异步

异步查询

例 getUserInfoAll 一种典型的异步任务管理模式

启动多个异步任务

- 使用 CompletableFuture.runAsync(() -> { }) 可以在异步线程池中运行一段代码（即一个无返回值的任务）。
- 每次调用都会返回一个 CompletableFuture<Void>，表示该任务的异步执行状态。

b. 将任务加入列表

- add() 方法将每个启动的异步任务（即 CompletableFuture<Void>）添加到 List<CompletableFuture<Void>> 中。
- 这个列表可以用来管理所有异步任务的生命周期。

c. 等待所有任务完成

- 使用 CompletableFuture.allOf() 方法可以统一等待列表中的所有任务完成。
- 它是常见的"批量任务管理"模式，尤其适用于需要并发处理多个操作的场景。

Java 复制代码

```java
CompletableFuture<Void> f = CompletableFuture.allOf(allUserFuture.toArray(new CompletableFuture[0]));
```

- **创建任务列表**
- 使用 List<CompletableFuture<Void>> futures 来存储所有异步任务的 CompletableFuture 实例。
- **启动异步任务**
- CompletableFuture.runAsync(() -> { ... }) 启动一个异步任务，每个任务在独立的线程中运行。
- 每个任务的 CompletableFuture 实例通过 add() 方法添加到 futures 列表中。
- **管理所有任务**
- CompletableFuture.allOf(futures.toArray(new CompletableFuture[0])) 将多个 CompletableFuture 聚合成一个新的 CompletableFuture，用于等待所有任务完成。
- **同步等待**
- allOf.join() 阻塞主线程，直到所有任务完成，确保主程序不会提前结束。
- **任务完成后**
- 所有任务完成后，主程序继续执行并打印 All tasks are completed!。

Lambda 表达式只能用于**函数式接口**。函数式接口是指只包含一个抽象方法的接口。

非常适合completablefuture

Java 复制代码

```java
CompletableFuture.runAsync(new Runnable() {
    @Override
    public void run() {
        System.out.println("Running in an asynchronous task");
    }
});

```