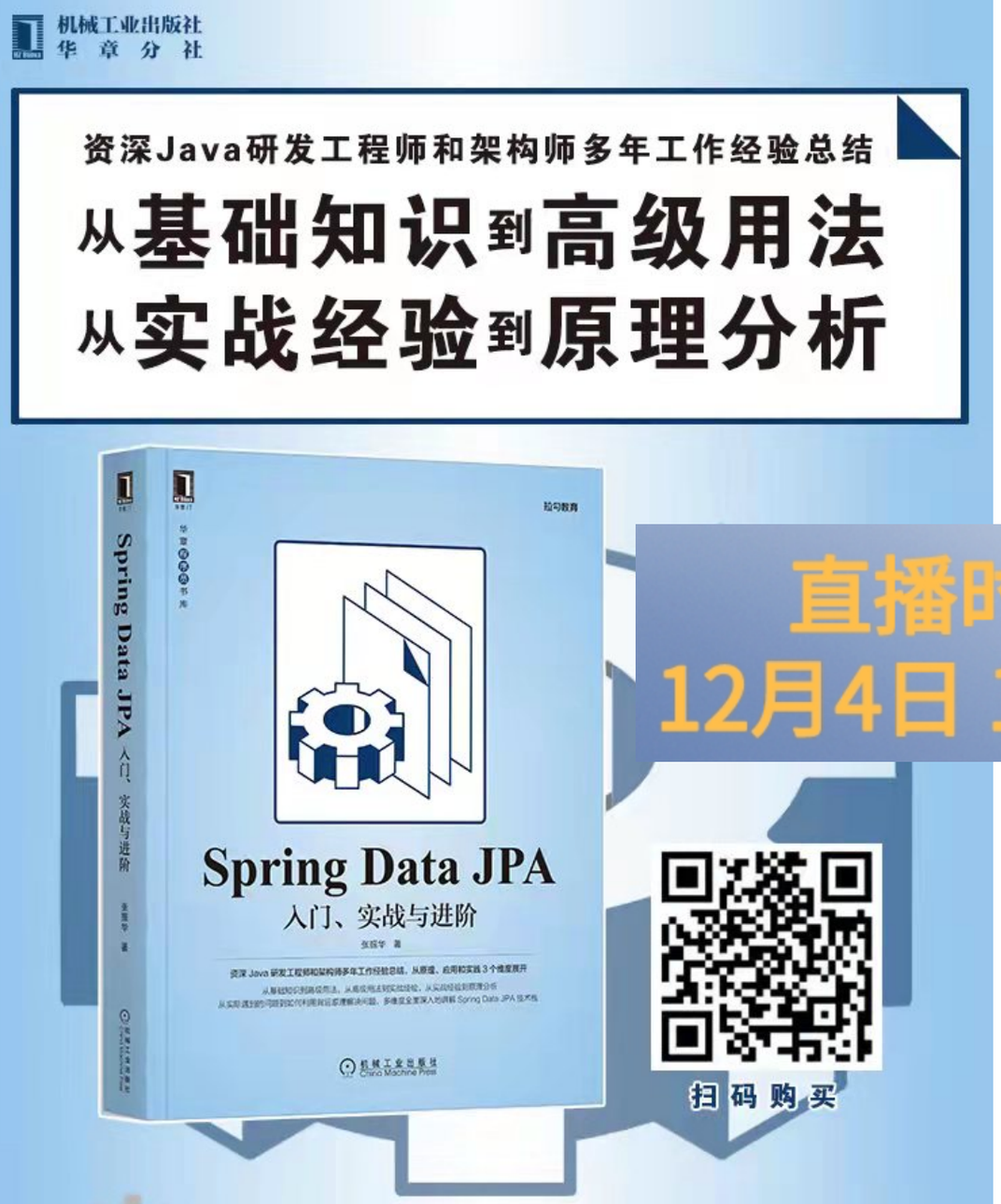
Spring Data JPA的优点和难点



赠一本书，限时(直播结束之前)，朋友圈点赞最先达到50的人，群里发消息，截图 @章鱼丸 第一名者得

https://item.jd.com/13001887.html

Spring Data JPA 交流群

章鱼丸

微信号: star1216zyy

QQ群：559701472

微信：A-zhangzhenhua

https://github.com/zhangzhenhuajack

Spring Data JPA的优点

1. 开发效率极高：

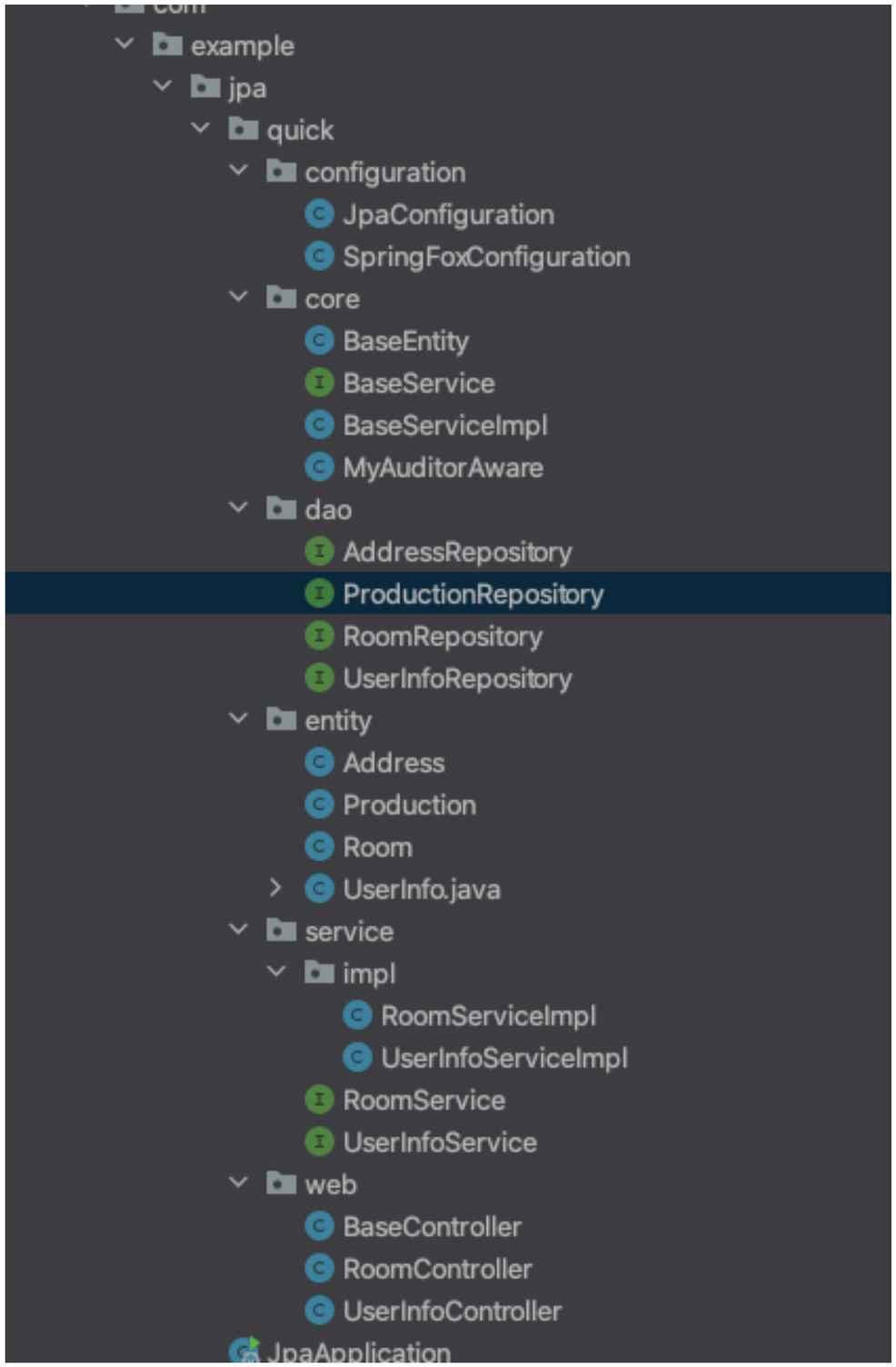
2. 成熟的语法结构：

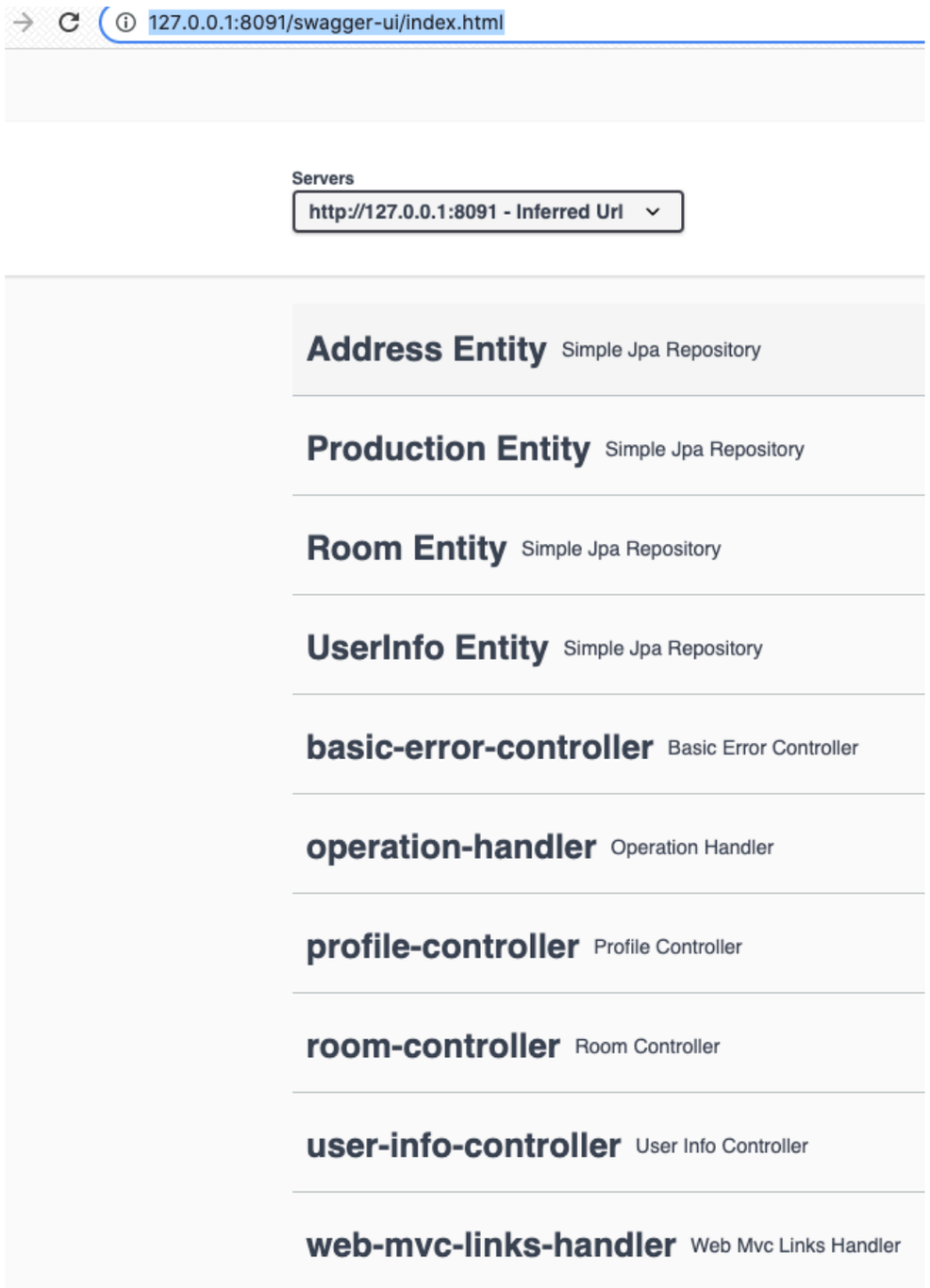
3. 与Spring全家桶结合紧密：

4. 成熟的框架和架构

微信：A-zhangzhenhua
https://github.com/zhangzhenhuajack

Spring Data JPA的优点：开发效率极高　　查看Demo





https://github.com/zhangzhenhuajack/spring-boot-guide/tree/master/spring-data/spring-data-jpa/quick_start

微信：A-zhangzhenhua
https://github.com/zhangzhenhuajack

Spring Data JPA的优点：开发效率极高　　查看Demo

127.0.0.1:8091/h2-console/login.jsp?jsessionid=b218db1c1a41ac59818039c43710e73b

English　Preferences　Tools　Help

Login

Saved Settings: Generic H2 (Embedded)

Setting Name: Generic H2 (Embedded)　Save　Remove

Driver Class: org.h2.Driver

JDBC URL: jdbc:h2:mem:da32dc6f-6db4-4475-9a15-d0b4ec889bef;[

User Name: sa

Password:

Connect　Test Connection

从IDEA的控制台来

127.0.0.1:8091/h2-console/login.do?jsessionid=b218db1c1a41ac59818039c43710e73b

Auto commit　Max rows: 1000　Auto complete Off　Auto select On

jdbc:h2:mem:da32dc6f-6db4-4475-9a15-d0b4ec889bef;DB_CLOSE_DELAY=-1;DB_CLOSE_ON　Run　Run Selected　Auto complete　Clear　SQL statement:

- ADDRESS
  - ID
  - CREATE_TIME
  - CREATE_USER_ID
  - LAST_MODIFIED_TIME
  - LAST_MODIFIED_USER_ID
  - VERSION
  - CITY
  - Indexes
- PRODUCTION
  - ID
  - CREATE_TIME
  - CREATE_USER_ID
  - LAST_MODIFIED_TIME
  - LAST_MODIFIED_USER_ID
  - VERSION
  - TITLE
  - Indexes
- ROOM
  - ID
  - CREATE_TIME
  - CREATE_USER_ID
  - LAST_MODIFIED_TIME
  - LAST_MODIFIED_USER_ID
  - VERSION
  - TITLE
  - USER_ID
  - Indexes
- USER_INFO
  - ID
  - CREATE_TIME
  - CREATE_USER_ID
  - LAST_MODIFIED_TIME
  - LAST_MODIFIED_USER_ID
  - VERSION
  - AGE
  - DELETED
  - EMAIL
  - NAME
  - SEX
  - Indexes

Important Commands

		Displays this Help Page
		Shows the Command History
	Ctrl+Enter	Executes the current SQL statement
	Shift+Enter	Executes the SQL statement defined by the text selection
	Ctrl+Space	Auto complete
		Disconnects from the database

Sample SQL Script

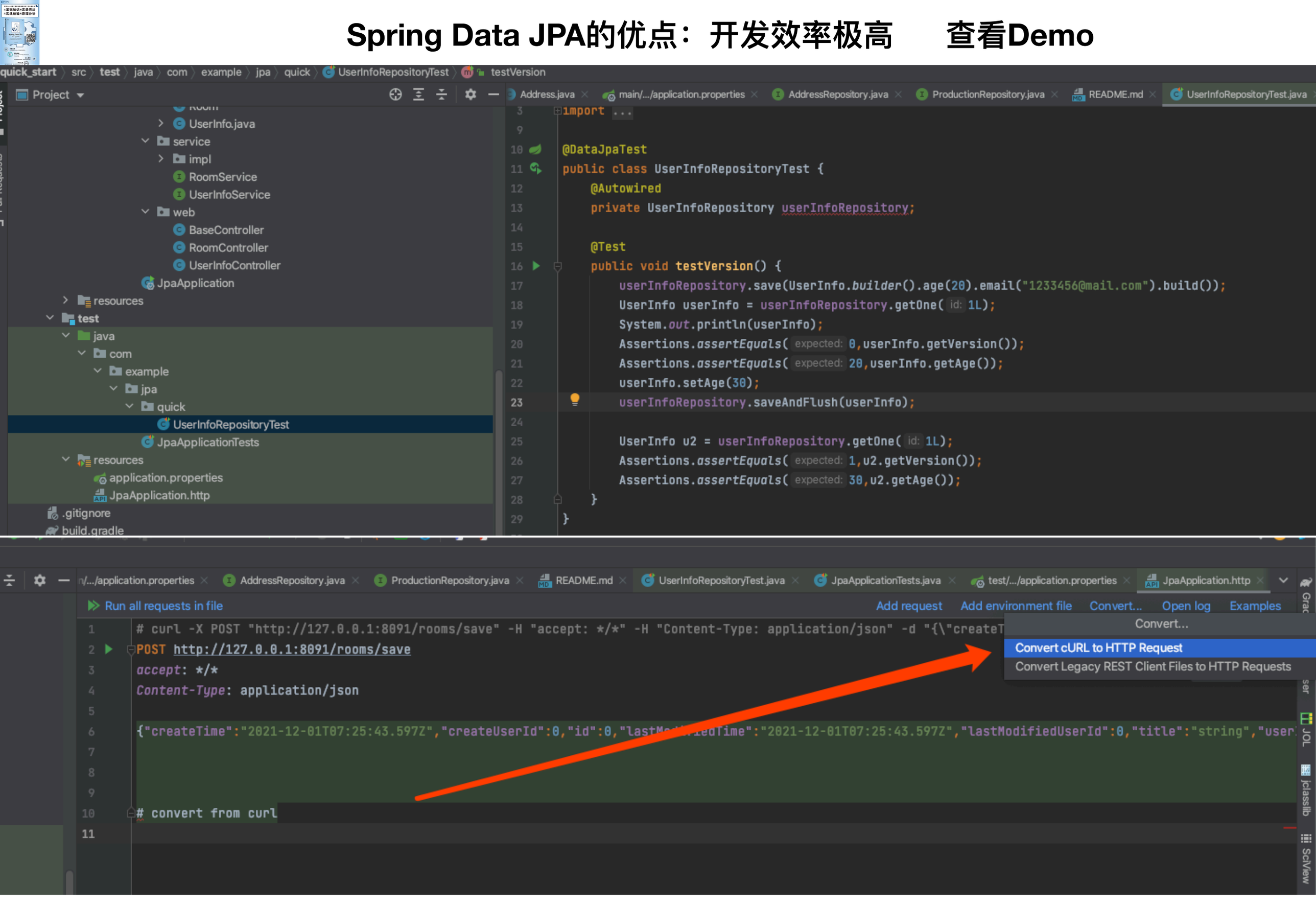
Delete the table if it exists	DROP TABLE IF EXISTS TEST;
Create a new table with ID and NAME columns	CREATE TABLE TEST(ID INT PRIMARY KEY, NAME VARCHAR(255));
Add a new row	INSERT INTO TEST VALUES(1, 'Hello');
Add another row	INSERT INTO TEST VALUES(2, 'World');
Query the table	SELECT * FROM TEST ORDER BY ID;
Change data in a row	UPDATE TEST SET NAME='Hi' WHERE ID=1;
Remove a row	DELETE FROM TEST WHERE ID=2;
Help	HELP ...

Adding Database Drivers

Additional database drivers can be registered by adding the Jar file location of the driver to the env
environment variable H2DRIVERS to C:/Programs/hsqldb/lib/hsqldb.jar.

```
: isolateInternalQueries.........false
: jdbcUrl.........................jdbc:h2:mem:da32dc6f-6db4-4475-9a15-d0b4ec889bef;DB_CLOSE_DELAY=-1;DB_CLOSE_ON_EXIT=FALS
```

Spring Data JPA的优点：开发效率极高　　查看Demo



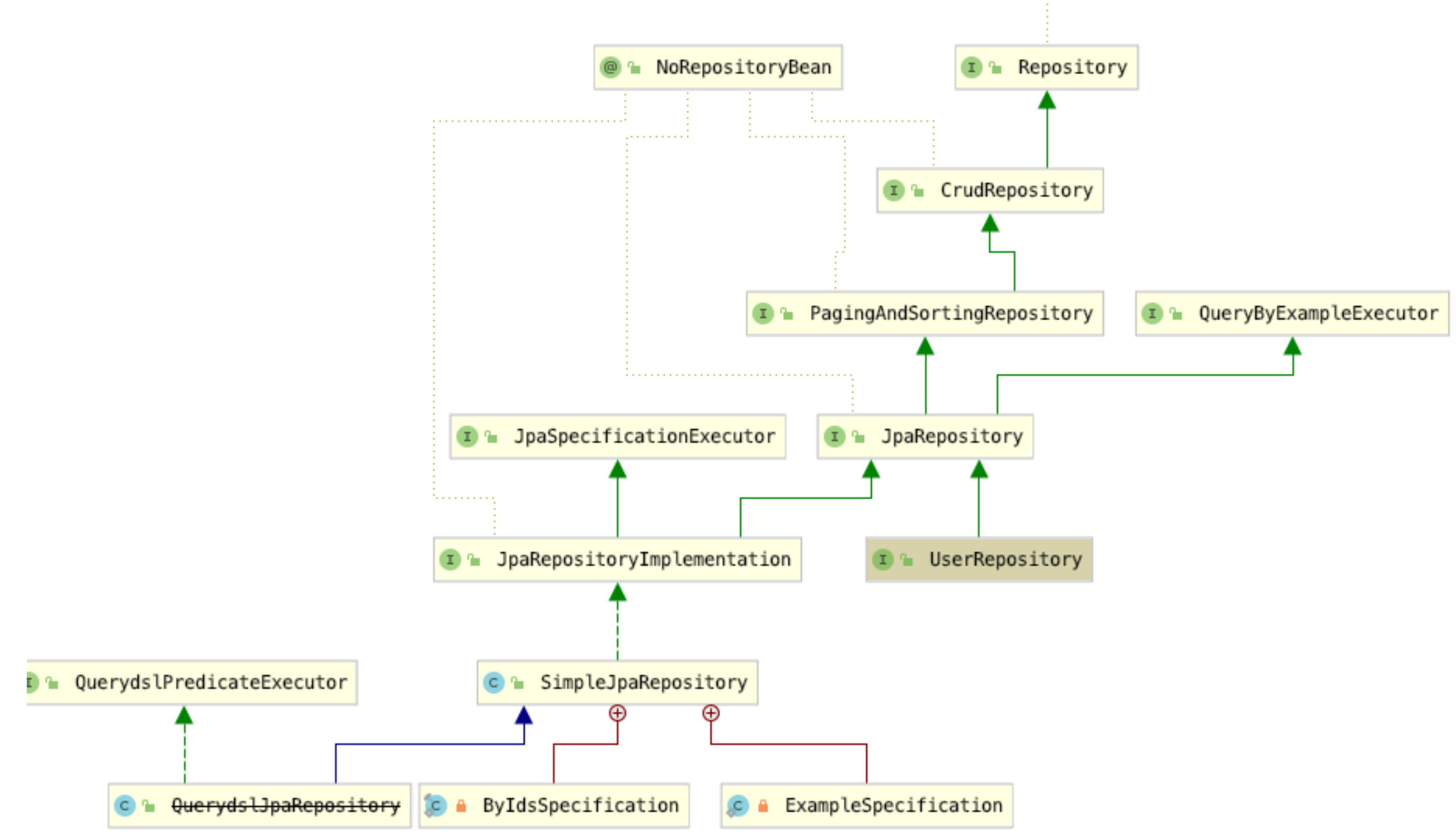
微信：A-zhangzhenhua
https://github.com/zhangzhenhuajack

Spring Data JPA的优点：成熟的语法结构之DQM方法名语法

```
// and 的查询关系
List<User> findByEmailAddressAndLastname(EmailAddress emailAddress, String lastname);
// 包含 distinct 去重，or 的 sql 语法
List<User> findDistinctPeopleByLastnameOrFirstname(String lastname, String firstname);
// 根据 lastname 字段查询忽略大小写
List<User> findByLastnameIgnoreCase(String lastname);
// 根据 lastname 和 firstname 查询 equal 并且忽略大小写
List<User> findByLastnameAndFirstnameAllIgnoreCase(String lastname, String firstname);
// 对查询结果根据 lastname 排序，正序
List<User> findByLastnameOrderByFirstnameAsc(String lastname);
// 对查询结果根据 lastname 排序，倒序
List<User> findByLastnameOrderByFirstnameDesc(String lastname);
```

```
//根据分页参数查询User，返回一个带分页结果的Page
Page<User> findByLastname(String lastname, Pageable pageable);
//我们根据分页参数返回一个Slice的user结果
Slice<User> findByLastname(String lastname, Pageable pageable);
//根据排序结果返回一个List
List<User> findByLastname(String lastname, Sort sort);
//根据分页参数返回一个List对象
List<User> findByLastname(String lastname, Pageable pageable);
//first Top
User findFirstByOrderByLastnameAsc();
User findTopByOrderByAgeDesc();
List<User> findDistinctUserTop3ByLastname(String lastname, Pageable pageable);
List<User> findFirst10ByLastname(String lastname, Sort sort);
List<User> findTop10ByLastname(String lastname, Pageable pageable);
```

Spring Data JPA的优点：成熟的语法结构之常见的CRUD的Repository接口



Spring Data JPA的优点：成熟的语法结构之常见的CRUD的Repository接口

spring-data-commons-2.3.3.RELEASE-sources.jar › org › springframework › data › repository › CrudRepository

Structure

CrudRepository
- count(): long
- delete(T): void
- deleteAll(): void
- deleteAll(Iterable<? extends T>): void
- deleteById(ID): void
- existsById(ID): boolean
- findAll(): Iterable<T>
- findAllById(Iterable<ID>): Iterable<T>
- findById(ID): Optional<T>
- save(S): S
- saveAll(Iterable<S>): Iterable<S>

CrudRepository.java | JpaRepository.java | JpaRepositoryImplementation.java | Simp

```
1   /.../
16  package org.springframework.data.repository;
17
18  import java.util.Optional;
19
20  /**
21   * Interface for generic CRUD operations on a repository for a specific
22   *
23   * @author Oliver Gierke
24   * @author Eberhard Wolff
25   */
26  @NoRepositoryBean
27  public interface CrudRepository<T, ID> extends Repository<T, ID> {
```

Structure

JpaRepository
- CrudRepository
  - findAll(): List<T> ↑CrudRepository
  - findAllById(Iterable<ID>): List<T> ↑CrudRepository
  - saveAll(Iterable<S>): List<S> ↑CrudRepository
- PagingAndSortingRepository
  - findAll(Sort): List<T> ↑PagingAndSortingRepository
- QueryByExampleExecutor
  - findAll(Example<S>): List<S> ↑QueryByExampleExecutor
  - findAll(Example<S>, Sort): List<S> ↑QueryByExampleExecutor
- deleteAllInBatch(): void
- deleteInBatch(Iterable<T>): void
- flush(): void
- getOne(ID): T
- saveAndFlush(S): S

Z: Structure | Persistence | Favorites

Spring Data JPA的优点：成熟的语法结构之优雅返回值

第一种方法：新建一张表的不同 Entity

首先，我们新增一个 Entity 类：通过 @Table 指向同一张表，这张表和 User 实例里面的表一样都是 user，完整内容如下：

```
@Entity
@Table(name = "user")
@Data
@Builder
@AllArgsConstructor
@NoArgsConstructor
public class UserOnlyNameEmailEntity {
    @Id
    @GeneratedValue(strategy= GenerationType.AUTO)
    private Long id;
    private String name;
    private String email;
}
```

Projections 的概念

然后，新增一个 UserOnlyNameEmailEntityRepository，做单独的查询：

```
package com.example.jpa.example1;
import org.springframework.data.jpa.repository.JpaRepository;
public interface UserOnlyNameEmailEntityRepository extends JpaRepository
}
```

第二种方法：直接定义一个 UserOnlyNameEmailDto

首先，我们新建一个 DTO 类来返回我们想要的字段，它是 UserOnlyNameEmailDto，用收 name、email 两个字段的值，具体如下：

```
@Data
@Builder
@AllArgsConstructor
public class UserOnlyNameEmailDto {
    private String name,email;
}
```

Plain T… 自动换行

```
public interface UserRepository extends JpaRepository<User,Long> {
    //测试只返回name和email的DTO
    UserOnlyNameEmailDto findByEmail(String email);
}
```

Slice	返回指定大小的数据和是否还有可用数据的信息。需要方法带有 Pageable 类型的参数
Page	在 Slice 的基础上附加返回分页总数等信息。需要方法带有 Pageable 类型的参数
Optional	返回 Java 8 或 Guava 中的 Optional 类。查询方法的返回结果最多只能有一个，如果超过了一个结果会抛出 IncorrectResultSizeDataAccessException 的异常

Repository 对 Feature/CompletableFuture 异步返回结果的支持：

我们可以使用 Spring 的异步方法执行 Repository 查询，这意味着方法将在调用时立并且实际的查询执行将发生在已提交给 Spring TaskExecutor 的任务中，比较适合定实际场景。异步使用起来比较简单，直接加 @Async 注解即可，如下所示：

```
@Async
Future<User> findByFirstname(String firstname); (1)
@Async
CompletableFuture<User> findOneByFirstname(String firstname); (
@Async
ListenableFuture<User> findOneByLastname(String lastname);(3)
```

Spring Data JPA的优点：与Spring全家桶结合紧密之MVC

```
@GetMapping("/users")
public Page<UserInfo> queryByPage(Pageable pageable, UserInfo userInfo) {
    return userInfoRepository.findAll(Example.of(userInfo),pageable);
}
@GetMapping("/users/sort")
public HttpEntity<List<UserInfo>> queryBySort(Sort sort) {
    return new HttpEntity<>(userInfoRepository.findAll(sort));
}
```

第三步：Controller 里面直接利用 @QuerydslPredicate 注解接收 Predicate predicate 参数。

复制代码

```
@GetMapping(value = "user/dsl")
Page<UserInfo> queryByDsl(@QuerydslPredicate(root = UserInfo.class) com.querydsl
//这里面我用的userInfoRepository里面的QuerydslPredicateExecutor里面的方法
    return userInfoRepository.findAll(predicate, pageable);
}
```

第四步：直接请求我们的 user / dsl 即可，这里利用 queryDsl 的语法 ， 使 &ages=10 作为我们的请求参数。

复制代码

```
GET http://127.0.0.1:8089/user/dsl?size=2&page=0&ages=10&sort=id%2Cdesc&ages=10
Content-Type: application/json
{
  "content": [
    {
      "id": 2,
      "version": 0,
      "ages": 10,
      "telephone": "123456789"
    },
    {
      "id": 1,
      "version": 0,
      "ages": 10,
      "telephone": "123456789"
    }
  ],
  "pageable": {
    "sort": {
      "sorted": true,
```



```
{
  "content": [
    {
      "id": 4,
      "version": 0,
      "ages": 10,
      "telephone": "123456
    },
    {
      "id": 3,
      "version": 0,
      "ages": 10,
      "telephone": "123456
    }
  ],
  "pageable": {
    "sort": {
      "sorted": true,
      "unsorted": false,
      "empty": false
    },
    "offset": 0,
    "pageNumber": 0,
    "pageSize": 2,
    "unpaged": false,
    "paged": true
  },
  "totalPages": 2,
  "totalElements": 4,
  "last": false,
  "size": 2,
  "number": 0,
  "numberOfElements": 2,
  "sort": {
    "sorted": true,
    "unsorted": false,
    "empty": false
  },
```

Spring Data JPA的优点：与Redis的优雅结合

第三步：在用到二级缓存的地方配置 @Cacheable 和 @Cache 的策略。

复制代码

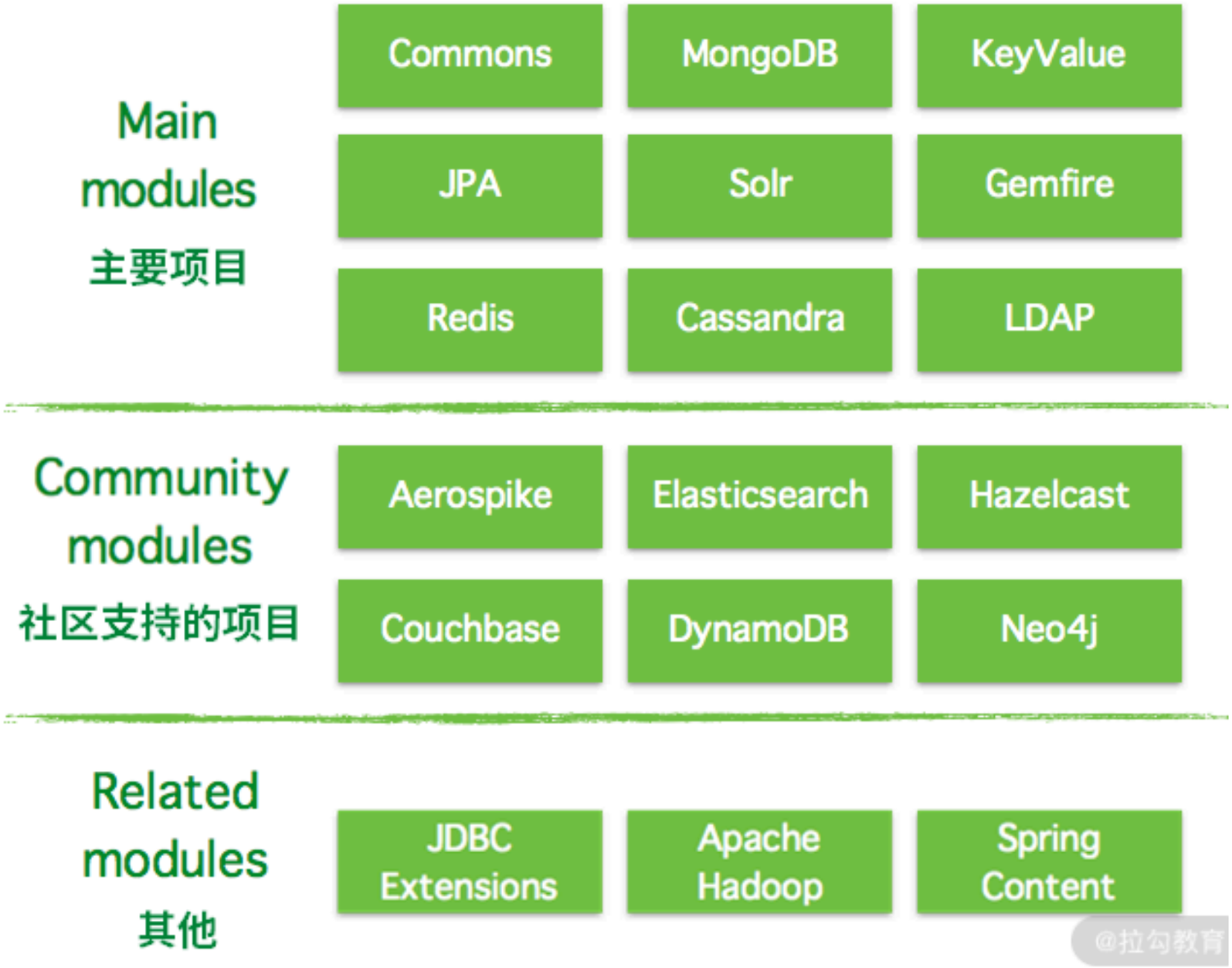
```
1. import javax.persistence.Cacheable;
2. import javax.persistence.Entity;
3. @Entity
4. @Cacheable
5. @org.hibernate.annotations.Cache(usage = CacheConcurrencyStrategy.READ_WRITE)
6. public class UserInfo extends BaseEntity {......}
```

第四步：在我们需要缓存的地方添加 @Cacheable 注解即可。为了方便演示，我把 @Cacheable 注解配置在了 controller 方法上，代码如下。

复制代码

```
1. @GetMapping("/user/info/{id}")
2. @Cacheable(value = "userInfo", key = "{#root.methodName, #id}", unless = "#resul
3. public UserInfo getUserInfo(@PathVariable("id") Long id) {
4.     //第二次就不会再执行这里了
5.     return userInfoRepository.findById(id).get();
6. }
```

Spring Data JPA的优点：Spring Data 抽象易于扩展



Spring Data JPA的优点：Spring Data 抽象易于扩展

第五步：新增一个 ElasticSearchConfiguration 的配置文件，主要是为了开启扫描的包。

复制代码

```
1. package com.example.data.es.demo.es;
2. import org.springframework.context.annotation.Configuration;
3. import org.springframework.data.elasticsearch.repository.config.EnableElasticsea
4. //利用@EnableElasticsearchRepositories注解指定Elasticsearch相关的Repository的包路径在
5. @EnableElasticsearchRepositories(basePackages = "com.example.data.es.demo.es")
6. @Configuration
7. public class ElasticSearchConfiguration {
8. }
```

```
@Data
@Builder
@Document(indexName = "topic")
@ToString(callSuper = true)
//论坛主题信息
public class Topic {
    @Id
    private Long id;
    private String title;
    @Field(type = FieldType.Nested, includeInParent = true)
    private List<Author> authors;
}
package com.example.data.es.demo.es;
import lombok.Builder;
import lombok.Data;
@Data
@Builder
//作者信息
public class Author {
    private String name;
}
```

第七步：新建一个 Elasticsearch 的 Repository，用来对 Elasticsearch 索引的增删改查，代码

所示。

复制代

```
1. package com.example.data.es.demo.es;
2. import org.springframework.data.elasticsearch.repository.ElasticsearchReposi
3. import java.util.List;
4. //类似JPA一样直接操作Topic类型的索引
5. public interface TopicRepository extends ElasticsearchRepository<Topic,Long>
6.     List<Topic> findByTitle(String title);
7. }
```

第八步: 新建一个 Controller，对 Topic 索引进行查询和添加。

复制代

```
1. @RestController
2. public class TopicController {
3.     @Autowired
4.     private TopicRepository topicRepository;
5.     //查询topic的所有索引
6.     @GetMapping("topics")
7.     public List<Topic> query(@Param("title") String title) {
8.         return topicRepository.findByTitle(title);
9.     }
10.    //保存 topic索引
11.    @PostMapping("topics")
12.    public Topic create(@RequestBody Topic topic) {
13.        return topicRepository.save(topic);
14.    }
15. }
```

Spring Data JPA的优点：　　简单容易上手的单元测试

```java
@Entity
@Table
@Data
@SuperBuilder
@AllArgsConstructor
@NoArgsConstructor
public class Address extends BaseEntity {
    private String city;
    private String address;
}
//Repository的DAO层
public interface AddressRepository extends JpaRepository<Address, Long>{

}
```

第三步：新建 RepsitoryTest，@DataJpaTest 即可，代码如下所示。

```java
@DataJpaTest
public class AddressRepositoryTest {
    @Autowired
    private AddressRepository addressRepository;
    //测试一下保存和查询
    @Test
    public  void testSave() {
        Address address = Address.builder().city("shanghai").build();
        addressRepository.save(address);
        List<Address> address1 = addressRepository.findAll();
        address1.stream().forEach(address2 -> System.out.println(address2));
    }
}
```

利用H2数据库
简单快捷的测试
'com.h2database:h2'

Spring Data JPA的优点：成熟的框架和架构 之 审计

@CreatedBy 是哪个用户创建的。

@CreatedDate 创建的时间。

@LastModifiedBy 最后修改实体的用户。

@LastModifiedDate 最后一次修改的时间。

```
@Entity
@Data
@Builder
@AllArgsConstructor
@NoArgsConstructor
@ToString(exclude = "addresses")
@EntityListeners(AuditingEntityListener.class)
public class User implements Serializable {
    @Id
    @GeneratedValue(strategy= GenerationType.AUTO)
    private Long id;
    private String name;
    private String email;
    @Enumerated(EnumType.STRING)
    private SexEnum sex;
    private Integer age;
    @OneToMany(mappedBy = "user")
    @JsonIgnore
    private List<UserAddress> addresses;
    private Boolean deleted;
    @CreatedBy
    private Integer createUserId;
    @CreatedDate
    private Date createTime;
    @LastModifiedBy
    private Integer lastModifiedUserId;
    @LastModifiedDate
    private Date lastModifiedTime;
}
```

Spring Data JPA的优点：成熟的框架和架构 之 成熟的乐观锁和重试机制

什么是乐观锁?

乐观锁在实际开发过程中很常用，它没有加锁、没有阻塞，在多线程环境以及高并发的情况下 CPU 的利用率是最高的，吞吐量也是最大的。

而 Java Persistence API 协议也对乐观锁的操作做了规定：通过指定 @Version 字段对数据增加版本号控制，进而在更新的时候判断版本号是否有变化。如果没有变化就直接更新；如果有变化，就会更新失败并抛出“OptimisticLockException”异常。我们用 SQL 表示一下乐观锁的做法，代码如下：

复制代码

```
1. select uid,name,version from user where id=1;
2. update user set name='jack', version=version+1 where id=1 and version=1
```

假设本次查询的 version=1，在更新操作时，加上这次查出来的 Version，这样和我们上一个版本相同，就会更新成功，并且不会出现互相覆盖的问题，保证了数据的原子性。

```
@Data
@MappedSuperclass
public class BaseEntity {
   @Id
   @GeneratedValue(strategy= GenerationType.AUTO)
   private Long id;
   @Version
   private Integer version;
   //......当然也可以用上一课时讲解的 auditing 字段，这里我们先省略
}
```

Spring Data JPA的优点：成熟的框架和架构 之 成熟的乐观锁和重试机制

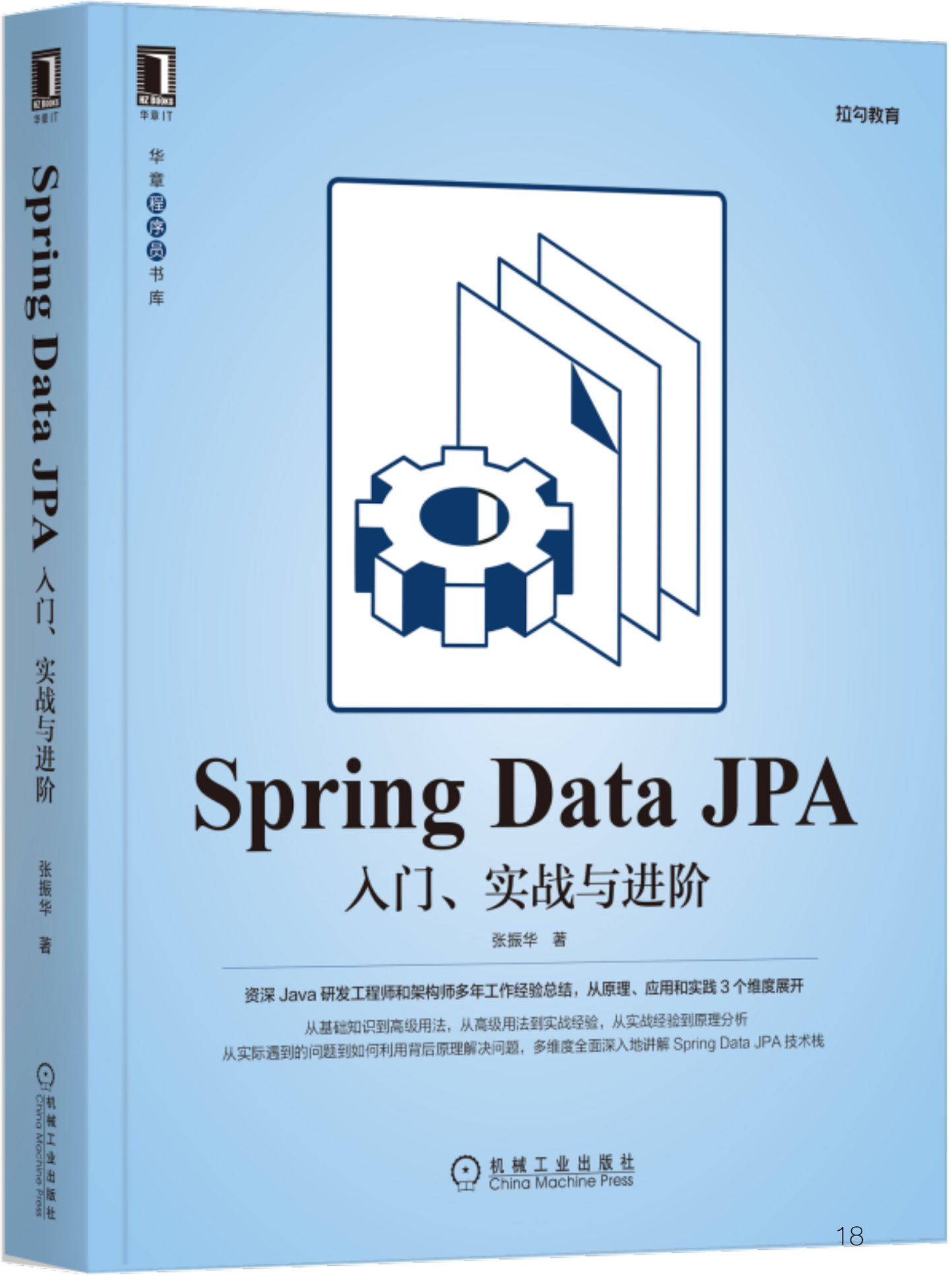
```
1. implementation 'org.springframework.retry:spring-retry'
```

第二步：在 UserInfoserviceImpl 的方法中添加 @Retryable 注解，就可以实现重试的机制了，代码如下：

@Retryable(value = ObjectOptimisticLockingFailureException.class,backoff = @Backoff(multiplier = 1.5,random = true))

ava | UserInfoServiceTest.java | UserInfoServiceImpl.java | UserInfoServiceRetryTest.java

```
14      @Autowired
15      private UserInfoRepository userInfoRepository;
16
17      /**
18       * 根据UserId产生的一些业务计算逻辑
19       *
20       * @param userId
21       * @return
22       */
23      @Override
24      @Transactional
25      @Retryable
26      public UserInfo calculate(Long userId) {
27          UserInfo userInfo = userInfoRepository.getOne(userId);
28          try {
29              //模拟复杂的业务计算逻辑耗时操作;
30              Thread.sleep( millis: 500);
31          } catch (InterruptedException e) {
32              e.printStackTrace();
33          }
34          userInfo.setAges(userInfo.getAges()+1);
35          userInfo.setTelephone(Instant.now().toString());
36          return userInfoRepository.saveAndFlush(userInfo);
37      }
38  }
```



提问送书环节

提问，并被老师选中回答问题者

微信：A-zhangzhenhua

https://github.com/zhangzhenhuajack

Spring Data JPA的难点

1. 常见的SQL性能问题，如何优雅处理?

2. 错综复杂的关联关系如何应对?

3. LazyException本质是什么?

4. 高并发高性能要求的API服务要用JPA吗?

Spring Data JPA的难点：常见的SQL性能问题，如何优雅处理

<Filter criteria>

	id	create_time	create_user_id	last_modified_time	last_modified_user_id	version
1	3	2020-11-29 07:59:19.387000	1	2020-11-29 07:59:19.387000	1	0
2	6	2020-11-29 07:59:28.687000	1	2020-11-29 07:59:28.687000	1	0
3	9	2020-11-29 07:59:31.798000	1	2020-11-29 07:59:31.798000	1	0

其中，每个 UserInfo 分别有两条 Address 数据，也就是一共 6 条 Address 的数据，如下图所示。

test.address [@localhost] | test.user_info [@localhost]

6 rows | Tx: Auto | DDL | Tab-se...d (TSV) | test.address

<Filter criteria>

	create_user_id	last_modified_time	last_modified_user_id	version	city	user_info_id
1	1	2020-11-29 07:59:19.790000	1	0	shanghai	3
2	1	2020-11-29 07:59:20.053000	1	0	jiangshu	3
3	1	2020-11-29 07:59:29.086000	1	0	shanghai	6
4	1	2020-11-29 07:59:29.419000	1	0	jiangshu	6
5	1	2020-11-29 07:59:32.179000	1	0	shanghai	9
6	1	2020-11-29 07:59:32.460000	1	0	jiangshu	9

```
userInfoRepository.findAll()
```

```
org.hibernate.SQL                          :
select userinfo0_.id                       as id1_1_,
       userinfo0_.create_time              as create_t2_1_,
       userinfo0_.create_user_id           as create_u3_1_,
       userinfo0_.last_modified_time       as last_mod4_1_,
       userinfo0_.last_modified_user_id    as last_mod5_1_,
       userinfo0_.version                  as version6_1_,
       userinfo0_.ages                     as ages7_1_,
       userinfo0_.email_address            as email_ad8_1_,
       userinfo0_.last_name                as last_nam9_1_,
       userinfo0_.name                     as name10_1_,
       userinfo0_.telephone                as telepho11_1_
from user_info userinfo0_ org.hibernate.SQL   ← 1条User   :
select addresslis0_.user_info_id           as user_inf8_0_0_,
       addresslis0_.id                     as id1_0_0_,
       addresslis0_.id                     as id1_0_1_,
       addresslis0_.create_time            as create_t2_0_1_,
       addresslis0_.create_user_id         as create_u3_0_1_,
       addresslis0_.last_modified_time     as last_mod4_0_1_,
       addresslis0_.last_modified_user_id  as last_mod5_0_1_,
       addresslis0_.version                as version6_0_1_,
       addresslis0_.city                   as city7_0_1_,
       addresslis0_.user_info_id           as user_inf8_0_1_
from address addresslis0_   ← N条address SQL
where addresslis0_.user_info_id = ? org.hibernate.SQL
select addresslis0_.user_info_id           as user_inf8_0_0_,
       addresslis0_.id                     as id1_0_0_,
       addresslis0_.id                     as id1_0_1_,
       addresslis0_.create_time            as create_t2_0_1_,
       addresslis0_.create_user_id         as create_u3_0_1_,
       addresslis0_.last_modified_time     as last_mod4_0_1_,
       addresslis0_.last_modified_user_id  as last_mod5_0_1_,
       addresslis0_.version                as version6_0_1_,
       addresslis0_.city                   as city7_0_1_,
```

Spring Data JPA的难点：常见的SQL性能问题，如何优雅处理

转化解决问题的思路

这时需要我们在思想上进行转变，利用 JPA 的优势，摒弃它的缺陷。想想我们没有用 JPA 的时候是怎么做的？难道一定要用实体之间的关联关系吗？如果用的是 Mybatis，你在给前端返回关联关系数据的时候一般怎么写呢？

@EntityGraph 使用详解

众所周知，实体与实体之间的关联关系错综复杂，就像一个大网图一样，网状分布交叉引用。而 JPA 协议在 2.1 版本之后企图用 Entity Graph 的方式，描绘出一个实体与实体之间的关联关系。

普通做法为，通过 @ManyToOne/@OneToMany/@ManyToMany/@OneToOne 这些关联关系注解表示它们之间的关系时，只能配置 EAGER 或者 LAZY，没办法根据不同的配置、不同的关联关系加载时机。

而 JPA 协议企图通过 @NamedEntityGraph 注解来描述实体之间的关联关系，当被 @EntityGraph 使用的时候进行 EAGER 加载，以减少 N+1 的 SQL，我们来看一下具体用法。

@NamedEntityGraph 和 @EntityGraph 用法

Spring Data JPA的难点：错综复杂的关联关系如何应对?

一定要用关联关系吗?
Mybatis里面如何实现的?

```
/**
 * 分页获取课程当前老师的 roomRecord
 */
@Query(value = "select * from room_record rrr inner join " +
        " (select distinct rr.id as id from room_record rr left join room r on rr.room_id = r.id " +
        " where rr.host_user_id = r.host_id " +
        " and (:roomUuid is null or r.uuid = :roomUuid) " +
        " and (:material is null or r.material = :material) " +
        " and (:scheduledTimeStart is null or r.scheduled_time >= :scheduledTimeStart)" +
        " and (:scheduledTimeEnd is null or r.scheduled_time <= :scheduledTimeEnd)" +
        " and (:headquarterCode is null or r.headquarter_code = :headquarterCode) " +
        " ) as temp " +
        " on rrr.id = temp.id ", nativeQuery = true)
Page<RoomRecord> findByRoomForCurrentHost(@Param("roomUuid") String roomUuid,
                                          @Param("material") String material,
                                          @Param("headquarterCode") String headquarterCode,
                                          @Param("scheduledTimeStart") Instant scheduledTimeStart,
                                          @Param("scheduledTimeEnd") Instant scheduledTimeEnd,
                                          Pageable pageable);
```

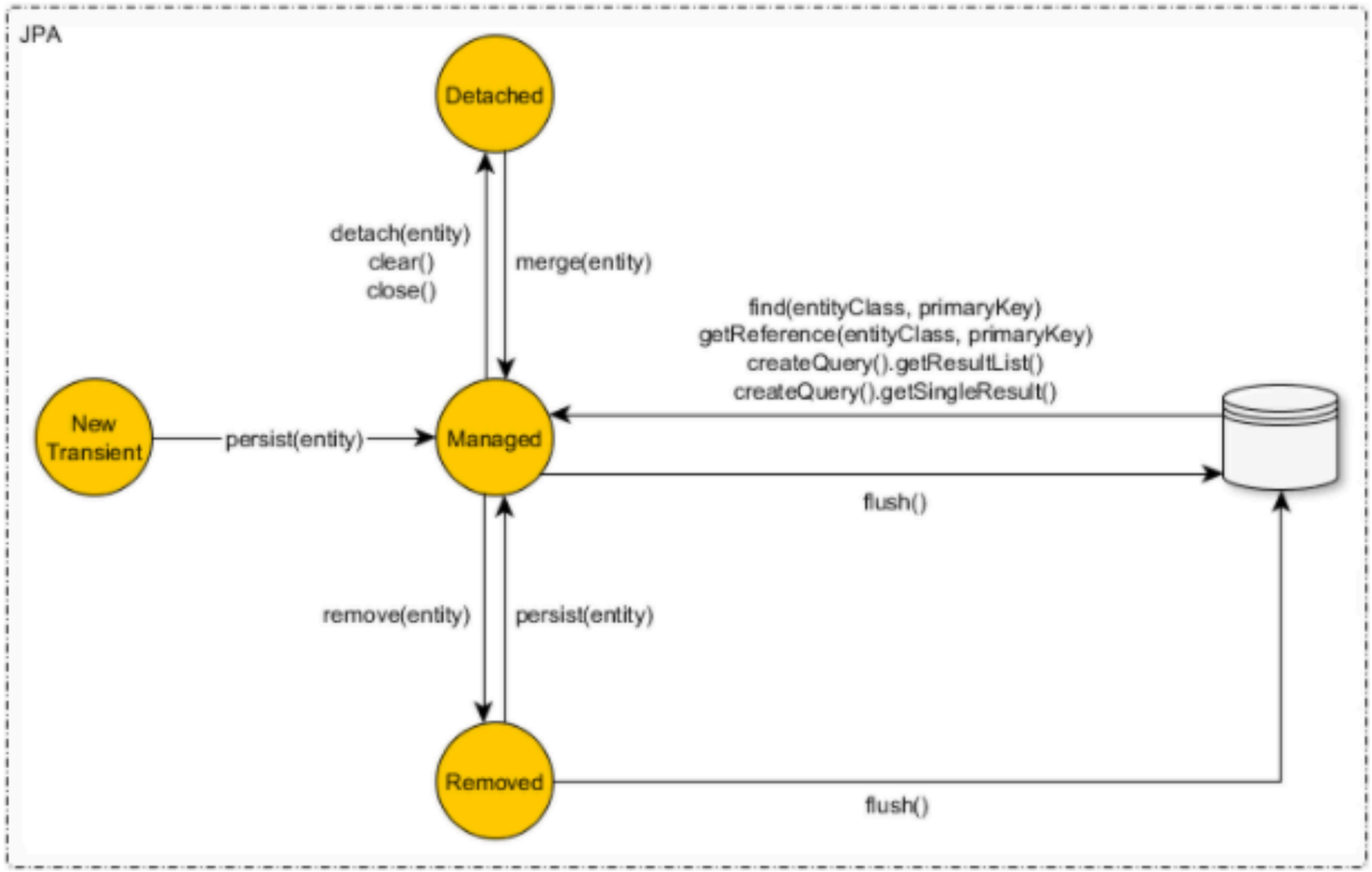
Spring Data JPA的难点：PersistenceContext是什么?

下面我们看一下 PersistenceContext 是怎么创建的。直接打开 SessionImpl 的构造方法，就可以知道 PersistenceContext 是和 Session 的生命周期绑定的，关键代码如下：

```
//session实例初始化的入口
public SessionImpl(SessionFactoryImpl factory, SessionCreationOptions op
    super( factory, options );
    //Session里面创建了persistenceContext，每次session都是新对象
    this.persistenceContext = createPersistenceContext();
.....省略一些不重要的代码
protected StatefulPersistenceContext createPersistenceContext() {
    return new StatefulPersistenceContext( this );
}
//StatefulPersistenceContext就是PersistenceContext的实现类
public class StatefulPersistenceContext implements PersistenceContext {.
```

Spring Data JPA的难点：PersistenceContext是什么?

既然 PersistenceContext 是存储 Entity 的，那么 Entity 在 PersistenceContext 里面肯定有不同的状态。对此，JPA 协议定义了四种状态：new、manager、detached、removed。我们通过一个图来整体认识一下。



Spring Data JPA的难点：搞清楚事务、Connection、Session之间的关系

MySQL 事务与连接的关系

我们要搞清楚事务和连接池的关系，必须要先知道二者存在的前提条件。

1. 事务必须在同一个连接里面的，离开连接没有事务可言；
2. MySQL 数据库默认 autocommit=1，即每一条 SQL 执行完自动提交事务；
3. 数据库里面的每一条 SQL 执行的时候必须有事务环境；
4. MySQL 创建连接的时候默认开启事务，关闭连接的时候如果存在事务没有 commit 的情况，则自动执行 rollback 操作；
5. 不同的 connect 之间的事务是相互隔离的。

Flush 与事务 Commit 的关系

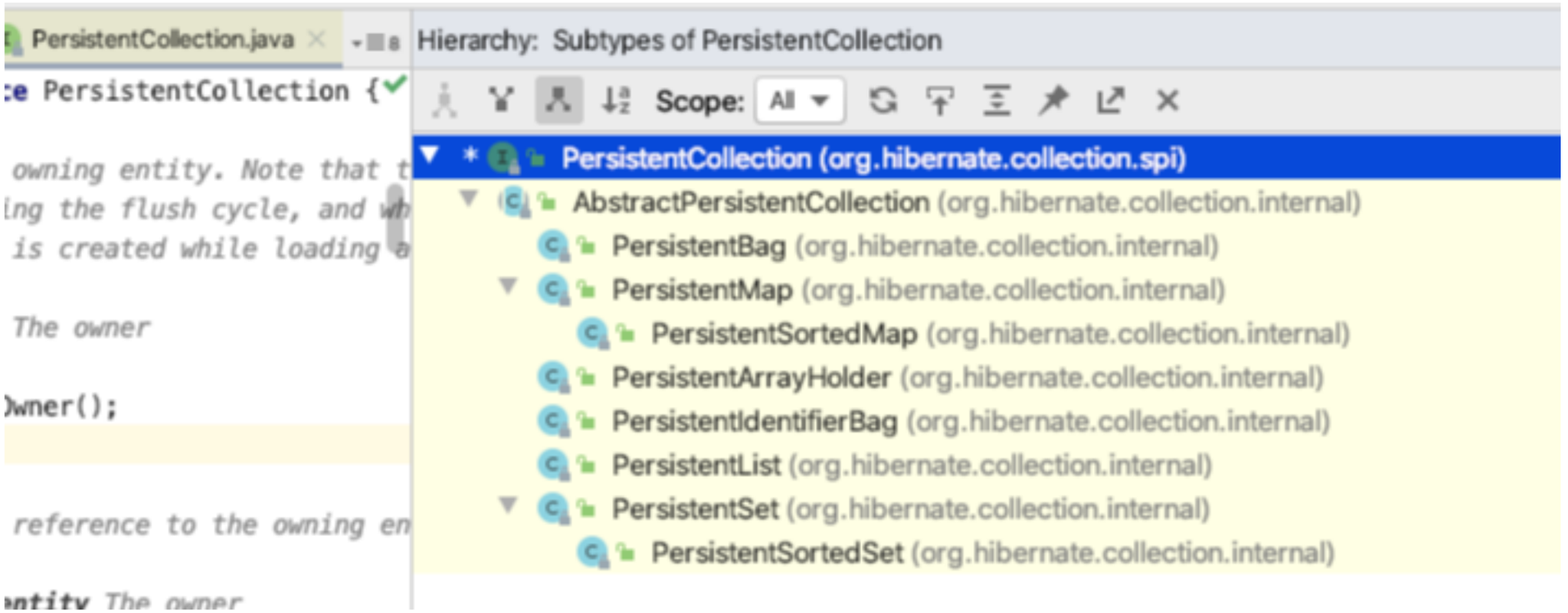
大概有以下几点：

1. 在当前的事务执行 commit 的时候，会触发 flush 方法；
2. 在当前的事务执行完 commit 的时候，如果隔离级别是可重复读的话，flush 之后执行的 update、insert、delete 的操作，会被其他的新事务看到最新结果；
3. 假设当前的事务是可重复读的，当我们手动执行 flush 方法之后，没有执行事务 commit 方法，那么其他事务是看不到最新值变化的，但是最新值变化对当前没有 commit 的事务是有效的；
4. 如果执行了 flush 之后，当前事务发生了 rollback 操作，那么数据将会被回滚（数据库的机制）。

Spring Data JPA的难点：经典flush到时机

1. 事务 commit 之前，即指执行 transactionManager.commit() 之前都会触发；
2. 执行任何的 JPQL 或者 native SQL（代替直接操作 Entity 的方法）都会触发 flush。

Spring Data JPA的难点：LazyException本质是什么?



Lazy 异常的常见场景与解决方法

场景一：跨事务，事务之外的场景

场景二：异步线程的时候

场景三：Controller 直接返回实体也会产生 Lazy Exception

场景四：自定义的拦截器和 filter 中无意的 toString 操作

Spring Data JPA的难点：高并发高性能要求的API服务要用JPA吗?

为什么不用呢?

QQ群：559701472

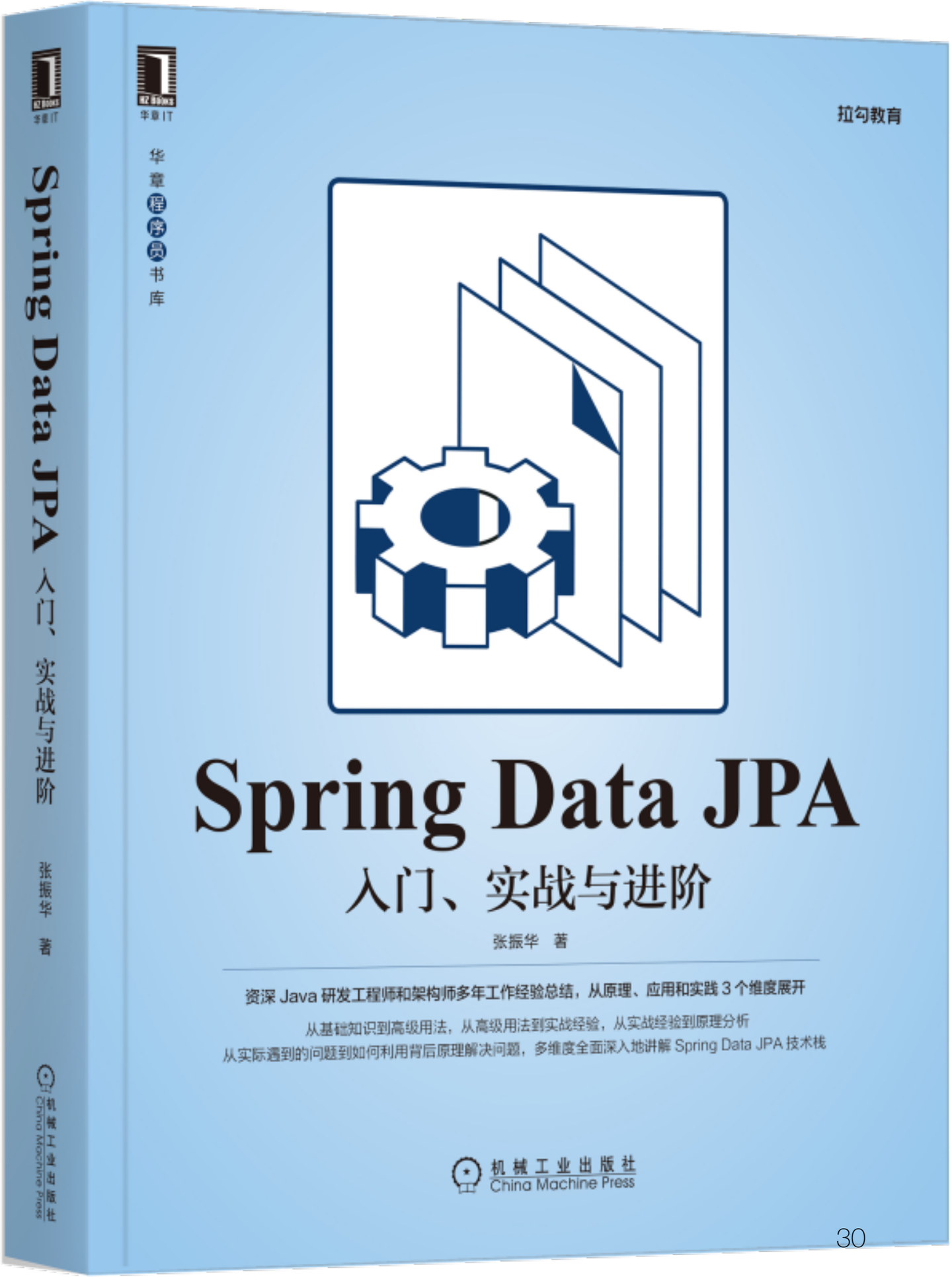
新书目录介绍一下

https://item.jd.com/13001887.html

Spring Data JPA交流群

微信：A-zhangzhenhua

https://github.com/zhangzhenhuajack



抽奖送书环节

被选中问题者，赠送一本书

微信：A-zhangzhenhua

https://github.com/zhangzhenhuajack