图：



一个课程往往包含很多个章（chapter），每一章下又包含了很多小节（section）。章本身没有课程内容，只是划分课程的一个概念，因此统计学习进度就是看用户学了多少个小节。

小节也分两种，一种是视频；一种是每章最后的阶段考试。用户学完一个视频，或者参加了最终的考试都算学完了一个小节。

考试只要提交了就算学完了，比较容易判断是否学完。但是视频该如何统计呢？达到什么样的标准才算这一小节的视频学完了呢？

这里我们不能要求用户一定要播放进度到100%，太苛刻了。所以，天机学堂的产品是这样设计的：

## 页面开发规则

### 页面逻辑

1. 学习区
  - 1) 顶部为课程小节名
  - 2) 下面为视频播放区，用户可以播放、暂停、调整音量、视频全屏，问题反馈
  - 3) 上下箭头为切换课程章节，没有上下章时 箭头变为灰色不可点击
2. 侧栏
  - 1) 课程信息模块，展示课程名称、主讲人名称、课程图片
  - 2) 点击双箭头收起侧栏
  - 3) 点击课程名称返回至课程详情页
3. tab切换栏
  - 1) 目录-展示课程各章节，小节前有学习状态，**学习进度超过50%则显示已学习状态。**
  - 2) 章节后面的考试需要完成本章所有小节课程后才可以进行。

黑马程序员-研究院

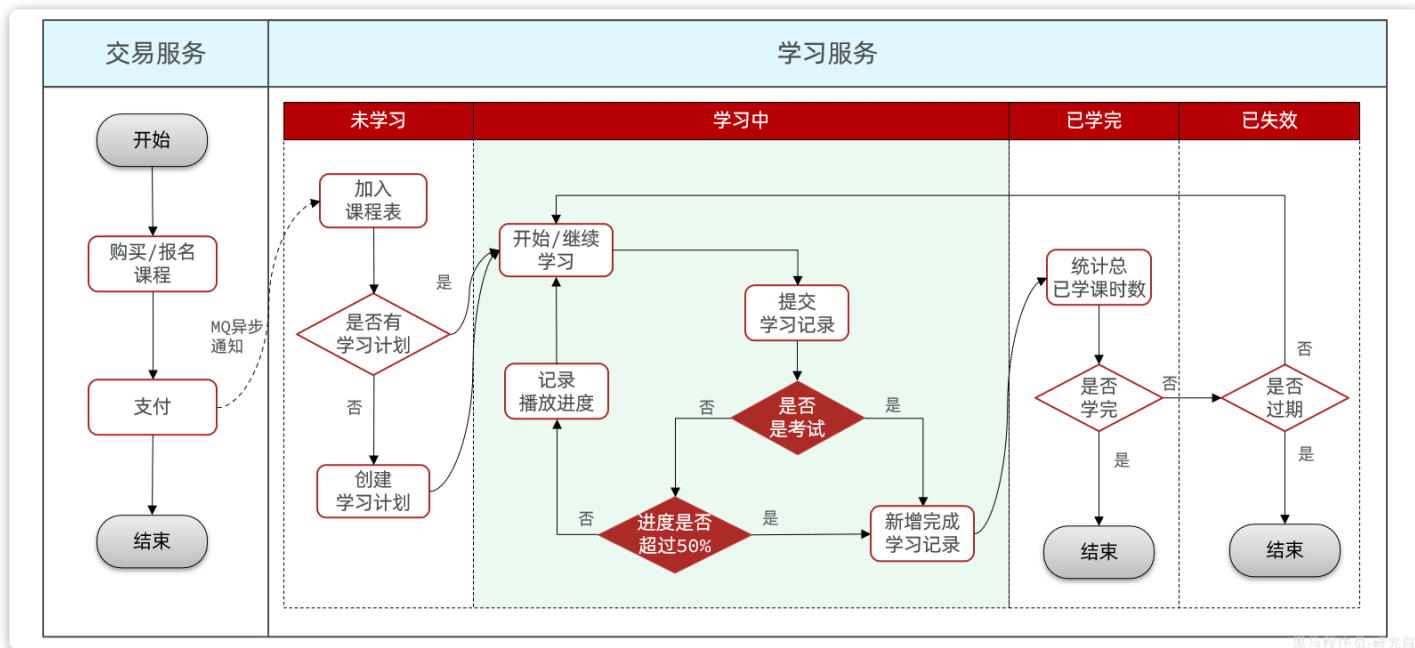
因此，只要视频播放进度达到**50%**就算是完成本节学习了。所以用户在播放视频的过程中，需要不断提交视频的播放进度，当我们发现视频进度超过50%时就可以标记这一小节为**已学完**。

当然，我们不能仅仅记录视频是否学完，还应该记录用户具体播放的进度到了**第几秒**。只有这样在用户关闭视频，再次播放时我们才能实现视频自动续播功能，用户体验会比较好。

也就是说，要记录用户学习进度，需要记录下列核心信息：

- 小节的基础信息（id、关联的课程id等）
- 当前的播放进度（第几秒）
- 当前小节是否已学完（播放进度是否超50%）

用户每学习一个小节，就会新增一条学习记录，当该课程的全部小节学习完毕，则该课程就从**学习中**进入**已学完**状态了。整体流程如图：



## 1.2.业务接口统计

接下来我们分析一下这部分功能相关的接口有哪些，按照用户的学习顺序，依次有下面几个接口：

- 创建学习计划
- 查询学习记录
- 提交学习记录
- 查询我的计划

### 1.2.1.创建学习计划

在个人中心的我的课表列表中，没有学习计划的课程都会有一个**创建学习计划**的按钮，在原型图就能看到：

1 按学习时间进行排序 按购买时间进行排序

创建学习计划

## 2 iOS应用开发入门

学习周期: 2022.01.05-2022.07.07

已学习 5/52 课时

继续学习

创建学习计划

## iOS应用开发入门

开始学习 使用对象

学习周期: 2022.01.05-2022.07.07

已学完

重新学习

创建学习计划，本质就是让用户设定自己每周的学习频率：

### 修改计划

选择课程

Java课程

每周学习  
章节数

5

预计学习  
完成时间

预计 2022年7月10日 完成

取消

修改

黑马程序员-研究院

而学习频率我们在设计learning\_lesson表的时候已经有两个字段来表示了：

```
1 CREATE TABLE `learning_lesson` (
2   `id` bigint NOT NULL COMMENT '主键',
3   `user_id` bigint NOT NULL COMMENT '学员id',
4   `course_id` bigint NOT NULL COMMENT '课程id',
5   `status` tinyint NULL DEFAULT 0 COMMENT '课程状态, 0-未学习, 1-学习中, 2-已学完, 3-已失效',
6   `week_freq` tinyint NULL DEFAULT NULL COMMENT '每周学习频率, 每周3天, 每天2节, 则频率为6',
```

```

7  `plan_status` tinyint NOT NULL DEFAULT 0 COMMENT '学习计划状态，0-没有计划，1-计划进行中',
8  `learned_sections` int NOT NULL DEFAULT 0 COMMENT '已学习小节数量',
9  `latest_section_id` bigint NULL DEFAULT NULL COMMENT '最近一次学习的小节id',
10 `latest_learn_time` datetime NULL DEFAULT NULL COMMENT '最近一次学习的时间',
11 `create_time` datetime NOT NULL DEFAULT CURRENT_TIMESTAMP COMMENT '创建时间',
12 `expire_time` datetime NOT NULL COMMENT '过期时间',
13 `update_time` datetime NOT NULL DEFAULT CURRENT_TIMESTAMP ON UPDATE
    CURRENT_TIMESTAMP COMMENT '更新时间',
14 PRIMARY KEY (`id`) USING BTREE,
15 UNIQUE INDEX `idx_user_id`(`user_id`, `course_id`) USING BTREE
16 ) ENGINE = InnoDB CHARACTER SET = utf8mb4 COLLATE = utf8mb4_0900_ai_ci COMMENT
    = '学生课程表' ROW_FORMAT = Dynamic;
17

```

当我们创建学习计划时，就是更新 `learning_lesson` 表，写入 `week_freq` 并更新 `plan_status` 为计划进行中即可。因此请求参数就是课程的id、每周学习频率。

再按照Restful风格，最终接口如下：

参数	说明		
请求方式	POST		
请求路径	/lessons/plans		
请求参数	参数名	类型	说明
	courseId	Long	课程id
	weekFreq	Integer	计划每周学习频率
返回值	无		

### 1.2.2.查询学习记录

用户创建完计划自然要开始学习课程，在用户学习视频的页面，首先要展示课程的一些基础信息。例如课程信息、章节目录以及每个小节的学习进度：





其中，课程、章节、目录信息等数据都在课程微服务，而学习进度肯定是在学习微服务。**课程信息是必备的，而学习进度却不一定存在。**

因此，查询这个接口的请求肯定是请求到课程微服务，查询课程、章节信息，再由课程微服务向学习微服务查询学习进度，合并后一起返回给前端即可。

所以，学习中心要提供一个查询章节学习进度的Feign接口，事实上这个接口已经在tj-api模块的LearningClient中定义好了：

```
1
2 /**
3  * 查询当前用户指定课程的学习进度
4  * @param courseId 课程id
5  * @return 课表信息、学习记录及进度信息
6  */
7 @GetMapping("/learning-records/course/{courseId}")
8 LearningLessonDTO queryLearningRecordByCourse(@PathVariable("courseId") Long
   courseId);
```

对应的DTO也都在tj-api模块定义好了，因此整个接口规范如下：

参数	说明
请求方式	GET
请求路径	/learning-records/course/{courseId}
请求参数	路径占位符参数，courseId：课表关联的课程id

返回值	参数名	类型	说明		
	id	Long	课表id		
	latestSection id	Long	最近学习的小节id		
	records	array	参数名	类型	说明
			sectionId	Long	小节id
			moment	int	视频播放进度，第几秒
			finished	boolean	是否学完

### 1.2.3.提交学习记录

之前分析业务流程的时候已经聊过，学习记录就是用户当前学了哪些小节，以及学习到该小节的进度如何。而小节类型分为考试、视频两种。

- 考试比较简单，只要提交了就说明这一节学完了。
- 视频比较麻烦，需要记录用户的播放进度，进度超过50%才算学完。因此视频播放的过程中需要不断提交播放进度到服务端，而服务端则需要保存学习记录到数据库。

只要记录了用户学过的每一个小节，以及小节对应的学习进度、是否学完。无论是**视频续播**、还是**统计学习计划进度**，都可以轻松实现了。

因此，提交学习记录就是提交小节的信息和小节的学习进度信息。考试提交一次即可，视频则是播放中频繁提交。提交的信息包括两大部分：

- 小节的基本信息
  - 小节id
  - lessonId
  - 小节类型：可能是视频，也可能是考试。考试无需提供播放进度信息
  - 提交时间
- 播放进度信息
  - 视频时长：时长结合播放进度可以判断有没有超过50%
  - 视频播放进度：也就是第几秒

综上，提交学习记录的接口信息如下：

参数	说明		
请求方式	POST		
请求路径	/learning-records		
请求参数	参数名	类型	说明
	lessonId	long	课表id
	sectionId	long	小节id
	sectionType	int	小节类型：1-视频，2-考试
	commitTime	LocalDateTime	提交时间
	duration	int	视频总时长，单位秒
	moment	int	视频播放进度，单位秒
返回值	无		
接口描述	<ul style="list-style-type: none"> <li>视频播放：当播放进度超过50%则判定为本节学完</li> <li>考试：考试结束时提交记录，直接判定为本节学完</li> </ul>		

### 1.2.4.查询我的学习计划

在个人中心的我的课程页面，会展示用户的学习计划及**本周**的学习进度，原型如图：



需要注意的是这个查询其实是一个分页查询，因为页面最多展示10行，而学员同时在学的课程可能会超过10个，这个时候就会分页展示，当然这个分页可能是滚动分页，所以没有进度条。另外，查询的是**我的**学习计划，隐含的查询条件就是当前登录用户，这个无需传递，通过请求头即可获得。

因此查询参数只需要**分页**参数即可。

查询结果中有很多对于已经学习的小节数量的统计，因此将来我们一定要保存用户对于每一个课程的学习记录，哪些小节已经学习了，哪些已经学完了。只有这样才能统计出学习进度。

查询的结果如页面所示，分上下两部分。：

总的统计信息：

- 本周已完成总章节数：需要对学习记录做统计
- 课程总计划学习数量：累加课程的总计划学习频率即可
- 本周学习积分：积分暂不实现

正在学习的N个课程信息的集合，其中每个课程包含下列字段：

- 该课程本周学了几节：统计学习记录
- 计划学习频率：在learning\_lesson表中有对应字段
- 该课程总共学了几节：在learning\_lesson表中有对应字段
- 课程总章节数：查询课程微服务
- 该课程最近一次学习时间：在learning\_lesson表中有对应字段

综上，查询学习计划进度的接口信息如下：

参数	说明			
请求方式	GET			
请求路径	/lessons/plans			
请求参数	分页参数：PageQuery			
返回值	参数名	类型	说明	
	weekPoints	int	本周学习积分	
	weekFinished	int	本周已学完小节数量	
	weekTotalPlan	int	本周计划学习小节数量	

	list	Array	参数	类型	说明
			courseId	Long	课程id
			courseName	String	课程名称
			weekLearnedSections	int	本周学习的小节数量
			weekFreq	int	本周计划学习数量
			learnedSections	int	总已学习小节数量
			sections	int	总小节数量
			latestLearnTime	LocalDateTime	最近一次学习时间

### 1.3.设计数据库

数据表的设计要满足学习计划、学习进度的功能需求。学习计划信息在 `learning_lesson` 表中已经设计，因此我们关键是设计学习进度记录表即可。

按照之前的分析，用户学习的课程包含多个小节，小节的类型包含两种：

- 视频：视频播放进度超过50%就算当节学完
- 考试：考完就算一节学完

学习进度除了要记录哪些小节学完，还要记录学过的小节、每小节的播放的进度（方便续播）。因此，需要记录的数据就包含以下部分：

- 学过的小节的基础信息
  - 小节id
  - 小节对应的lessonId
  - 用户id：学习课程的人
- 小节的播放进度信息
  - 视频播放进度：也就是播放到了第几秒
  - 是否已经学完：播放进度有没有超过50%




- 第一次学完的时间：用户可能重复学习，第一次从未学到学完的时间要记录下来

再加上一些表基础字段，整张表结构就出来了：

```
1 CREATE TABLE IF NOT EXISTS `learning_record` (  
2   `id` bigint NOT NULL COMMENT '学习记录的id',  
3   `lesson_id` bigint NOT NULL COMMENT '对应课表的id',  
4   `section_id` bigint NOT NULL COMMENT '对应小节的id',  
5   `user_id` bigint NOT NULL COMMENT '用户id',  
6   `moment` int DEFAULT '0' COMMENT '视频的当前观看时间点，单位秒',  
7   `finished` bit(1) NOT NULL DEFAULT b'0' COMMENT '是否完成学习，默认false',  
8   `create_time` datetime NOT NULL DEFAULT CURRENT_TIMESTAMP COMMENT '第一次观看时  
   间',  
9   `update_time` datetime NOT NULL DEFAULT CURRENT_TIMESTAMP ON UPDATE  
   CURRENT_TIMESTAMP COMMENT '更新时间（最近一次观看时间）',  
10  PRIMARY KEY (`id`) USING BTREE,  
11  KEY `idx_update_time` (`update_time`) USING BTREE,  
12  KEY `idx_user_id` (`user_id`) USING BTREE,  
13  KEY `idx_lesson_id` (`lesson_id`,`section_id`) USING BTREE  
14 ) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_0900_ai_ci  
   ROW_FORMAT=DYNAMIC COMMENT='学习记录表';  
15
```

课前资料也提供了对应的SQL语句：

新加卷 (D:) > 课程资料 > 天机学堂 > 课件 > day03-学习计划和进度 > 资料 >

名称	类型	大小
 learning_record.sql	SQL 源文件	5 KB
 LearningPlanDTO.java	Java 源文件	1 KB
 LearningPlanPageVO.java	Java 源文件	2 KB

黑马程序员-研究院

## 1.4.生成基础代码

接下来我们就可以生成数据库实体对应的基础代码了。

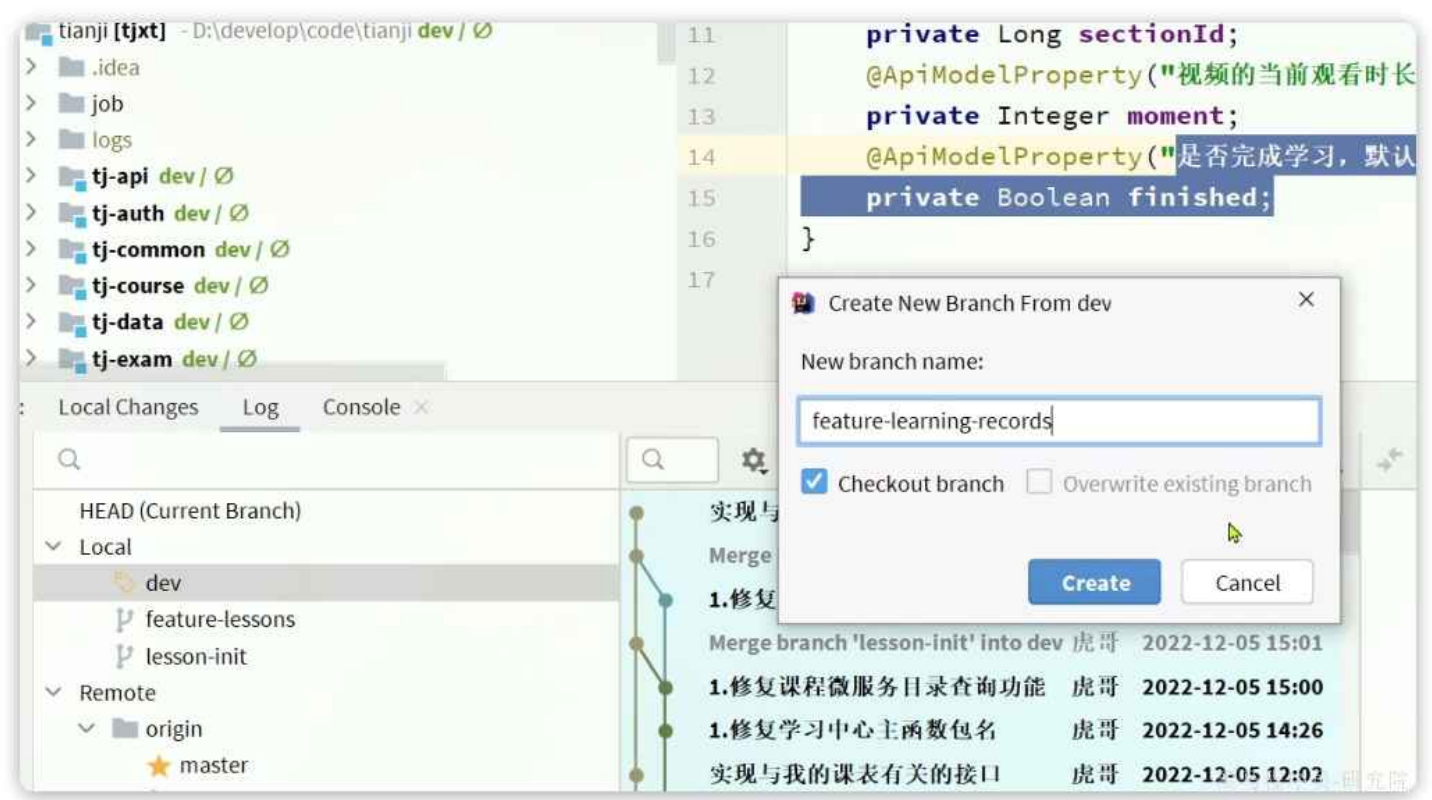
### 1.4.1.创建新分支

动手之前，不要忘了开发新功能需要创建新的分支。这里我们依然在 `DEV` 分支基础上，创建一个新的 `feature` 类型分支：`feature-learning-records`

我们可以选择用命令：

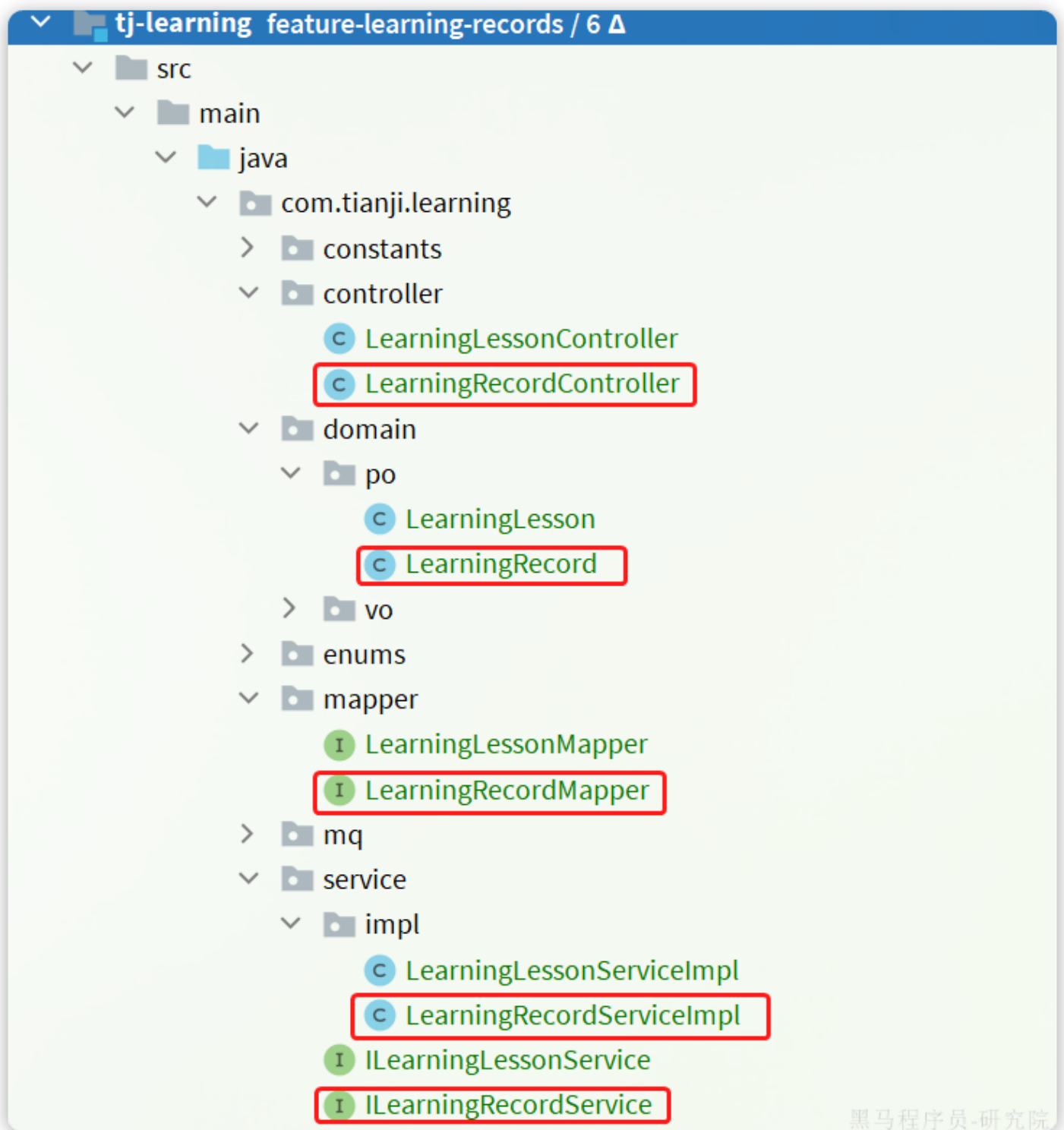
```
1 git checkout -b feature-learning-records
```

也可以选择图形界面方式：



### 1.4.2.代码生成

同样是使用MybatisPlus插件，这里不再赘述。效果如下：



需要注意的是，我们同样需要把生成的实体类的ID策略改成雪花算法：



LearningRecord.java ×

```
@TableName("learning_record")
public class LearningRecord implements Serializable {

    private static final long serialVersionUID = 1L;

    | 学习记录的id
    @TableId(value = "id", type = IdType.ASSIGN_ID)
    private Long id;
```

另外，按照Restful风格，把controller的路径做修改：

```
@RestController
@RequestMapping("/learning-records")
public class LearningRecordController {

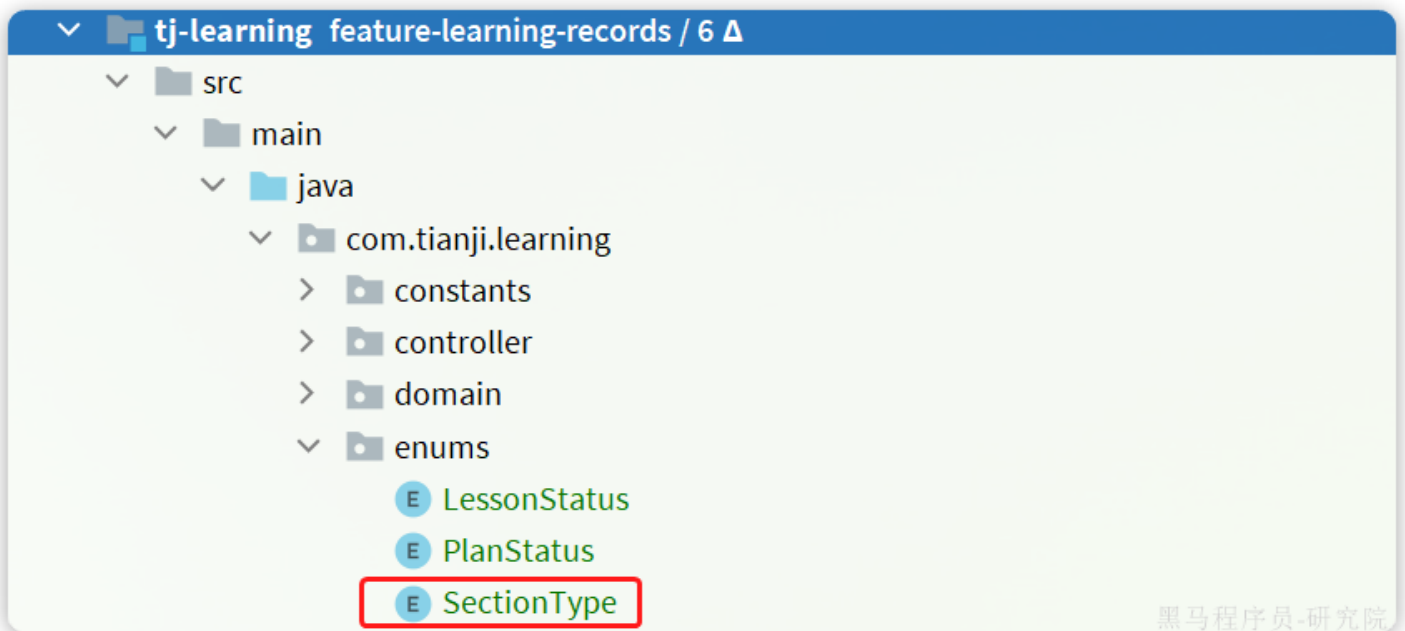
}
```

### 1.4.3.类型枚举

在昨天学习的课表中，有一种状态枚举，就是把课程的状态通过枚举定义出来，避免出现错误。而在学习记录中，有一个section\_type字段，代表记录的小节有两种类型：

- 1，视频类型
- 2，考试类型

为了方便我们也定义为枚举，称为类型枚举：



具体代码：

```
1 package com.tianji.learning.enums;
2
3 import com.baomidou.mybatisplus.annotation.EnumValue;
4 import com.fasterxml.jackson.annotation.JsonCreator;
5 import com.fasterxml.jackson.annotation.JsonValue;
6 import com.tianji.common.enums.BaseEnum;
7 import lombok.Getter;
8
9 @Getter
10 public enum SectionType implements BaseEnum {
11     VIDEO(1, "视频"),
12     EXAM(2, "考试"),
13     ;
14     @JsonValue
15     @EnumValue
16     int value;
17     String desc;
18
19     SectionType(int value, String desc) {
20         this.value = value;
21         this.desc = desc;
22     }
23
24
25     @JsonCreator(mode = JsonCreator.Mode.DELEGATING)
26     public static SectionType of(Integer value){
27         if (value == null) {
28             return null;
```

```
29     }
30     for (SectionType status : values()) {
31         if (status.equalsValue(value)) {
32             return status;
33         }
34     }
35     return null;
36 }
37 }
```

## 2.实现接口

### 2.1.查询学习记录

首先回顾一下接口基本信息：

参数	说明				
请求方式	GET				
请求路径	/learning-records/course/{courseId}				
请求参数	路径占位符参数，courseId：课表关联的课程id				
返回值	参数名	类型	说明		
	id	Long	课表id		
	latestSection id	Long	最近学习的小节id		
	records	array	参数名	类型	说明
			sectionId	Long	小节id
			moment	int	视频播放进度，第几秒
			finished	boolean	是否学完

#### 2.1.1.思路分析

做个接口是给课程微服务调用的，因此在tj-api模块的LearningClient中定义好了：

```

1
2 /**
3  * 查询当前用户指定课程的学习进度
4  * @param courseId 课程id
5  * @return 课表信息、学习记录及进度信息
6  */
7 @GetMapping("/learning-records/course/{courseId}")
8 LearningLessonDTO queryLearningRecordByCourse(@PathVariable("courseId") Long
courseId);

```

对应的DTO也都在tj-api模块定义好了。我们直接实现接口即可。

由于请求参数是 `courseId`，而返回值中包含 `lessonId` 和 `latestSectionid` 都在 `learning_lesson` 表中，因此我们需要根据courseId和userId查询出lesson信息。然后再根据lessonId查询学习记录。整体流程如下：

- 获取当前登录用户id
- 根据courseId和userId查询LearningLesson
- 判断是否存在或者是否过期
  - 如果不存在或过期直接返回空
  - 如果存在并且未过期，则继续
- 查询lesson对应的所有学习记录

## 2.1.2.代码实现

首先在 `tj-learning` 模块下的

`com.tianji.learning.controller.LearningRecordController` 下定义接口：

```

1 package com.tianji.learning.controller;
2
3
4 import com.tianji.api.dto.learning.LearningLessonDTO;
5 import com.tianji.learning.service.ILearningRecordService;
6 import io.swagger.annotations.Api;
7 import io.swagger.annotations.ApiOperation;
8 import io.swagger.annotations.ApiParam;
9 import lombok.RequiredArgsConstructor;
10 import org.springframework.web.bind.annotation.*;
11
12 /**
13  * <p>
14  * 学习记录表 前端控制器

```

```

15  * </p>
16  */
17  @RestController
18  @RequestMapping("/learning-records")
19  @Api(tags = "学习记录的相关接口")
20  @RequiredArgsConstructor
21  public class LearningRecordController {
22
23      private final ILearningRecordService recordService;
24
25      @ApiOperation("查询指定课程的学习记录")
26      @GetMapping("/course/{courseId}")
27      public LearningLessonDTO queryLearningRecordByCourse(
28          @ApiParam(value = "课程id", example = "2") @PathVariable("courseId")
29          Long courseId){
30          return recordService.queryLearningRecordByCourse(courseId);
31      }
32  }

```

然后在 `com.tianji.learning.service.ILearningRecordService` 中定义方法：

```

1  package com.tianji.learning.service;
2
3  import com.baomidou.mybatisplus.extension.service.IService;
4  import com.tianji.api.dto.learning.LearningLessonDTO;
5  import com.tianji.learning.domain.po.LearningRecord;
6
7  /**
8   * <p>
9   * 学习记录表 服务类
10  * </p>
11  */
12  public interface ILearningRecordService extends IService<LearningRecord> {
13
14      LearningLessonDTO queryLearningRecordByCourse(Long courseId);
15  }

```

最后在 `com.tianji.learning.service.impl.LearningRecordServiceImpl` 中定义实现类：

```

1  package com.tianji.learning.service.impl;
2
3  import com.baomidou.mybatisplus.extension.service.impl.ServiceImpl;

```

```
4 import com.tianji.api.dto.leanring.LearningLessonDTO;
5 import com.tianji.api.dto.leanring.LearningRecordDTO;
6 import com.tianji.common.utils.BeanUtils;
7 import com.tianji.common.utils.UserContext;
8 import com.tianji.learning.domain.po.LearningLesson;
9 import com.tianji.learning.domain.po.LearningRecord;
10 import com.tianji.learning.mapper.LearningRecordMapper;
11 import com.tianji.learning.service.ILearningLessonService;
12 import com.tianji.learning.service.ILearningRecordService;
13 import lombok.RequiredArgsConstructor;
14 import org.springframework.stereotype.Service;
15
16 import java.util.List;
17
18 /**
19  * <p>
20  * 学习记录表 服务实现类
21  * </p>
22  *
23  * @author 虎哥
24  * @since 2022-12-10
25  */
26 @Service
27 @RequiredArgsConstructor
28 public class LearningRecordServiceImpl extends
    ServiceImpl<LearningRecordMapper, LearningRecord> implements
    ILearningRecordService {
29
30     private final ILearningLessonService lessonService;
31
32     @Override
33     public LearningLessonDTO queryLearningRecordByCourse(Long courseId) {
34         // 1. 获取登录用户
35         Long userId = UserContext.getUser();
36         // 2. 查询课表
37         LearningLesson lesson = lessonService.queryByUserAndCourseId(userId,
courseId);
38         // 3. 查询学习记录
39         // select * from xx where lesson_id = #{lessonId}
40         List<LearningRecord> records = lambdaQuery()
41             .eq(LearningRecord::getLessonId,
lesson.getId()).list();
42         // 4. 封装结果
43         LearningLessonDTO dto = new LearningLessonDTO();
44         dto.setId(lesson.getId());
45         dto.setLatestSectionId(lesson.getLatestSectionId());
46         dto.setRecords(BeanUtils.copyList(records, LearningRecordDTO.class));
```

```
47         return dto;
48     }
49 }
```

其中查询课表的时候，需要调用 `ILessonService` 中的 `queryByUserAndCourseId()` 方法，该方法代码如下：

```
1 @Override
2 public LearningLesson queryByUserAndCourseId(Long userId, Long courseId) {
3     return getOne(buildUserIdAndCourseIdWrapper(userId, courseId));
4 }
5
6 private LambdaQueryWrapper<LearningLesson> buildUserIdAndCourseIdWrapper(Long
    userId, Long courseId) {
7     LambdaQueryWrapper<LearningLesson> queryWrapper = new
    QueryWrapper<LearningLesson>()
8         .lambda()
9         .eq(LearningLesson::getUserId, userId)
10        .eq(LearningLesson::getCourseId, courseId);
11     return queryWrapper;
12 }
```

## 2.2.提交学习记录

回顾一下接口信息：

参数	说明		
请求方式	POST		
请求路径	/learning-records		
请求参数	参数名	类型	说明
	lessonId	long	课表id
	sectionId	long	小节id
	sectionType	int	

			小节类型：1-视频，2-考试
	commitTime	LocalDateTime	提交时间
	duration	int	视频总时长，单位秒
	moment	int	视频播放进度，单位秒
返回值	无		
接口描述	<ul style="list-style-type: none"> <li>• 视频播放：当播放进度超过50%则判定为本节学完</li> <li>• 考试：考试结束时提交记录，直接判定为本节学完</li> </ul>		

### 2.2.1.思路分析

学习记录就是用户当前学了哪些小节，以及学习到该小节的进度如何。而小节类型分为考试、视频两种。

- 考试比较简单，只要提交了就说明这一节学完了。
- 视频比较麻烦，需要记录用户的播放进度，进度超过50%才算学完。因此视频播放的过程中需要不断提交播放进度到服务端，而服务端则需要保存学习记录到数据库。

以上信息都需要保存到learning\_record表中。

特别需要注意的是，学习记录learning\_record表记录的是每一个小节的学习进度。而在learning\_lesson表也需要记录一些学习进度相关字段：

```
CREATE TABLE `learning_lesson` (
  `id` bigint NOT NULL COMMENT '主键',
  `user_id` bigint NOT NULL COMMENT '学员id',
  `course_id` bigint NOT NULL COMMENT '课程id',
  `status` tinyint NULL DEFAULT 0 COMMENT '课程状态, 0-未学习, 1-学习中, 2-已学完, 3-已失效',
  `week_freq` tinyint NULL DEFAULT NULL COMMENT '每周学习频率, 每周3天, 每天2节, 则频率为6',
  `plan_status` tinyint NOT NULL DEFAULT 0 COMMENT '学习计划状态, 0-没有计划, 1-计划进行中',
  `learned_sections` int NOT NULL DEFAULT 0 COMMENT '已学习小节数量',
  `latest_section_id` bigint NULL DEFAULT NULL COMMENT '最近一次学习的小节id',
  `latest_learn_time` datetime NULL DEFAULT NULL COMMENT '最近一次学习的时间',
  `create_time` datetime NOT NULL DEFAULT CURRENT_TIMESTAMP COMMENT '创建时间',
  `expire_time` datetime NOT NULL COMMENT '过期时间',
  `update_time` datetime NOT NULL DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP COMMENT '更新时间'
```

这些字段是整个课程的进度统计：

- learned\_sections：已学习小节数量
- latest\_section\_id：最近一次学习的小节id
- latest\_learn\_time：最近一次学习时间



💡 每当有一个小节被学习，都应该更新 `latest_section_id` 和 `latest_learn_time`；每当有一个小节学习完后，`learned_sections` 都应该累加1。不过这里有一点容易出错的地方：

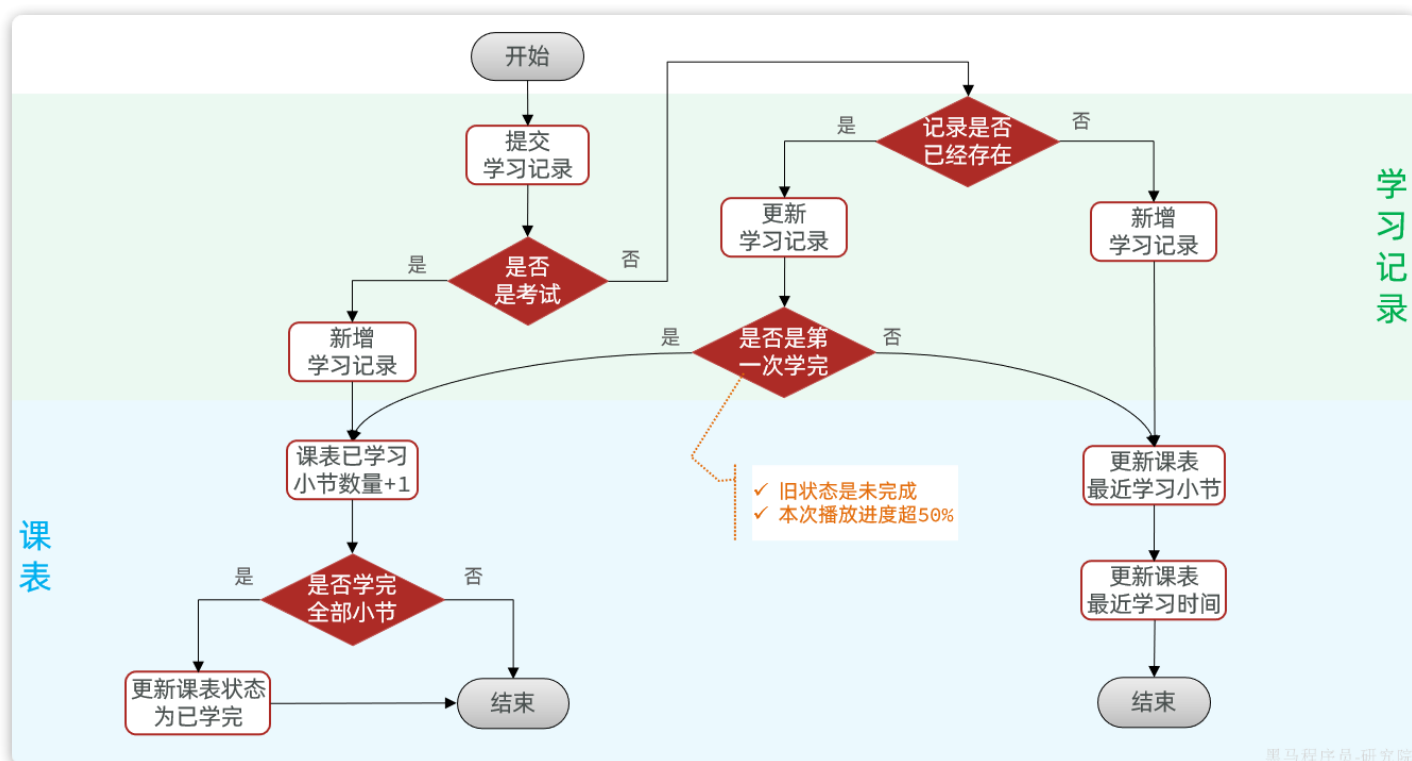
- 考试只会被参加一次，考试提交则小节学完，`learned_sections` 累加1
- 视频可以被重复播放，只有在第一次学完一个视频时，`learned_sections` 才需要累加1

那么问题来了，如何判断视频是否是第一次学完？我认为应该同时满足两个条件：

- 视频播放进度超过50%
- 之前学习记录的状态为未学完

另外，随着`learned_sections`字段不断累加，最终会到达课程的最大小节数，这就意味着当前课程被全部学完了。那么课程状态需要从“学习中”变更为“已学完”。

综上，最终的提交学习记录处理流程如图：



## 2.2.2.表单实体

请求参数比较多，所以需要定义一个表单DTO实体，这个在课前资料已经提供好了：

窗 > 新加卷 (D:) > 课程资料 > 天机学堂 > 课件 > day03-学习计划和进度 > 资料 >

名称	类型	大小
 learning_record.sql	SQL 源文件	5 KB
 LearningPlanDTO.java	Java 源文件	1 KB
 LearningPlanPageVO.java	Java 源文件	2 KB
 LearningPlanVO.java	Java 源文件	1 KB
 LearningRecordFormDTO.java	Java 源文件	2 KB
 SectionType.java	Java 源文件	1 KB

黑马程序员-研究院

具体代码如下：

```
1 package com.tianji.learning.domain.dto;
2
3 import com.tianji.common.validate.annotations.EnumValid;
4 import com.tianji.learning.enums.SectionType;
5 import io.swagger.annotations.ApiModel;
6 import io.swagger.annotations.ApiModelProperty;
7 import lombok.Data;
8
9 import javax.validation.constraints.NotNull;
10 import java.time.LocalDateTime;
11
12 @Data
13 @ApiModel(description = "学习记录")
14 public class LearningRecordFormDTO {
15
16     @ApiModelProperty("小节类型：1-视频，2-考试")
17     @NotNull(message = "小节类型不能为空")
18     @EnumValid(enumeration = {1, 2}, message = "小节类型错误，只能是：1-视频，2-考
19         试")
19     private SectionType sectionType;
20
21     @ApiModelProperty("课表id")
22     @NotNull(message = "课表id不能为空")
23     private Long lessonId;
24
25     @ApiModelProperty("对应节的id")
26     @NotNull(message = "节的id不能为空")
27     private Long sectionId;
```

```

28
29     @ApiModelProperty("视频总时长, 单位秒")
30     private Integer duration;
31
32     @ApiModelProperty("视频的当前观看时长, 单位秒, 第一次提交填0")
33     private Integer moment;
34
35     @ApiModelProperty("提交时间")
36     private LocalDateTime commitTime;
37 }

```

### 2.2.3.代码实现

首先在 `tj-learning` 模块下的

`com.tianji.learning.controller.LearningRecordController` 下定义接口:

```

1 package com.tianji.learning.controller;
2
3
4 import com.tianji.api.dto.learning.LearningLessonDTO;
5 import com.tianji.learning.domain.dto.LearningRecordFormDTO;
6 import com.tianji.learning.service.ILearningRecordService;
7 import io.swagger.annotations.Api;
8 import io.swagger.annotations.ApiOperation;
9 import io.swagger.annotations.ApiParam;
10 import lombok.RequiredArgsConstructor;
11 import org.springframework.web.bind.annotation.*;
12
13 /**
14  * <p>
15  * 学习记录表 前端控制器
16  * </p>
17  *
18  * @author 虎哥
19  * @since 2022-12-10
20  */
21 @RestController
22 @RequestMapping("/learning-records")
23 @Api(tags = "学习记录的相关接口")
24 @RequiredArgsConstructor
25 public class LearningRecordController {
26
27     private final ILearningRecordService recordService;

```

```

28
29     @ApiOperation("查询指定课程的学习记录")
30     @GetMapping("/course/{courseId}")
31     public LearningLessonDTO queryLearningRecordByCourse(
32         @ApiParam(value = "课程id", example = "2") @PathVariable("courseId")
33         Long courseId){
34         return recordService.queryLearningRecordByCourse(courseId);
35     }
36     @ApiOperation("提交学习记录")
37     @PostMapping
38     public void addLearningRecord(@RequestBody LearningRecordFormDTO formDTO){
39         recordService.addLearningRecord(formDTO);
40     }
41 }

```

然后在 `com.tianji.learning.service.ILearningRecordService` 中定义方法：

```

1 package com.tianji.learning.service;
2
3 import com.baomidou.mybatisplus.extension.service.IService;
4 import com.tianji.api.dto.learning.LearningLessonDTO;
5 import com.tianji.learning.domain.dto.LearningRecordFormDTO;
6 import com.tianji.learning.domain.po.LearningRecord;
7
8 /**
9  * <p>
10  * 学习记录表 服务类
11  * </p>
12  *
13  * @author 虎哥
14  * @since 2022-12-10
15  */
16 public interface ILearningRecordService extends IService<LearningRecord> {
17
18     LearningLessonDTO queryLearningRecordByCourse(Long courseId);
19
20     void addLearningRecord(LearningRecordFormDTO formDTO);
21 }

```

最后在 `com.tianji.learning.service.impl.LearningRecordServiceImpl` 中定义实现类：

```

1 package com.tianji.learning.service.impl;
2
3 import com.baomidou.mybatisplus.extension.service.impl.ServiceImpl;
4 import com.tianji.api.client.course.CourseClient;
5 import com.tianji.api.dto.course.CourseFullInfoDTO;
6 import com.tianji.api.dto.leanring.LearningLessonDTO;
7 import com.tianji.api.dto.leanring.LearningRecordDTO;
8 import com.tianji.common.exceptions.BizIllegalException;
9 import com.tianji.common.exceptions.DbException;
10 import com.tianji.common.utils.BeanUtils;
11 import com.tianji.common.utils.UserContext;
12 import com.tianji.learning.domain.dto.LearningRecordFormDTO;
13 import com.tianji.learning.domain.po.LearningLesson;
14 import com.tianji.learning.domain.po.LearningRecord;
15 import com.tianji.learning.enums.LessonStatus;
16 import com.tianji.learning.enums.SectionType;
17 import com.tianji.learning.mapper.LearningRecordMapper;
18 import com.tianji.learning.service.ILearningLessonService;
19 import com.tianji.learning.service.ILearningRecordService;
20 import lombok.RequiredArgsConstructor;
21 import org.springframework.stereotype.Service;
22 import org.springframework.transaction.annotation.Transactional;
23
24 import java.util.List;
25
26 /**
27  * <p>
28  * 学习记录表 服务实现类
29  * </p>
30  */
31 @Service
32 @RequiredArgsConstructor
33 public class LearningRecordServiceImpl extends
    ServiceImpl<LearningRecordMapper, LearningRecord> implements
    ILearningRecordService {
34
35     private final ILearningLessonService lessonService;
36
37     private final CourseClient courseClient;
38
39     // ... 略
40
41     @Override
42     @Transactional
43     public void addLearningRecord(LearningRecordFormDTO recordDTO) {
44         // 1. 获取登录用户
45         Long userId = UserContext.getUser();

```

```

46         // 2.处理学习记录
47         boolean finished = false;
48         if (recordDTO.getSectionType() == SectionType.VIDEO) {
49             // 2.1.处理视频
50             finished = handleVideoRecord(userId, recordDTO);
51         }else{
52             // 2.2.处理考试
53             finished = handleExamRecord(userId, recordDTO);
54         }
55
56         // 3.处理课表数据
57         handleLearningLessonsChanges(recordDTO, finished);
58     }
59
60     private void handleLearningLessonsChanges(LearningRecordFormDTO recordDTO,
61 boolean finished) {
62         // 1.查询课表
63         LearningLesson lesson = lessonService.getById(recordDTO.getLessonId());
64         if (lesson == null) {
65             throw new BizIllegalException("课程不存在, 无法更新数据!");
66         }
67         // 2.判断是否有新的完成小节
68         boolean allLearned = false;
69         if(finished){
70             // 3.如果有新完成的小节, 则需要查询课程数据
71             CourseFullInfoDTO cInfo =
72 courseClient.getCourseInfoById(lesson.getCourseId(), false, false);
73             if (cInfo == null) {
74                 throw new BizIllegalException("课程不存在, 无法更新数据!");
75             }
76             // 4.比较课程是否全部学完: 已学习小节 >= 课程总小节
77             allLearned = lesson.getLearnedSections() + 1 >=
78 cInfo.getSectionNum();
79         }
80         // 5.更新课表
81         lessonService.lambdaUpdate()
82             .set(lesson.getLearnedSections() == 0,
83 LearningLesson::getStatus, LessonStatus.LEARNING.getValue())
84             .set(allLearned, LearningLesson::getStatus,
85 LessonStatus.FINISHED.getValue())
86             .set(!finished, LearningLesson::getLatestSectionId,
87 recordDTO.getSectionId())
88             .set(!finished, LearningLesson::getLatestLearnTime,
89 recordDTO.getCommitTime())
90             .setSql(finished, "learned_sections = learned_sections + 1")
91             .eq(LearningLesson::getId, lesson.getId())
92             .update();

```

```
86     }
87
88     private boolean handleVideoRecord(Long userId, LearningRecordFormDTO
recordDTO) {
89         // 1.查询旧的学习记录
90         LearningRecord old = queryOldRecord(recordDTO.getLessonId(),
recordDTO.getSectionId());
91         // 2.判断是否存在
92         if (old == null) {
93             // 3.不存在, 则新增
94             // 3.1.转换PO
95             LearningRecord record = BeanUtils.copyBean(recordDTO,
LearningRecord.class);
96             // 3.2.填充数据
97             record.setUserId(userId);
98             // 3.3.写入数据库
99             boolean success = save(record);
100             if (!success) {
101                 throw new DbException("新增学习记录失败! ");
102             }
103             return false;
104         }
105         // 4.存在, 则更新
106         // 4.1.判断是否是第一次完成
107         boolean finished = !old.getFinished() && recordDTO.getMoment() * 2 >=
recordDTO.getDuration();
108         // 4.2.更新数据
109         boolean success = lambdaUpdate()
110             .set(LearningRecord::getMoment, recordDTO.getMoment())
111             .set(finished, LearningRecord::getFinished, true)
112             .set(finished, LearningRecord::getFinishTime,
recordDTO.getCommitTime())
113             .eq(LearningRecord::getId, old.getId())
114             .update();
115         if (!success) {
116             throw new DbException("更新学习记录失败! ");
117         }
118         return finished ;
119     }
120
121     private LearningRecord queryOldRecord(Long lessonId, Long sectionId) {
122         return lambdaQuery()
123             .eq(LearningRecord::getLessonId, lessonId)
124             .eq(LearningRecord::getSectionId, sectionId)
125             .one();
126     }
127
```

```
128     private boolean handleExamRecord(Long userId, LearningRecordFormDTO
recordDTO) {
129         // 1.转换DTO为PO
130         LearningRecord record = BeanUtils.copyBean(recordDTO,
LearningRecord.class);
131         // 2.填充数据
132         record.setUserId(userId);
133         record.setFinished(true);
134         record.setFinishTime(recordDTO.getCommitTime());
135         // 3.写入数据库
136         boolean success = save(record);
137         if (!success) {
138             throw new DbException("新增考试记录失败!");
139         }
140         return true;
141     }
142 }
```

## 2.3.创建学习计划

回顾下接口信息：

参数	说明		
请求方式	POST		
请求路径	/lessons/plans		
请求参数	参数名	类型	说明
	courseld	Long	课程id
	weekFreq	Integer	计划每周学习频率
返回值	无		

### 2.3.1.思路分析

创建学习计划，本质就是让用户设定自己每周的学习频率：



## 修改计划

选择课程

Java课程

每周学习  
章节数

5

预计学习  
完成时间

预计 2022年7月10日 完成

取消

修改

黑马程序员-研究院

虽说接口是创建学习计划，但本质这是一个更新的接口。因为学习计划字段都保存在learning\_lesson表中。

```
1 CREATE TABLE `learning_lesson` (  
2   `id` bigint NOT NULL COMMENT '主键',  
3   `user_id` bigint NOT NULL COMMENT '学员id',  
4   `course_id` bigint NOT NULL COMMENT '课程id',  
5   `status` tinyint NULL DEFAULT 0 COMMENT '课程状态, 0-未学习, 1-学习中, 2-已学完,  
6   3-已失效',  
7   `week_freq` tinyint NULL DEFAULT NULL COMMENT '每周学习频率, 每周3天, 每天2节, 则  
8   频率为6',  
9   `plan_status` tinyint NOT NULL DEFAULT 0 COMMENT '学习计划状态, 0-没有计划, 1-计  
10  划进行中',  
11  `learned_sections` int NOT NULL DEFAULT 0 COMMENT '已学习小节数量',  
12  `latest_section_id` bigint NULL DEFAULT NULL COMMENT '最近一次学习的小节id',  
13  `latest_learn_time` datetime NULL DEFAULT NULL COMMENT '最近一次学习的时间',  
14  `create_time` datetime NOT NULL DEFAULT CURRENT_TIMESTAMP COMMENT '创建时间',  
15  `expire_time` datetime NOT NULL COMMENT '过期时间',  
16  `update_time` datetime NOT NULL DEFAULT CURRENT_TIMESTAMP ON UPDATE  
17  CURRENT_TIMESTAMP COMMENT '更新时间',  
18  PRIMARY KEY (`id`) USING BTREE,  
19  UNIQUE INDEX `idx_user_id` (`user_id`, `course_id`) USING BTREE  
20 ) ENGINE = InnoDB CHARACTER SET = utf8mb4 COLLATE = utf8mb4_0900_ai_ci COMMENT  
21 = '学生课程表' ROW_FORMAT = Dynamic;
```

当我们创建学习计划时，就是更新 learning\_lesson 表，写入 week\_freq 并更新 plan\_status 为计划进行中即可。







## 2.3.2.表单实体

表单包含两个字段：

- courseId
- weekFreq

前端是以JSON方式提交，我们需要定义一个表单DTO实体。在课前资料中已经提供给大家了：

新加卷 (D:) > 课程资料 > 天机学堂 > 课件 > day03-学习计划和进度 > 资料 >

名称	类型	大小
 learning_record.sql	SQL 源文件	5 KB
 LearningPlanDTO.java	Java 源文件	1 KB
 LearningPlanPageVO.java	Java 源文件	2 KB
 LearningPlanVO.java	Java 源文件	1 KB
 LearningRecordFormDTO.java	Java 源文件	2 KB
 SectionType.java	Java 源文件	1 KB

黑马程序员-研究院

具体代码：

```
1 package com.tianji.learning.domain.dto;
2
3 import io.swagger.annotations.ApiModel;
4 import io.swagger.annotations.ApiModelProperty;
5 import lombok.Data;
6 import org.hibernate.validator.constraints.Range;
7
8 import javax.validation.constraints.Min;
9 import javax.validation.constraints.NotNull;
10
11 @Data
12 @ApiModel(description = "学习计划表单实体")
13 public class LearningPlanDTO {
14     @NotNull
15     @ApiModelProperty("课程表id")
16     @Min(1)
17     private Long courseId;
18     @NotNull
19     @Range(min = 1, max = 50)
20     @ApiModelProperty("每周学习频率")
21     private Integer freq;
```

### 2.3.3.代码实现

首先，在 `com.tianji.learning.controller.LearningLessonController` 中添加一个接口：

```

1 package com.tianji.learning.controller;
2
3 import com.tianji.learning.domain.dto.LearningPlanDTO;
4 import com.tianji.learning.service.ILearningLessonService;
5 import io.swagger.annotations.Api;
6 import io.swagger.annotations.ApiOperation;
7 import io.swagger.annotations.ApiParam;
8 import lombok.RequiredArgsConstructor;
9 import org.springframework.web.bind.annotation.*;
10
11 import javax.validation.Valid;
12
13 /**
14  * <p>
15  * 学生课程表 前端控制器
16  * </p>
17  *
18  * @author 虎哥
19  * @since 2022-12-02
20  */
21 @RestController
22 @RequestMapping("/lessons")
23 @Api(tags = "我的课表相关接口")
24 @RequiredArgsConstructor
25 public class LearningLessonController {
26
27     private final ILearningLessonService lessonService;
28
29     // 略。。。
30
31     @ApiOperation("创建学习计划")
32     @PostMapping("/plans")
33     public void createLearningPlans(@Valid @RequestBody LearningPlanDTO
planDTO){

```

```
34         lessonService.createLearningPlan(planDTO.getCourseId(),
35         planDTO.getFreq());
36     }
```

然后，在 `com.tianji.learning.service.ILearningLessonService` 中定义service方法：

```
1 package com.tianji.learning.service;
2
3 import com.baomidou.mybatisplus.extension.service.IService;
4 import com.tianji.learning.domain.po.LearningLesson;
5
6 import java.util.List;
7
8 /**
9  * <p>
10  * 学生课程表 服务类
11  * </p>
12  */
13 public interface ILearningLessonService extends IService<LearningLesson> {
14     // ... 略
15
16     void createLearningPlan(Long courseId, Integer freq);
17 }
```

最后，在 `com.tianji.learning.service.impl.LearningLessonServiceImpl` 中实现方法：

```
1 // ... 略
2
3 @Override
4 public void createLearningPlan(Long courseId, Integer freq) {
5     // 1. 获取当前登录的用户
6     Long userId = UserContext.getUser();
7     // 2. 查询课表中的指定课程有关的数据
8     LearningLesson lesson = queryByUserAndCourseId(userId, courseId);
9     AssertUtils.isNotNull(lesson, "课程信息不存在!");
10    // 3. 修改数据
11    LearningLesson l = new LearningLesson();
```

```

12     l.setId(lesson.getId());
13     l.setWeekFreq(freq);
14     if(lesson.getPlanStatus() == PlanStatus.NO_PLAN) {
15         l.setPlanStatus(PlanStatus.PLAN_RUNNING);
16     }
17     updateById(l);
18 }
19
20 // ... 略

```

## 2.4.查询学习计划进度

页面原型如图：



黑马程序员-研究院

接口回顾：

参数	说明		
请求方式	GET		
请求路径	/lessons/plans		
请求参数	分页参数：PageQuery		
返回值	参数名	类型	说明
	weekPoints	int	本周学习积分

	weekFinished	int	本周已学完小节数量		
	weekTotalPlan	int	本周计划学习小节数量		
	list	Array	参数	类型	说明
			courseId	Long	课程id
			courseName	String	课程名称
			weekLearnedSections	int	本周学习的小节数量
			weekFreq	int	本周计划学习数量
			learnedSections	int	总已学习小节数量
			sections	int	总小节数量
			latestLearnTime	LocalDateTime	最近一次学习时间

2.4.1.思路分析

要查询的数据分为两部分：

- 本周计划学习的每个课程的学习进度
- 本周计划学习的课程总的学习进度

对于**本周计划学习的每个课程的学习进度**，首先需要查询出学习中的LearningLesson的信息，查询条件包括：

- 属于当前登录用户
- 学习计划进行中

查询到的LearningLesson可能有多个，而且查询到的PO数据跟最终的VO相比还有差距：

PO：

VO：

名称	数据类型	注释
id	BIGINT	主键
user_id	BIGINT	学员id
course_id	BIGINT	课程id
status	TINYINT	课程状态, 0-未学习, 1-已学习
week_freq	TINYINT	每周学习频率, 每周3天
plan_status	TINYINT	学习计划状态, 0-没有计划, 1-有计划
learned_sections	INT	已学习小节数量
latest_section_id	BIGINT	最近一次学习的小节id
latest_learn_time	DATETIME	最近一次学习的时间
create_time	DATETIME	创建时间

参数	类型	说明
courseId	Long	课程id
courseName	String	课程名称
weekSections	int	本周学习的小节数量
weekFreq	int	本周计划学习数量
learnedSections	int	总已学习小节数量
sections	int	总小节数量
latestLearnTime	LocalDateTime	最近一次学习时间

具体来说，PO中缺少了courseName和weekSections两个字段。其中courseName可以通过courseId去课程微服务查询。weekSections只能对学习记录做统计得到。

因此，我们需要搜集查询到的课表中的courseId，查询出对应的课程信息；还需要搜集查询到的课表的id，去learning\_record中统计每个课表**本周**已学习的小节数量。

最终遍历处理每个PO，转换为VO格式。

除了本周每个课程的学习进度以外，我们还要统计**本周计划学习的课程总的学习进度**。其中的积分数据暂时不管，剩下的两个需要分别对两张表统计：

- weekTotalPlan：对learning\_lesson表统计，查询计划学习的课程的weekFreq字段做累加即可
- weekFinished：对learning\_record表，对已学完的小节记录做count即可



#### 注意：

虽然这里是分页查询，但是每个用户购买的课程其实是有限的，为了便于数据统计，建议采用查询全部数据，然后手动逻辑分页的方式。这样在统计全部课程学习进度的时候会方便很多。

## 2.4.2. 实体

VO实体已经在课前资料中给出：

窗 > 新加卷 (D:) > 课程资料 > 天机学堂 > 课件 > day03-学习计划和进度 > 资料 >

名称	类型	大小
 learning_record.sql	SQL 源文件	5 KB
 LearningPlanDTO.java	Java 源文件	1 KB
 LearningPlanPageVO.java	Java 源文件	2 KB
 LearningPlanVO.java	Java 源文件	1 KB
 LearningRecordFormDTO.java	Java 源文件	2 KB
 SectionType.java	Java 源文件	1 KB

黑马程序员-研究院

### 2.4.3.代码实现

首先在 `tj-learning` 模块的

`com.tianji.learning.controller.LearningLessonController` 中定义 `controller` 接口：

```
1 @ApiOperation("查询我的学习计划")
2 @GetMapping("/plans")
3 public LearningPlanPageVO queryMyPlans(PageQuery query){
4     return lessonService.queryMyPlans(query);
5 }
```

然后在 `com.tianji.learning.service.ILearningLessonService` 中定义service方法：

```
1 LearningPlanPageVO queryMyPlans(PageQuery query);
```

最后在 `com.tianji.learning.service.impl.LearningLessonServiceImpl` 中实现该方法：

版本1：物理分页，分别统计

```
1
2 @Override
3 public LearningPlanPageVO queryMyPlans(PageQuery query) {
4     LearningPlanPageVO result = new LearningPlanPageVO();
5     // 1. 获取当前登录用户
```



```

6      Long userId = UserContext.getUser();
7      // 2. 获取本周起始时间
8      LocalDate now = LocalDate.now();
9      LocalDateTime begin = DateUtils.getWeekBeginTime(now);
10     LocalDateTime end = DateUtils.getWeekEndTime(now);
11     // 3. 查询总的统计数据
12     // 3.1. 本周总的已学习小节数量
13     Integer weekFinished = recordMapper.selectCount(new
LambdaQueryWrapper<LearningRecord>()
14         .eq(LearningRecord::getUserId, userId)
15         .eq(LearningRecord::getFinished, true)
16         .gt(LearningRecord::getFinishTime, begin)
17         .lt(LearningRecord::getFinishTime, end)
18     );
19     result.setWeekFinished(weekFinished);
20     // 3.2. 本周总的计划学习小节数量
21     Integer weekTotalPlan = getBaseMapper().queryTotalPlan(userId);
22     result.setWeekTotalPlan(weekTotalPlan);
23     // TODO 3.3. 本周学习积分
24
25     // 4. 查询分页数据
26     // 4.1. 分页查询课表信息以及学习计划信息
27     Page<LearningLesson> p = lambdaQuery()
28         .eq(LearningLesson::getUserId, userId)
29         .eq(LearningLesson::getPlanStatus, PlanStatus.PLAN_RUNNING)
30         .in(LearningLesson::getStatus, LessonStatus.NOT_BEGIN,
LessonStatus.LEARNING)
31         .page(query.toMpPage("latest_learn_time", false));
32     List<LearningLesson> records = p.getRecords();
33     if (CollUtils.isEmpty(records)) {
34         return result.emptyPage(p);
35     }
36     // 4.2. 查询课表对应的课程信息
37     Map<Long, CourseSimpleInfoDTO> cMap = queryCourseSimpleInfoList(records);
38     // 4.3. 统计每一个课程本周已学习小节数量
39     List<IdAndNumDTO> list = recordMapper.countLearnedSections(userId, begin,
end);
40     Map<Long, Integer> countMap = IdAndNumDTO.toMap(list);
41     // 4.4. 组装数据VO
42     List<LearningPlanVO> voList = new ArrayList<>(records.size());
43     for (LearningLesson r : records) {
44         // 4.4.1. 拷贝基础属性到vo
45         LearningPlanVO vo = BeanUtils.copyBean(r, LearningPlanVO.class);
46         // 4.4.2. 填充课程详细信息
47         CourseSimpleInfoDTO cInfo = cMap.get(r.getCourseId());
48         if (cInfo != null) {
49             vo.setCourseName(cInfo.getName());

```

```

50         vo.setSections(cInfo.getSectionNum());
51     }
52     // 4.4.3. 每个课程的本周已学习小节数量
53     vo.setWeekLearnedSections(countMap.getOrDefault(r.getId(), 0));
54     voList.add(vo);
55 }
56 return result.pageInfo(p.getTotal(), p.getPages(), voList);
57 }
58
59
60 private Map<Long, CourseSimpleInfoDTO>
    queryCourseSimpleInfoList(List<LearningLesson> records) {
61     // 3.1. 获取课程id
62     Set<Long> cIds =
        records.stream().map(LearningLesson::getCourseId).collect(Collectors.toSet());
63     // 3.2. 查询课程信息
64     List<CourseSimpleInfoDTO> cInfoList = courseClient.getSimpleInfoList(cIds);
65     if (CollUtils.isEmpty(cInfoList)) {
66         // 课程不存在, 无法添加
67         throw new BadRequestException("课程信息不存在!");
68     }
69     // 3.3. 把课程集合处理成Map, key是courseId, 值是course本身
70     Map<Long, CourseSimpleInfoDTO> cMap = cInfoList.stream()
71         .collect(Collectors.toMap(CourseSimpleInfoDTO::getId, c -> c));
72     return cMap;
73 }

```

其中需要调用LearningRecordMapper实现对本周每个课程的已学习小节的统计，对应实现如下：

```

1 public interface LearningRecordMapper extends BaseMapper<LearningRecord> {
2
3     List<IdAndNumDTO> countLearnedSections(
4         @Param("userId") Long userId,
5         @Param("begin") LocalDateTime begin,
6         @Param("end") LocalDateTime end);
7 }

```

对应的SQL如下：

```

1 <select id="countLearnedSections" resultType="com.tianji.api.dto.IdAndNumDTO">
2     SELECT lesson_id AS id, COUNT(1) AS num
3     FROM learning_record

```

```
4 WHERE user_id = #{userId}
5 AND finished = 1
6 AND finish_time > #{begin} AND finish_time < #{end}
7 GROUP BY lesson_id;
8 </select>
```

版本2，不分页，stream流统计：

```
1 @Override
2 public LearningPlanPageVO queryMyPlans(PageQuery query) {
3     LearningPlanPageVO result = new LearningPlanPageVO();
4     // 1. 获取当前登录用户
5     Long userId = UserContext.getUser();
6     // 2. 获取本周起始时间
7     LocalDate now = LocalDate.now();
8     LocalDateTime begin = DateUtils.getWeekBeginTime(now);
9     LocalDateTime end = DateUtils.getWeekEndTime(now);
10    // 3. 查询本周计划学习的所有课程，满足三个条件：属于当前用户、有学习计划、学习中
11    List<LearningLesson> lessons = lambdaQuery()
12        .eq(LearningLesson::getUserId, userId)
13        .eq(LearningLesson::getPlanStatus, PlanStatus.PLAN_RUNNING)
14        .in(LearningLesson::getStatus, LessonStatus.NOT_BEGIN,
15            LessonStatus.LEARNING)
16        .list();
17    if (CollUtils.isEmpty(lessons)) {
18        return null;
19    }
20    // 4. 统计当前用户每个课程的已学习小节数量
21    List<LearningRecord> learnedRecords = recordMapper.selectList(new
22        QueryWrapper<LearningRecord>().lambda()
23            .eq(LearningRecord::getUserId, userId)
24            .eq(LearningRecord::getFinished, true)
25            .gt(LearningRecord::getFinishTime, begin)
26            .lt(LearningRecord::getFinishTime, end)
27        );
28    Map<Long, Long> countMap = learnedRecords.stream()
29        .collect(Collectors.groupingBy(LearningRecord::getLessonId,
30            Collectors.counting()));
31    // 5. 查询总的统计数据
32    // 5.1. 本周总的已学习小节数量
33    int weekFinished = learnedRecords.size();
```

```

32     result.setWeekFinished(weekFinished);
33     // 5.2.本周总的计划学习小节数量
34     int weekTotalPlan =
        lessons.stream().mapToInt(LearningLesson::getWeekFreq).sum();
35     result.setWeekTotalPlan(weekTotalPlan);
36     // TODO 5.3.本周学习积分
37
38     // 6.处理分页数据
39     // 6.1.分页查询课表信息以及学习计划信息
40     Page<LearningLesson> p = new Page<>(query.getPageNo(),
        query.getPageSize(), lessons.size());
41     List<LearningLesson> records = CollUtils.sub(lessons, query.from(),
        query.from() + query.getPageSize());
42     if (CollUtils.isEmpty(records)) {
43         return result;
44     }
45     // 6.2.查询课表对应的课程信息
46     Map<Long, CourseSimpleInfoDTO> cMap = queryCourseInfo(records);
47     // 6.3.组装数据VO
48     List<LearningPlanVO> voList = new ArrayList<>(records.size());
49     for (LearningLesson r : records) {
50         // 6.4.1.拷贝基础属性到vo
51         LearningPlanVO vo = BeanUtils.copyBean(r, LearningPlanVO.class);
52         // 6.4.2.填充课程详细信息
53         CourseSimpleInfoDTO cInfo = cMap.get(r.getCourseId());
54         if (cInfo != null) {
55             vo.setCourseName(cInfo.getName());
56             vo.setSections(cInfo.getSectionNum());
57         }
58         // 6.4.3.每个课程的本周已学习小节数量
59         vo.setWeekLearnedSections(countMap.getOrDefault(r.getId(),
            0L).intValue());
60         voList.add(vo);
61     }
62     return result.pageInfo(p.getTotal(), p.getPages(), voList);
63 }

```

## 3.练习

### 3.1.课程过期


编写一个SpringTask定时任务，定期检查learning\_lesson表中的课程是否过期，如果过期则将课程状态修改为已过期。

## 3.2.方案思考

思考题：思考一下目前提交学习记录功能可能存在哪些问题？有哪些可以改进的方向？

## 4.面试题

面试官：你在开发中参与了哪些功能开发让你觉得比较有挑战性？

 答：我参与了整个学习中心的功能开发，其中有很多的学习辅助功能都很有特色。比如视频播放的进度记录。我们网站的课程是以录播视频为主，为了提高用户的学习体验，需要实现视频续播功能。这个功能本身并不复杂，只不过我们产品提出的要求比较高：

- 首先续播时间误差要控制在30秒以内。
- 而且要做到用户突然断开，甚至切换设备后，都可以继续上一次播放

要达成这个目的，使用传统的手段显然是不行的。

首先，要做到切换设备后还能续播，用户的播放进度必须保存在服务端，而不是客户端。

其次，用户突然断开或者切换设备，续播的时间误差不能超过30秒，那播放进度的记录频率就需要比较高。我们会在前端每隔15秒就发起一次心跳请求，提交最新的播放进度，记录到服务端。这样用户下一次续播时直接读取服务端的播放进度，就可以将时间误差控制在15秒左右。

注：此时面试官会追问：播放进度写到服务端保存在哪里？如果写在数据库，那写数据库的压力是不是太大了？等一系列问题，这个会在下一节内容中讲解。