

# Spring Data JPA

浙江大学城市学院

彭彬

[pengb@zucc.edu.cn](mailto:pengb@zucc.edu.cn)

# Spring Data JPA

**JDBC:** Java操作数据库（关系数据库）的接口

**Hibernate:** 基于ORM-Mapping思想开发的库

**JPA:** Java官方为ORM-Mapping操作提供的对象存储规范（从Java EE 5开始）

<https://www.oracle.com/technetwork/java/javaee/tech/persistence-jsp-140049.html>

**Spring Data JPA:** 是Spring Data项目的组成部分，为基于JPA快速方便的实现数据库访问提供支持。

<https://spring.io/projects/spring-data-jpa>

# Spring Data JPA官方介绍解析

Spring Data JPA, part of the larger Spring Data family, makes it easy to **easily implement JPA based repositories**. This module deals with enhanced support for JPA based data access layers. It makes it easier to build Spring-powered applications that use data access technologies.

Implementing a data access layer of an application has been cumbersome for quite a while. **Too much boilerplate code has to be written to execute simple queries** as well as perform pagination, and auditing. Spring Data JPA aims **to significantly improve the implementation of data access layers by reducing** the effort to the amount that's actually needed. As a developer you write your repository interfaces, including custom finder methods, and Spring will **provide the implementation automatically**.

## Features(节选)

- Sophisticated **support to build repositories** based on Spring and JPA
- Support for **Querydsl predicates** and thus type-safe JPA queries
- **Pagination support, dynamic query execution**, ability to integrate custom data access code

# Spring Data项目

Spring Data项目尝试为各种数据访问方案提供统一的解决方案，并且通过最核心的Repository对象来完成核心的数据库操作。比如下面分别是访问图数据库Neo4j、非关系数据库MongoDB和关系数据库MySQL的方法（下面的查询都是自动实现，需要写的代码就是下面这些，而且模式非常相似）：

```
@Component
public interface SchoolRepository extends Neo4jRepository<SchoolNode, Long> {
    SchoolNode findByCode(String code);

    @Query("MATCH (s:School)-[:USE_MODEL]->(m:ModelRoot),p=(m:ModelRoot)<-[*]-(l:ModelLeaf)
        WHERE id(s) = $0 AND id(m) = $1 " +
        "RETURN id(l) as id, l.title as title, l.code as code, reduce(v = \"\", x IN noe
        "ORDER BY l.code")
    List<ModelLeafResult> queryModelLeafOfSchool(Long schoolId, Long modelId);
```

Neo4j

```
public interface PersonRepository extends MongoRepository<Person, String>

    @Query("{ 'firstname' : ?0 }")
    List<Person> findByThePersonsFirstname(String firstname);

}
```

MongoDB

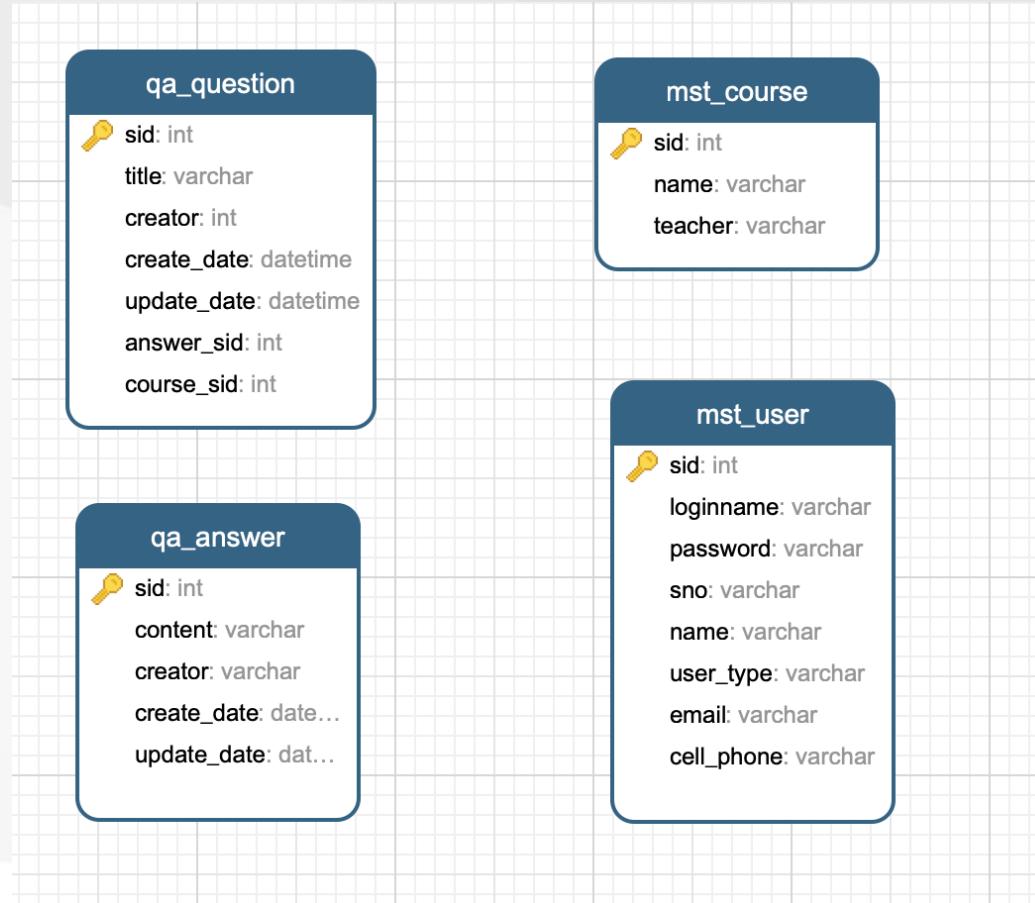
```
public interface UserRepository extends JpaRepository<User, Long> {

    @Query("select u from User u where u.firstname like %?1")
    List<User> findByFirstnameEndsWith(String firstname);
}
```

MySQL

# 基于我们简单设计的数据库来探索JPA

作为示例, 我们沿用上次课程设计了几个简单的表格 (注意: 这是示例! )



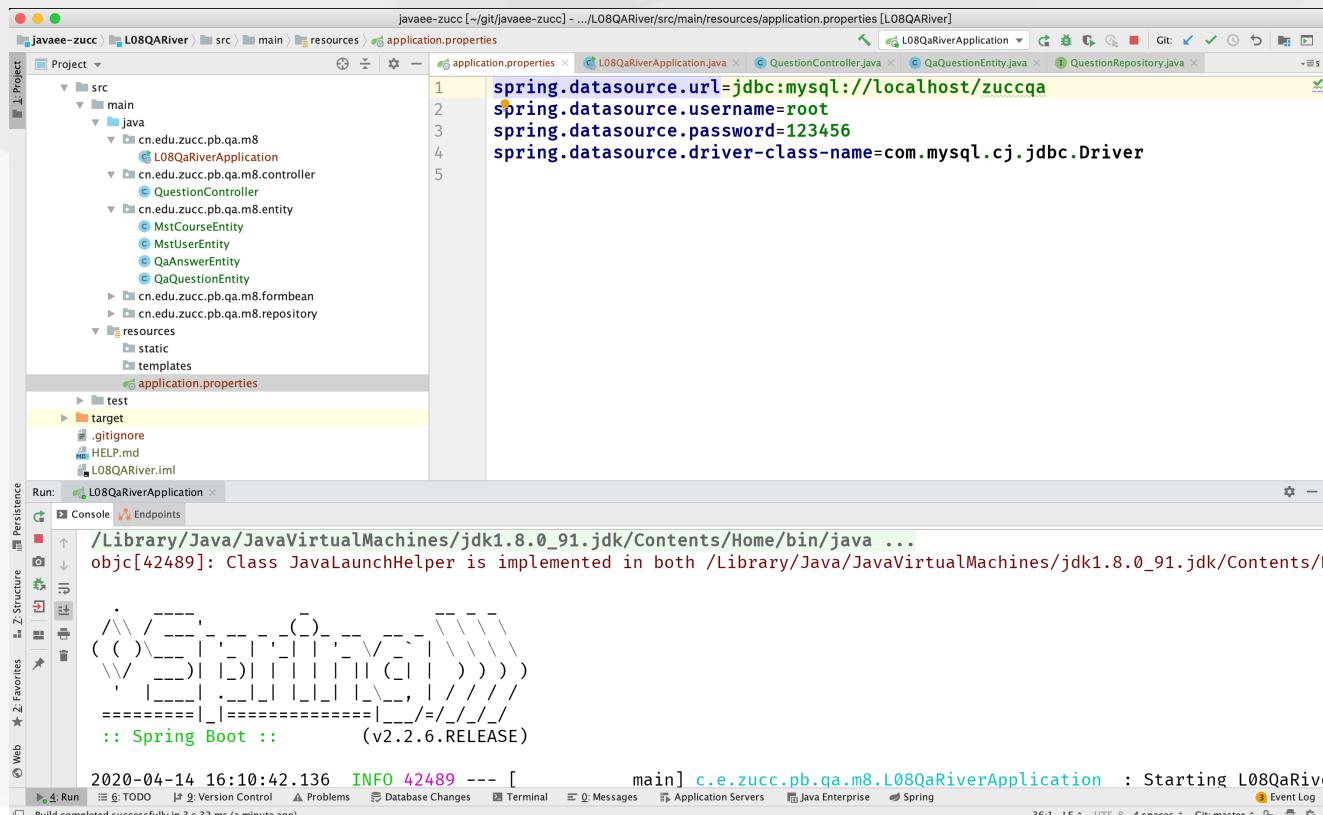
基于JPA我们可以很快完成以下操作:

- 1) 基本的增删改查询
- 2) 处理分页
- 3) 处理多表查询
- 4) 基于SQL语句的各种复杂查询统计

再次重申: 我们为什么不设计外键?

# 配置SpringBoot数据源参数

我们使用spring boot配置数据源，特别简单，在application.properties文件中输入下面的信息即可。（沿用上一课的方式）



The screenshot shows the IntelliJ IDEA interface with the project 'L08QARiver' open. The 'application.properties' file is selected in the left sidebar under 'resources'. The code editor displays the following configuration:

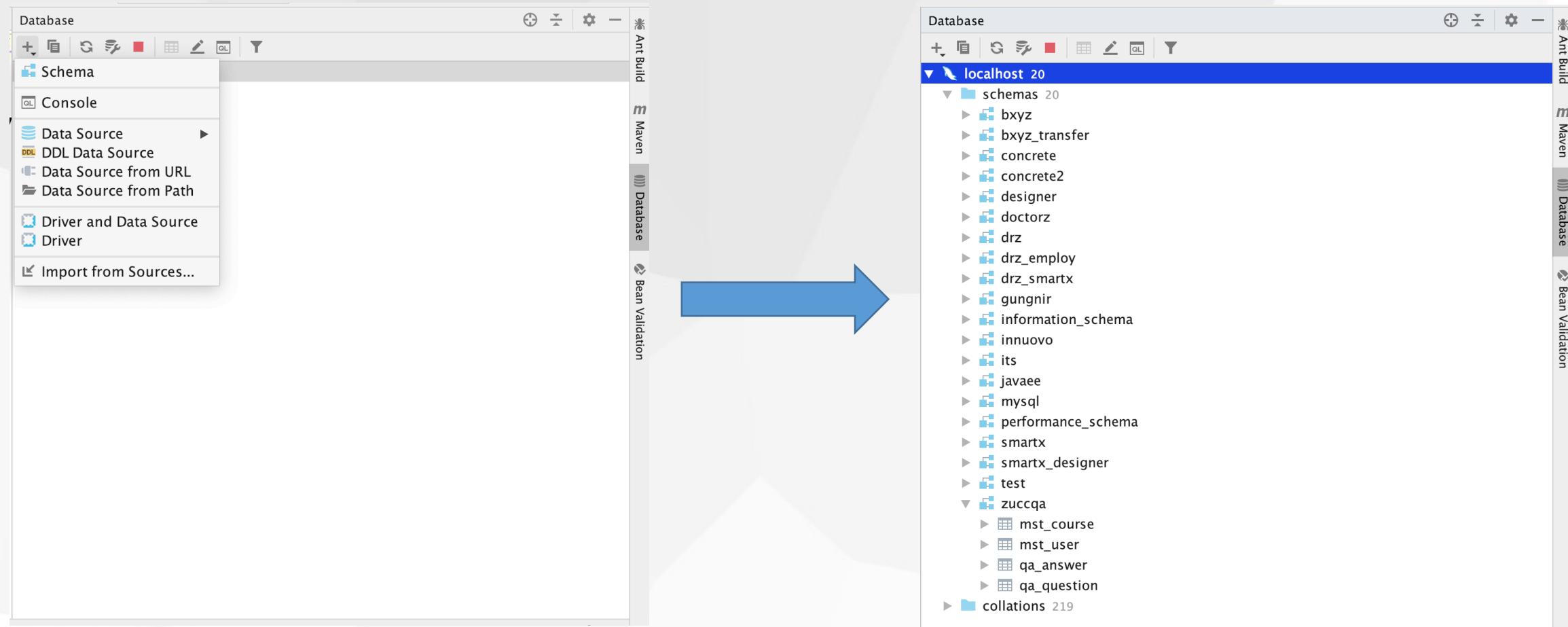
```
spring.datasource.url=jdbc:mysql://localhost/zuccqa
spring.datasource.username=root
spring.datasource.password=123456
spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver
```

The 'Run' tool window at the bottom shows the application is starting successfully with the message: '/Library/Java/JavaVirtualMachines/jdk1.8.0\_91.jdk/Contents/Home/bin/java ...' and 'objc[42489]: Class JavaLaunchHelper is implemented in both /Library/Java/JavaVirtualMachines/jdk1.8.0\_91.jdk/Contents/I...'. The status bar at the bottom indicates the build completed successfully in 3 s 32 ms (a minute ago).

<https://spring.io/guides/gs/accessing-data-mysql/>

# 在Idea中配置数据源

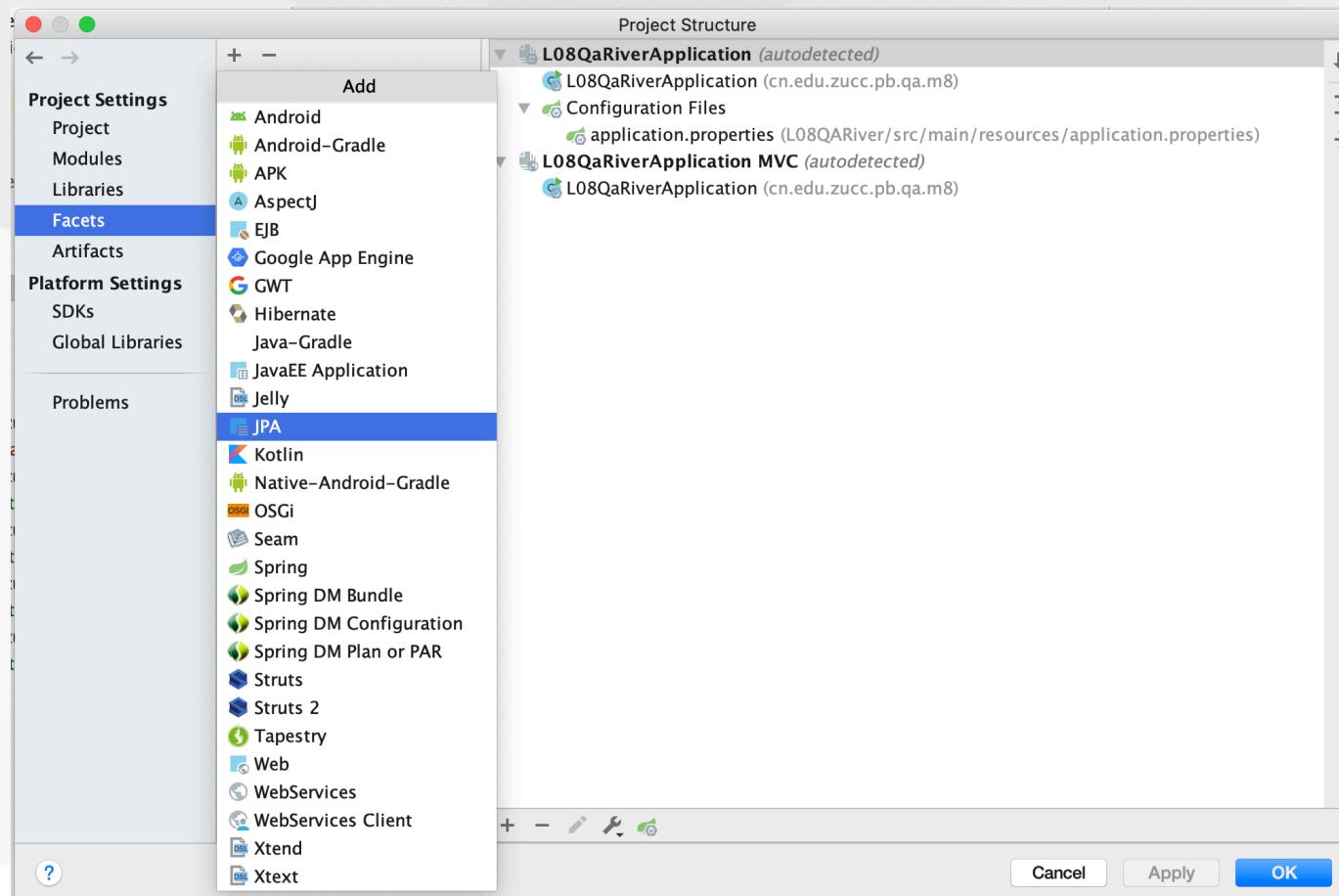
为了方便操作和后续生成各种Entity类，可以通过Idea配置数据源



在右侧打开Database对话框，然后增加一个Datasource连接到你的Mysql数据库，连接成功后可以打开库和表格

# 为项目增加JPA和Hibernate的支持

在Idea中增加JPA和Hibernate支持后，可以使用插件工具自动创建Entity类

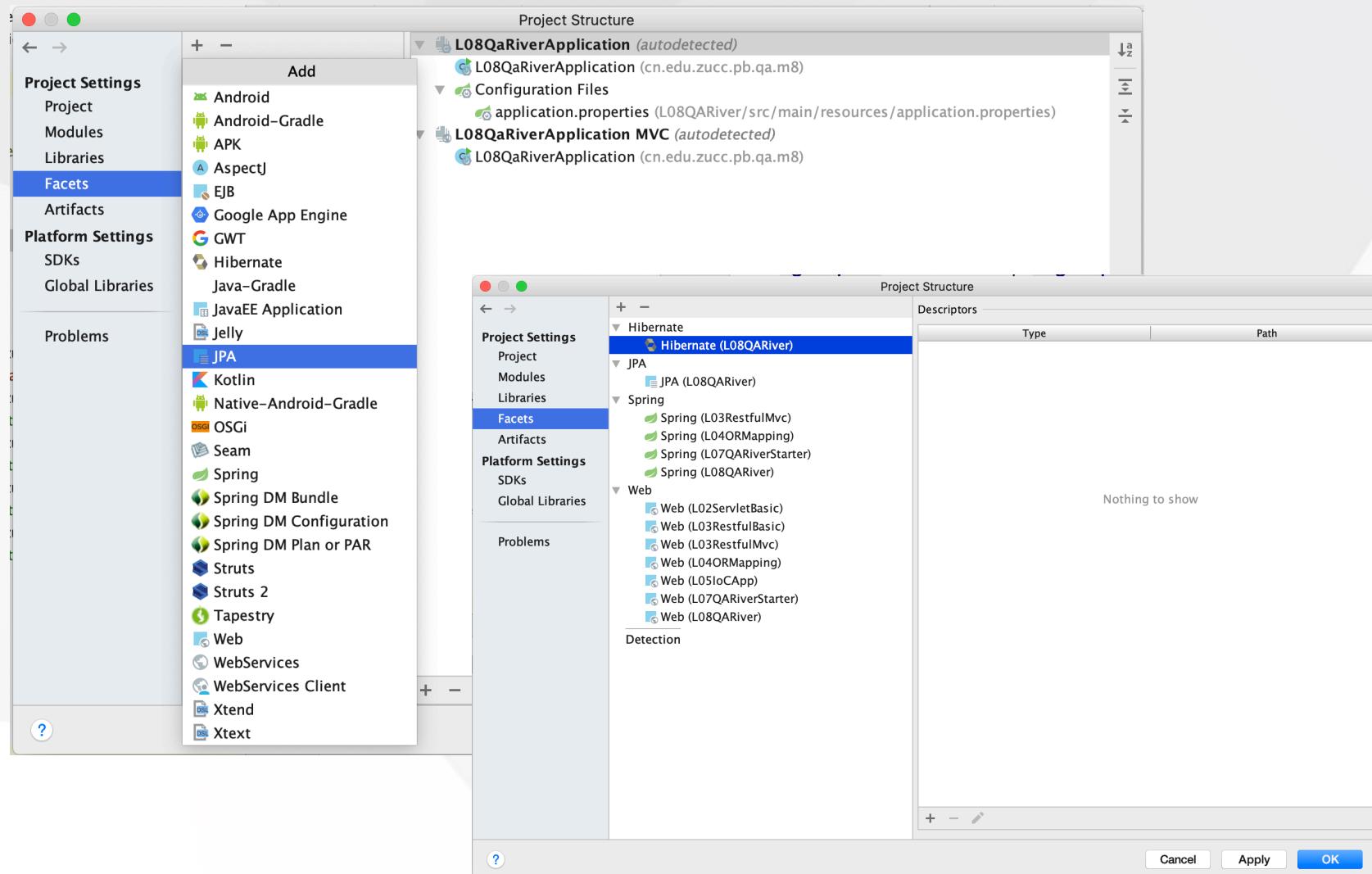


在项目配置中，增加Facets的支持。  
增加的Facets包括

- JPA
- Hibernate

# 为项目增加JPA和Hibernate的支持

在Idea中增加JPA和Hibernate支持后，可以使用插件工具自动创建Entity类

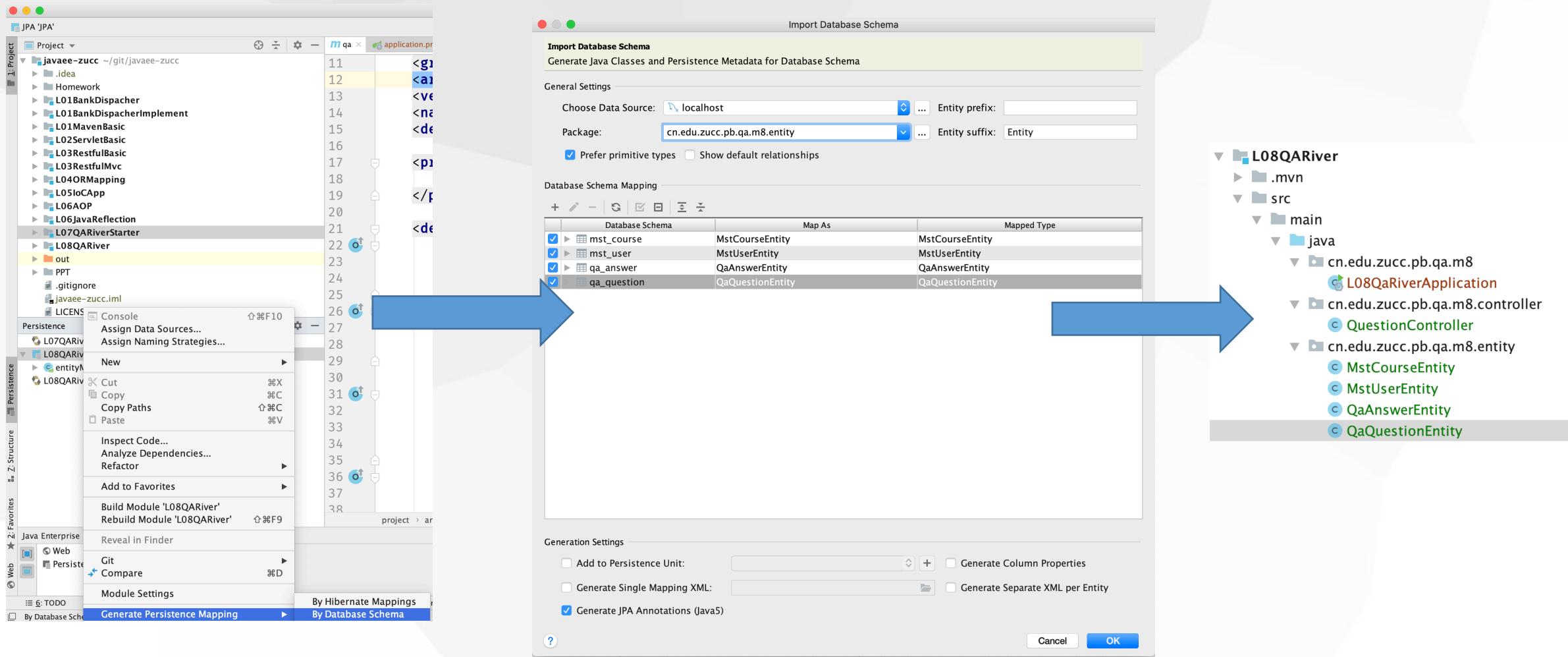


在项目配置中，增加Facets的支持。增加的Facets包括

- JPA
- Hibernate

# 可以从数据库表生成Entity类

上次课中我们是自己手工创建了Entity类（针对Question表），本次我们使用工具一次性为示例的四张表增加对应的Entity类



# JPA Entity类

JPA的Entity类是普通的Java Class。但是这个Java Class有一些特定的标注

```
javaee-zucc [~/git/javaee-zucc - .../L08QARiver/src/main/java/cn/edu/zucc/pb/qa/m8/entity/MstCourseEntity.java [L08QARiver]
Project L04ORMapping L05IoCApp L06AOP L06JavaReflection L07QARiverStarter L08QARiver .mvn src main java cn edu zucc pb qa m8 entity MstCourseEntity
MstCourseEntity.java
1 package cn.edu.zucc.pb.qa.m8.entity;
2
3 import ...
4
5 /**
6  * @author pengbin
7  * @version 1.0
8  * @date 2020-04-14 16:03
9 */
10
11 @Entity
12 @Table(name = "mst_course", schema = "zuccqa", catalog = "")
13 public class MstCourseEntity {
14     private int sid;
15     private String name;
16     private String teacher;
17
18     @Id
19     @Column(name = "sid")
20     public int getSid() { return sid; }
21
22     public void setSid(int sid) { this.sid = sid; }
23
24     @Basic
25     @Column(name = "name")
26     public String getName() { return name; }
27
28     public void setName(String name) { this.name = name; }
29
30     @Basic
31     @Column(name = "teacher")
32     public String getTeacher() { return teacher; }
33
34     public void setTeacher(String teacher) { this.teacher = teacher; }
35
36     @Override
37     public boolean equals(Object o) {
38         if (this == o) return true;
39
40         return false;
41     }
42
43     @Override
44     public int hashCode() {
45         return Objects.hash(sid);
46     }
47
48     @Override
49     public String toString() {
50         return "MstCourseEntity{" +
51             "sid=" + sid +
52             ", name='" + name + '\'' +
53             ", teacher='" + teacher + '\''
54     }
55 }
```

@Entity

@Table

@Id

@Column

@CreationTimestamp

@UpdateTimestamp

JPA定义了非常多的标注，  
参考下面的官方列表

# Entity类

当然我们可以修改Entity类，以支持我们的更多特性，比如增加timestamp标注

```
@Basic  
@Column(name = "create_date")  
@CreationTimestamp  
public Timestamp getCreateDate() { return createDate; }  
  
public void setCreateDate(Timestamp createDate) { this.createDate = createDate; }  
  
@Basic  
@Column(name = "update_date")  
@UpdateTimestamp  
public Timestamp getUpdateDate() { return updateDate; }
```

# 数据库操作的核心Repository

我们通过接口继承JpaRepository就具备的主要的增删改查接口，不用写任何SQL语句。

```
package cn.edu.zucc.pb.qa.repositories;

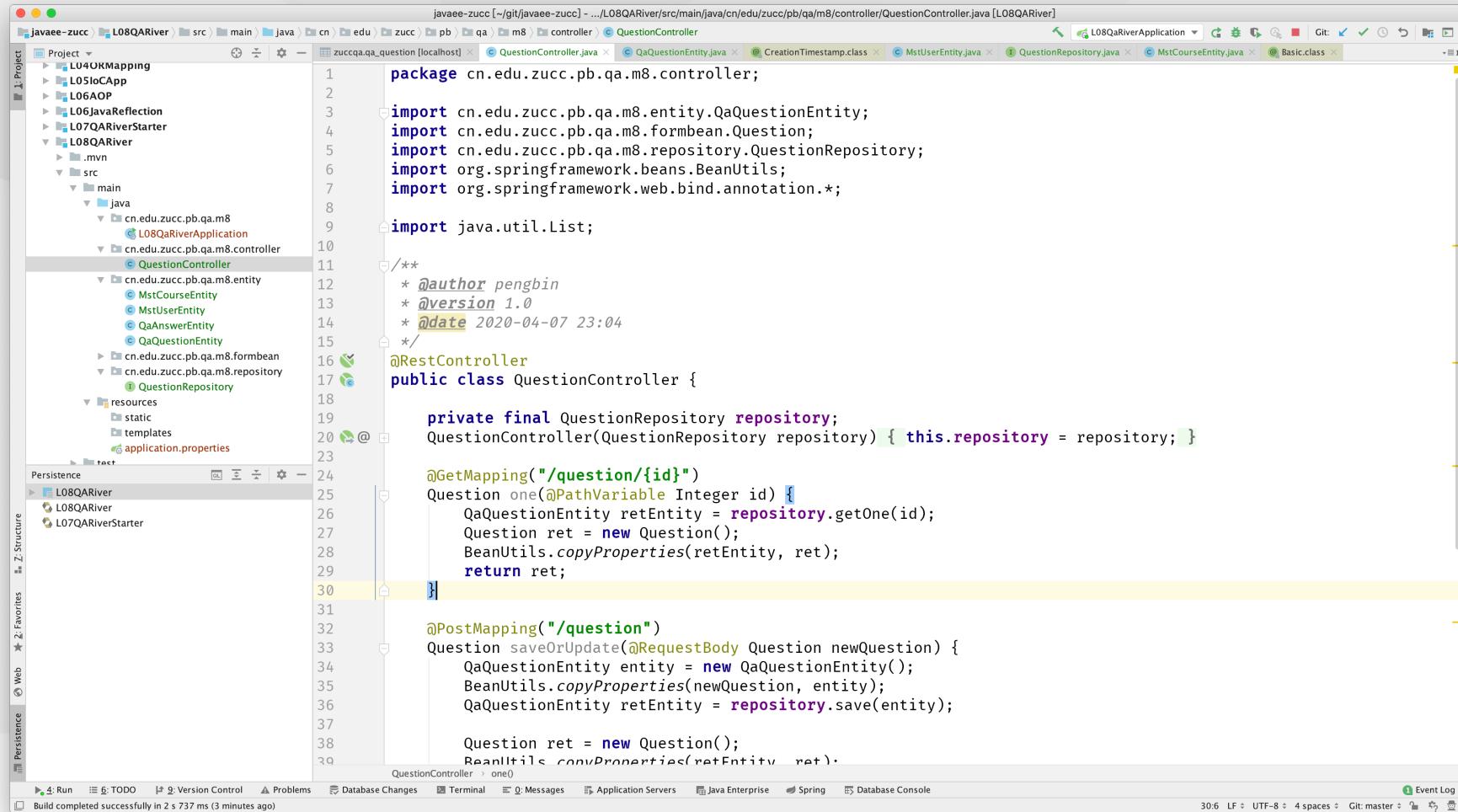
import cn.edu.zucc.pb.qa.entity.QuestionEntity;
import org.springframework.data.jpa.repository.JpaRepository;

/**
 * @author pengbin
 * @version 1.0
 * @date 2020-04-07 23:25
 */
public interface QuestionRepository extends JpaRepository<QuestionEntity, Integer>
}
```

\*具体的SQL操作实现由spring data框架帮我们完成了，所以之前说的自动生成查询语句就体现在这些地方

# Spring MVC

我们先使用Spring MVC框架创建一个Controller，以构造WebApi



The screenshot shows a Java code editor within an IDE. The project structure on the left includes packages like L04ORMapping, L05IOCApp, L06AOP, L06JavaReflection, L07QARiverStarter, L08QARiver, and L08QARiverApplication. The current file is QuestionController.java, which contains the following code:

```
package cn.edu.zucc.pb.qa.m8.controller;

import cn.edu.zucc.pb.qa.m8.entity.QaQuestionEntity;
import cn.edu.zucc.pb.qa.m8.formbean.Question;
import cn.edu.zucc.pb.qa.m8.repository.QuestionRepository;
import org.springframework.beans.BeanUtils;
import org.springframework.web.bind.annotation.*;

import java.util.List;

/**
 * @author pengbin
 * @version 1.0
 * @date 2020-04-07 23:04
 */
@RestController
public class QuestionController {

    private final QuestionRepository repository;
    QuestionController(QuestionRepository repository) { this.repository = repository; }

    @GetMapping("/question/{id}")
    Question one(@PathVariable Integer id) {
        QaQuestionEntity retEntity = repository.getOne(id);
        Question ret = new Question();
        BeanUtils.copyProperties(retEntity, ret);
        return ret;
    }

    @PostMapping("/question")
    Question saveOrUpdate(@RequestBody Question newQuestion) {
        QaQuestionEntity entity = new QaQuestionEntity();
        BeanUtils.copyProperties(newQuestion, entity);
        QaQuestionEntity retEntity = repository.save(entity);

        Question ret = new Question();
        BeanUtils.copyProperties(retEntity, ret);
        return ret;
    }
}
```

The code implements a RESTful API for managing questions. It uses annotations like `@RestController`, `@GetMapping`, and `@PostMapping` to map HTTP requests to controller methods. The `repository` field is injected via constructor injection. The `one` method retrieves a question by its ID, and the `saveOrUpdate` method saves or updates a new question.

@RestController

@GetMapping

@PostMapping

## 依赖注入

Controller中需要repository，这个时候我们就使用了Spring的DI功能注入一个Repository。构造函数注入方式：

```
@RestController  
public class QuestionController {  
  
    private final QuestionRepository repository;  
    QuestionController(QuestionRepository repository){  
        this.repository = repository;  
    }  
}
```

Spring会自动发现依赖关系，然后进行注入。

# 实现一个使用ID查询数据

我们在Controller中增加一个接口， 使用Question的id来获取其实例数据

```
@RestController
public class QuestionController {

    private final QuestionRepository repository;
    QuestionController(QuestionRepository repository) { this.repository = repository; }

    @GetMapping("/question/{id}")
    Question one(@PathVariable Integer id) {
        QaQuestionEntity retEntity = repository.getOne(id);
        Question ret = new Question();
        BeanUtils.copyProperties(retEntity, ret);
        return ret;
    }
}
```

我们获取id参数后， 直接调用repository的getOne方法就可以获取对应的Entity对象， 其需要的select语句及其执行代码全部由spring data框架代劳了。

再次强调为啥我们要创建FormBean对象和Entity对象， 然后复制呢？

# 实现一个使用ID查询数据

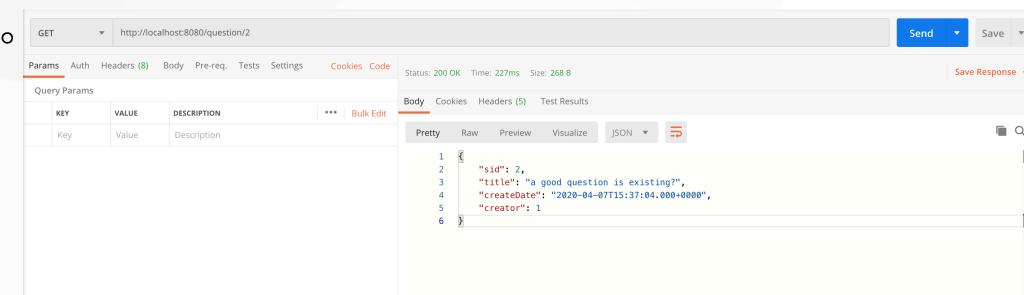
我们在Controller中增加一个接口， 使用Question的id来获取其实例数据

```
@RestController
public class QuestionController {

    private final QuestionRepository repository;
    QuestionController(QuestionRepository repository) { this.repository = repository; }

    @GetMapping("/question/{id}")
    Question one(@PathVariable Integer id) {
        QaQuestionEntity retEntity = repository.getOne(id);
        Question ret = new Question();
        BeanUtils.copyProperties(retEntity, ret);
        return ret;
    }
}
```

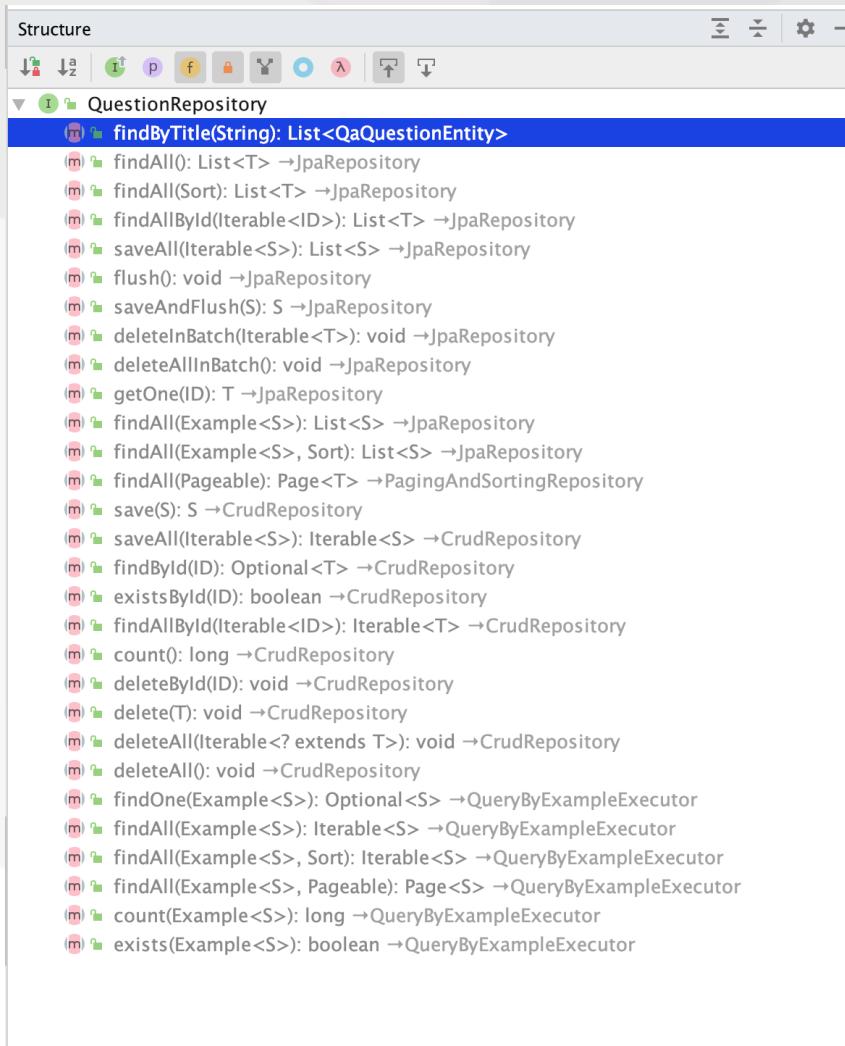
我们获取id参数后， 直接调用repository的getOne方法就可以获取对应的Entity对象， 其需要的select语句及其执行代码全部由spring data框架代劳了。



再次强调为啥我们要创建FormBean对象  
和Entity对象， 然后复制呢？

# Repository里面有什么

通过打开Idea的Structure窗口，我们可以仔细观察到其包含了很多方法，这些方法继承自多个父接口



Findxxxx:这一组方法实现了select功能

Save: 实现insert或者update功能

Delete: 实现delete功能

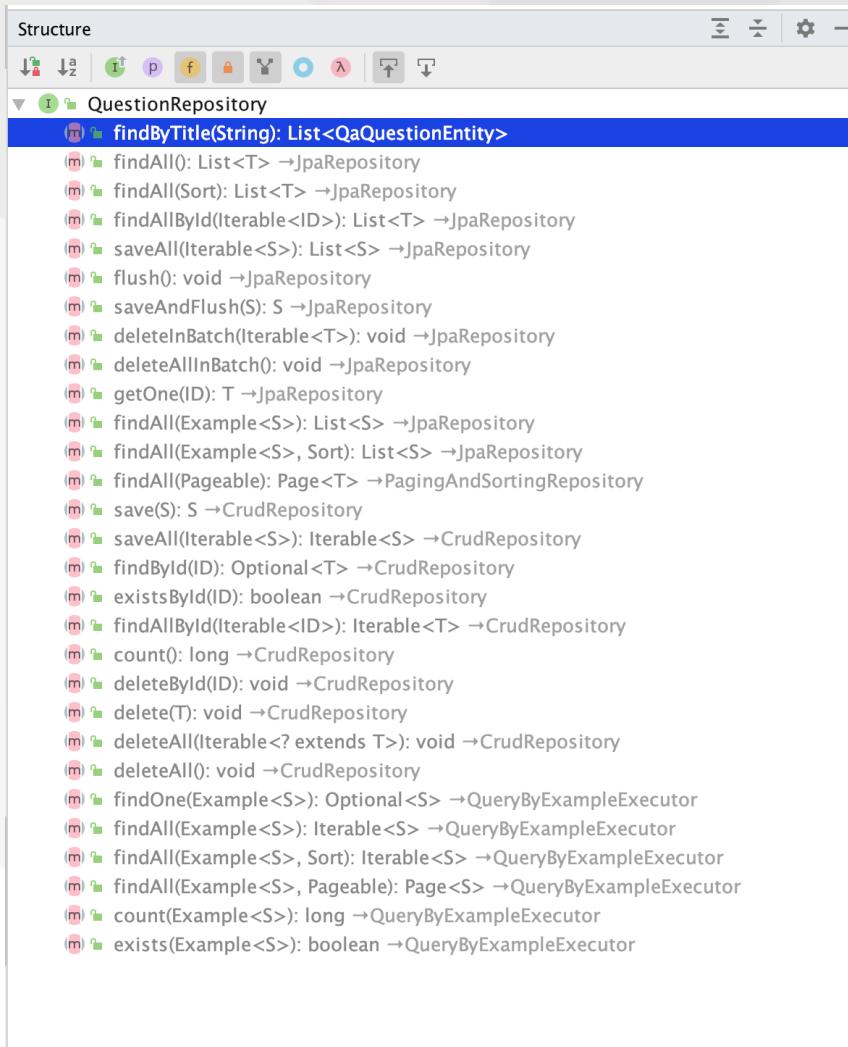
最常用的增删改查功能使用上面的功能就可以了；还有一些常用方法：

Count: 计数

Existsxxxx:判断存在

# Repository基本使用

我们增加一个接口来使用基本的CRUD功能



Findxxxx:这一组方法实现了select功能

Save: 实现insert或者update功能

Delete: 实现delete功能

最常用的增删改查功能使用上面的功能就可以了；还有一些常用方法：

Count: 计数

Existsxxxx:判断存在

# Repository 基本使用

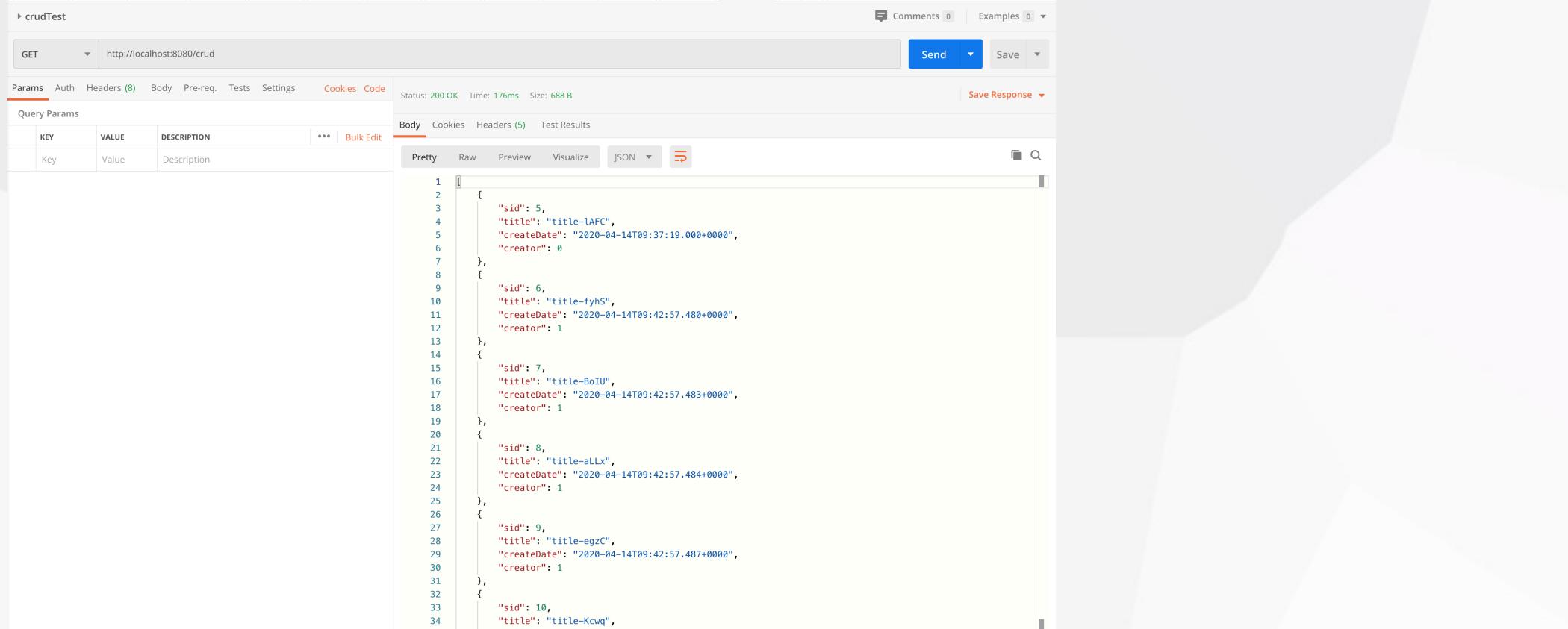
我们增加一个接口来使用基本的CRUD功能

The screenshot shows an IntelliJ IDEA interface with the following details:

- Project Tree:** The left sidebar displays the project structure under "L08QARiver". It includes modules like "src" (main, test), "target", ".gitignore", "HELP.md", "mvnw", "mvnw.cmd", "pom.xml", and "L08QARiver" (with subfolders ".mvn", "src", "main", "java").
- Persistence:** A persistence configuration window is open, showing entities "L08QARiver", "L08QARiver", and "L07QARiverStarter".
- Code Editor:** The main editor shows Java code for a "CrudController" class. The code performs basic CRUD operations on "QuestionEntity", "MstCourseEntity", and "MstUserEntity". It includes logic for saving entities, creating a course, creating a user, and saving 5 questions.
- Toolbars and Status Bar:** The bottom toolbar includes buttons for Run, Debug, TODO, Version Control, Problems, Database Changes, Terminal, Messages, Application Servers, and Spring. The status bar at the bottom right shows the build time as "Build completed successfully in 1 s 359 ms (3 minutes ago)" and the current time as "69:57".

# Repository基本使用

我们增加一个接口来使用基本的CRUD功能



The screenshot shows a REST API testing interface with the following details:

- Request:** GET http://localhost:8080/crud
- Status:** 200 OK (Time: 176ms, Size: 688 B)
- Body:** A JSON array containing 10 objects, each representing a question. The objects have the following structure:

```
1 [  
2 {  
3     "sid": 5,  
4     "title": "title-lAFC",  
5     "createDate": "2020-04-14T09:37:19.000+0000",  
6     "creator": 0  
},  
7 {  
8     "sid": 6,  
9     "title": "title-fyhS",  
10    "createDate": "2020-04-14T09:42:57.480+0000",  
11    "creator": 1  
},  
12 {  
13     "sid": 7,  
14     "title": "title-BoIU",  
15     "createDate": "2020-04-14T09:42:57.483+0000",  
16     "creator": 1  
},  
17 {  
18     "sid": 8,  
19     "title": "title-aLLx",  
20     "createDate": "2020-04-14T09:42:57.484+0000",  
21     "creator": 1  
},  
22 {  
23     "sid": 9,  
24     "title": "title-egzC",  
25     "createDate": "2020-04-14T09:42:57.487+0000",  
26     "creator": 1  
},  
27 {  
28     "sid": 10,  
29     "title": "title-Kcwq",  
30     "createDate": "2020-04-14T09:42:57.488+0000",  
31     "creator": 1  
},  
32 {  
33     "sid": 11,  
34     "title": "title-1Kwv",  
35     "createDate": "2020-04-14T09:42:57.489+0000",  
36     "creator": 1  
},  
37 {  
38     "sid": 12,  
39     "title": "title-1Kwv",  
40     "createDate": "2020-04-14T09:42:57.490+0000",  
41     "creator": 1  
},  
42 {  
43     "sid": 13,  
44     "title": "title-1Kwv",  
45     "createDate": "2020-04-14T09:42:57.491+0000",  
46     "creator": 1  
},  
47 {  
48     "sid": 14,  
49     "title": "title-1Kwv",  
50     "createDate": "2020-04-14T09:42:57.492+0000",  
51     "creator": 1  
},  
52 {  
53     "sid": 15,  
54     "title": "title-1Kwv",  
55     "createDate": "2020-04-14T09:42:57.493+0000",  
56     "creator": 1  
},  
57 }]
```

**Table:** Shows a list of objects with columns: sid, title, creator, createDate, updateDate, answer\_sid, and course\_sid. The data corresponds to the 10 objects in the JSON array.

sid	title	creator	createDate	updateDate	answer_sid	course_sid
6	title-fyhS	1	2020-04-14 04:42:57	2020-04-14 04:42:57	[Null]	1
7	title-BoIU	1	2020-04-14 04:42:57	2020-04-14 04:42:57	[Null]	1
8	title-aLLx	1	2020-04-14 04:42:57	2020-04-14 04:42:57	[Null]	1
9	title-egzC	1	2020-04-14 04:42:57	2020-04-14 04:42:57	[Null]	1
10	title-Kcwq	1	2020-04-14 04:42:57	2020-04-14 04:42:57	[Null]	1

# 使用“约定”查询

我们通过Spring Data JPA的“名称约定”来完成一个查询接口，同样我们不需要编写任何查询的实现代码，依赖Spring自动帮我们完成。我们只需做“合适”的方法命名就可以了

```
 /**
 * @author pengbin
 * @version 1.0
 * @date 2020-04-07 23:25
 */
public interface QuestionRepository extends JpaRepository<QaQuestionEntity, Integer> {
    List<QaQuestionEntity> findByTitleLike(String title);
    List<QaQuestionEntity> findByTitleLikeOrderByTitleDesc(String title);
}
```

我们通过“名称约定”来完成了两个查询

- 根据title的模糊查询
- 根据title的模糊查询并排序

# 使用“约定”查询

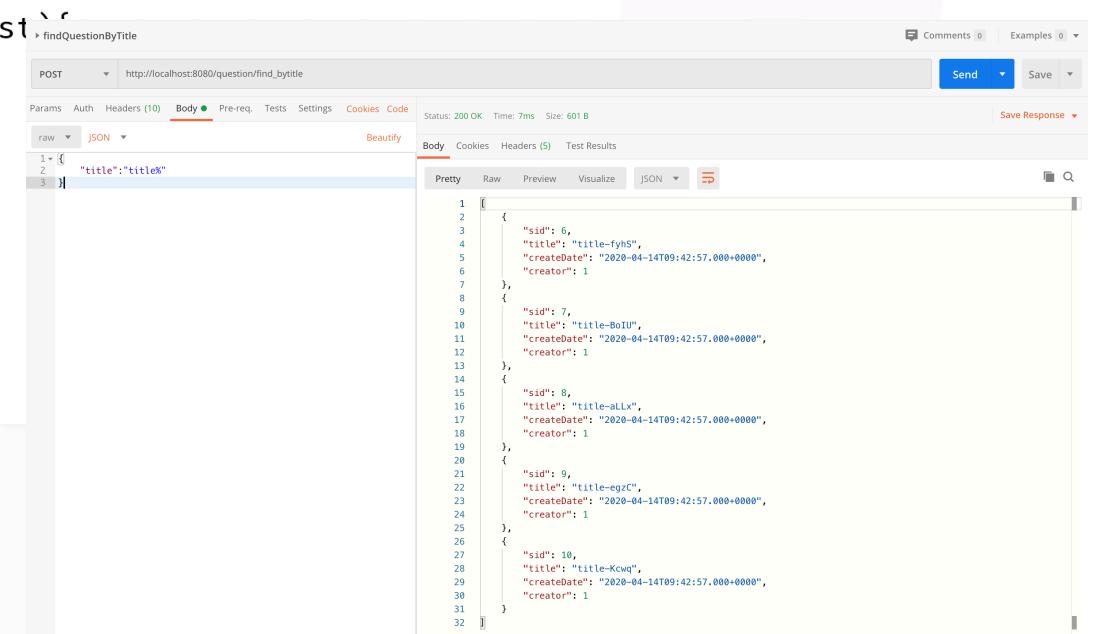
直接调用我们“只写了名称”的方法即可使用

```
@PostMapping("/question/find_bytitle")
List<Question> findByTitle(@RequestBody Map<String, String> queryExample) {
    return convert(repository.findByTitleLike(queryExample.get("title")));
}

@PostMapping("/question/find_bytitle_orderby")
List<Question> findByDateBetween(@RequestBody Map<String, String> queryExample) {
    return convert(repository.findByTitleLikeOrderByTitleDesc(queryExample.get("title")));
}

private List<Question> convert(List<QaQuestionEntity> entityList) {
    List<Question> questionList = new ArrayList<>();
    entityList.stream().forEach(item -> {
        Question q = new Question();
        BeanUtils.copyProperties(item, q);
        questionList.add(q);
    });

    return questionList;
}
```



# 有哪些命名模式?

Spring Data支持非常多常用的命名模式，以支持常见的快速查询，非常方便

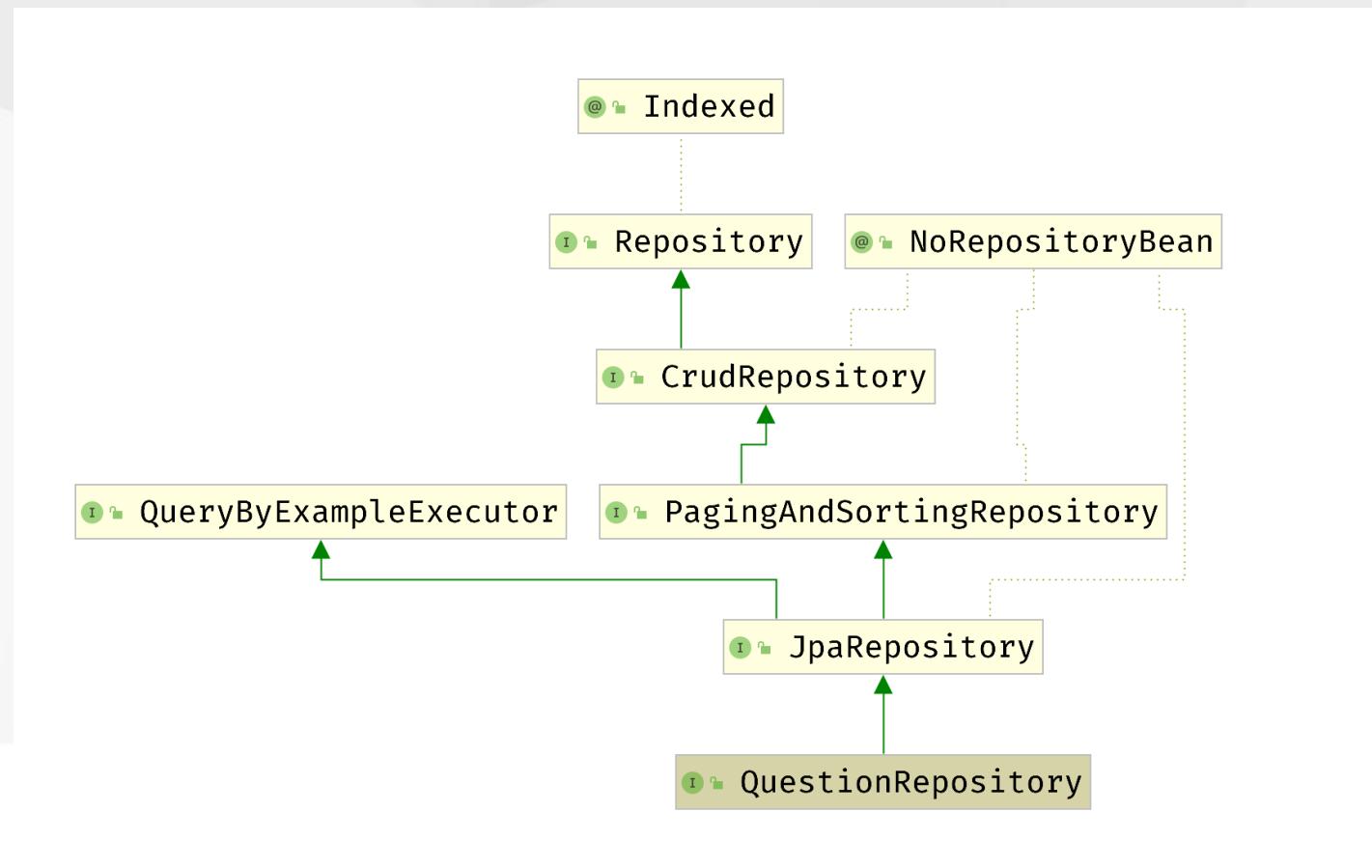
Table 3. Supported keywords inside method names

Keyword	Sample	JPQL snippet
And	findByLastnameAndFirstname	... where x.lastname = ?1 and x.firstname = ?2
Or	findByLastnameOrFirstname	... where x.lastname = ?1 or x.firstname = ?2
Is , Equals	findByFirstname, findByFirstnameIs, findByFirstnameEquals	... where x.firstname = ?1
Between	findByStartDateBetween	... where x.startDate between ?1 and ?2
LessThan	findByAgeLessThan	... where x.age < ?1
LessThanEqual	findByAgeLessThanEqual	... where x.age <= ?1
GreaterThan	findByAgeGreaterThan	... where x.age > ?1
GreaterThanOrEqual	findByAgeGreaterThanOrEqual	... where x.age >= ?1
After	findByStartDateAfter	... where x.startDate > ?1
Before	findByStartDateBefore	... where x.startDate < ?1
IsNull , Null	findByAge(Is)Null	... where x.age is null
IsNotNull , NotNull	findByAge(Is)NotNull	... where x.age not null
Like	findByFirstnameLike	... where x.firstname like ?1
NotLike	findByFirstnameNotLike	... where x.firstname not like ?1
StartingWith	findByFirstnameStartingWith	... where x.firstname like ?1 (parameter bound with appended %)

<https://docs.spring.io/spring-data/jpa/docs/current/reference/html/#jpa.query-methods>

# 分页查询

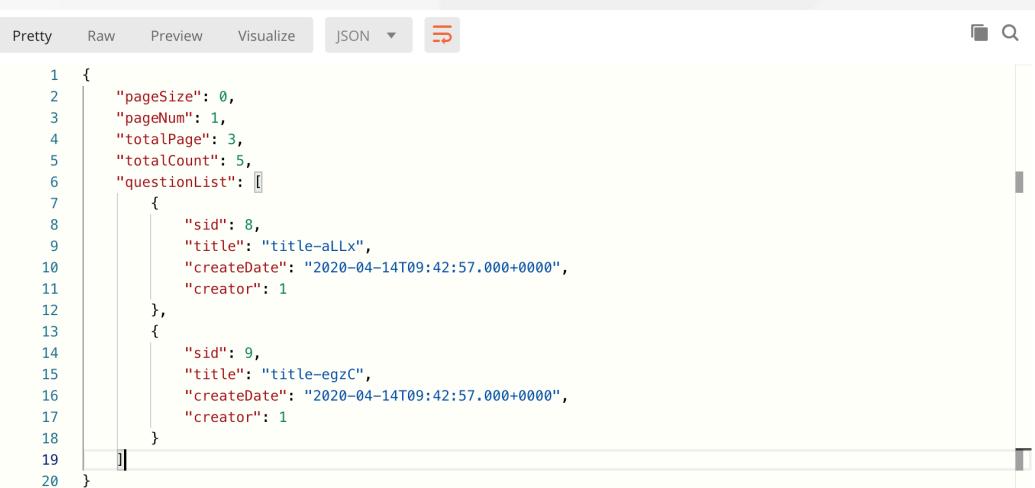
Spring Data JPA提供了分页模式。我们可以通过下图清楚的看到我们的Repository的继承位置，而PagingAndSortingRepository父接口提供了分页的功能，其中提供的分页功能可以直接使用



# 分页查询

直接使用父接口提供的支持分页的findAll方法如下：

```
@PostMapping("/question/find_all_pagination")
QuestionPage findPagination() {
    Page<QaQuestionEntity> thisPage = repository.findAll(PageRequest.of(page: 1, size: 2));
    QuestionPage retPage = new QuestionPage();
    retPage.setPageNum(1);
    retPage.setPageSize(2);
    retPage.setTotalPage(thisPage.getTotalPages());
    retPage.setTotalCount(thisPage.getTotalElements());
    retPage.setQuestionList(convert(thisPage.getContent()));
    return retPage;
}
```



The screenshot shows a JSON editor interface with the following configuration:

- Pretty: Selected
- Raw: Unselected
- Preview: Unselected
- Visualize: Unselected
- JSON: Selected
- Copy icon: Unselected

The JSON response is displayed as follows:

```
1  {
2      "pageSize": 0,
3      "pageNum": 1,
4      "totalPage": 3,
5      "totalCount": 5,
6      "questionList": [
7          {
8              "sid": 8,
9              "title": "title-aLLx",
10             "createDate": "2020-04-14T09:42:57.000+0000",
11             "creator": 1
12         },
13         {
14             "sid": 9,
15             "title": "title-egzC",
16             "createDate": "2020-04-14T09:42:57.000+0000",
17             "creator": 1
18         }
19     ]
20 }
```

## 直接使用sql语句查询

我们还可以直接通过使用sql语句查询，我们使用Query标注可以直接写sql语句，通过映射返回值的方式获取结果，比如：

```
public interface AnswerRepository extends JpaRepository<QaAnswerEntity, Integer> {  
    @Query(value="select count(*) from qa_answer", nativeQuery = true)  
    Long countAll();
```

直接使用新增加的方法：

```
@PostMapping("/answer/query_sql")  
String queryBySql(@RequestBody Map<String, String> queryExample) {  
    return repository.countAll().toString();  
}
```

<https://docs.spring.io/spring-data/jpa/docs/current/reference/html/#jpa.query-methods.at-query>  
<https://docs.spring.io/spring-data/jpa/docs/current/reference/html/#jpa.query.spel-expressions>

# 多表查询~直接使用SQL语句

我们同样可以使用SQL语句查询来解决复杂的查询或者多表连接的问题

```
public interface AnswerRepository extends JpaRepository<QaAnswerEntity, Integer> {
    @Query(value="select count(*) from qa_answer", nativeQuery = true)
    Long countAll();

    @Query(value="select u.sid as sid, u.loginname as name, count(q.sid) as answerCount " +
        "from qa_answer q, mst_user u where q.creator = u.sid and u.sid = ?1", nativeQuery = true)
    AnswerUserView countAllByUser(int uid);

}
```

其中返回值可以使用Projection机制来获取（注意interface，get方法及上面sql语句返回字段名匹配

```
/**
 * @author pengbin
 * @version 1.0
 * @date 2020-04-14 21:56
 */
public interface AnswerUserView {
    int getSid();
    String getName();
    int getAnswerCount();
}
```

<https://docs.spring.io/spring-data/jpa/docs/current/reference/html/#projections>

# 多表查询~使用对象映射

如果我们创建了外键关系，我们可以使用JPA提供的  
OneToMany、ManyToMany等标注进行操作。可以参考下面  
的文章内容

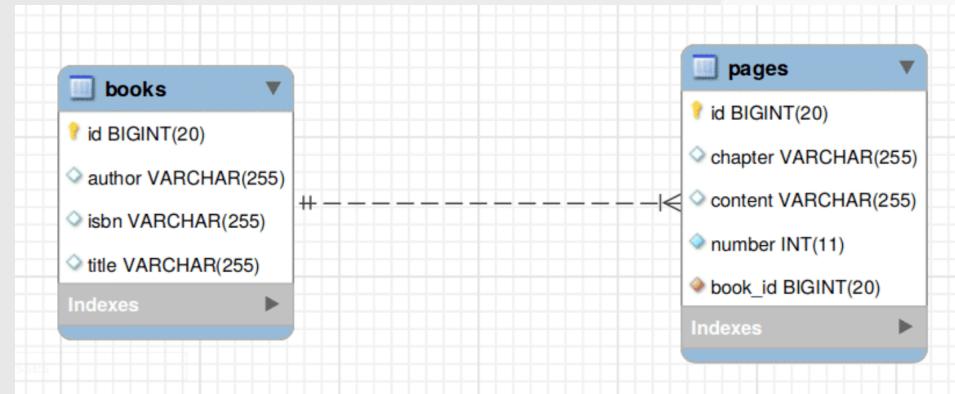
<https://attacomsian.com/blog/spring-data-jpa-one-to-many-mapping>

Page Entity

```
@ManyToOne(fetch = FetchType.LAZY, optional = false)
@JoinColumn(name = "book_id", nullable = false)
private Book book;
```

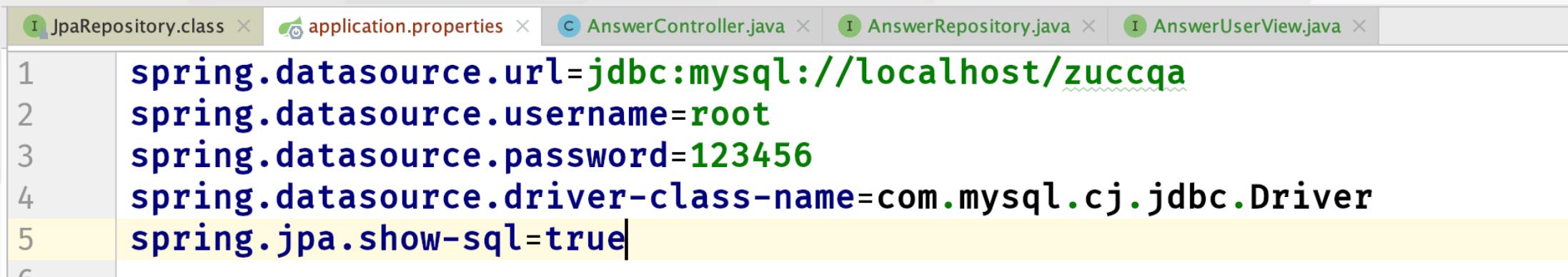
Book Entity

```
@OneToMany(mappedBy = "book", fetch = FetchType.LAZY,
           cascade = CascadeType.ALL)
private Set<Page> pages;
```



# 开启SQL语句日志

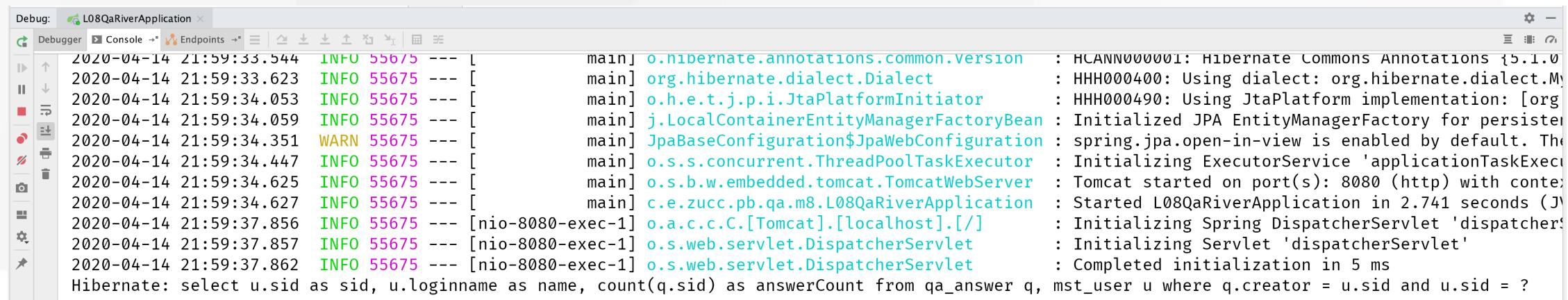
为了方便查看Spring Data JPA发往数据库的日志，我们可以打开SQL语句日志



The screenshot shows a code editor with several tabs at the top: JpaRepository.class, application.properties, AnswerController.java, AnswerRepository.java, and AnswerUserView.java. The application.properties tab is active, displaying the following configuration:

```
1 spring.datasource.url=jdbc:mysql://localhost/zuccqa
2 spring.datasource.username=root
3 spring.datasource.password=123456
4 spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver
5 spring.jpa.show-sql=true
```

打开sql日志后可以在控制台中查看SQL语句日志：



The screenshot shows a terminal window within a Java IDE. The title bar says "Debug: L08QaRiverApplication". The log output is as follows:

```
2020-04-14 21:59:33.544 INFO 55675 --- [           main] o.hibernate.annotations.common.version : HCANN000001: Hibernate Commons Annotations {5.1.0
2020-04-14 21:59:33.623 INFO 55675 --- [           main] org.hibernate.dialect.Dialect       : HHH000400: Using dialect: org.hibernate.dialect.My
2020-04-14 21:59:34.053 INFO 55675 --- [           main] o.h.e.t.j.p.i.JtaPlatformInitiator  : HHH000490: Using JtaPlatform implementation: [org
2020-04-14 21:59:34.059 INFO 55675 --- [           main] j.LocalContainerEntityManagerFactoryBean: Initialized JPA EntityManagerFactory for persiste
2020-04-14 21:59:34.351 WARN 55675 --- [           main] JpaBaseConfiguration$JpaWebConfiguration: spring.jpa.open-in-view is enabled by default. The
2020-04-14 21:59:34.447 INFO 55675 --- [           main] o.s.s.concurrent.ThreadPoolTaskExecutor: Initializing ExecutorService 'applicationTaskExecu
2020-04-14 21:59:34.625 INFO 55675 --- [           main] o.s.b.w.embedded.tomcat.TomcatWebServer: Tomcat started on port(s): 8080 (http) with conte
2020-04-14 21:59:34.627 INFO 55675 --- [           main] c.e.zucc.pb.qa.m8.L08QaRiverApplication: Started L08QaRiverApplication in 2.741 seconds (J
2020-04-14 21:59:37.856 INFO 55675 --- [nio-8080-exec-1] o.a.c.c.C.[Tomcat].[localhost].[]:      : Initializing Spring DispatcherServlet 'dispatcherServlet'
2020-04-14 21:59:37.857 INFO 55675 --- [nio-8080-exec-1] o.s.web.servlet.DispatcherServlet   : Initializing Servlet 'dispatcherServlet'
2020-04-14 21:59:37.862 INFO 55675 --- [nio-8080-exec-1] o.s.web.servlet.DispatcherServlet   : Completed initialization in 5 ms
Hibernate: select u.sid as sid, u.loginname as name, count(q.sid) as answerCount from qa_answer q, mst_user u where q.creator = u.sid and u.sid = ?
```



END

---

Pb&Lois