一，前置知识

1，用户线程和守护线程

1.所有用户线程执行完毕，程序就会退出，不再等待守护线程。

```
    public static void main(String[] args) {
      new Thread(()-> System.out.println("当前线
程"+Thread.currentThread().getName()+"是"+(Thread.currentThread().isDaemon()?"守护
线程":"用户线程"))).start();
    }
```

2.用户线程不结束，程序就不会退出

```
public class DemoA {

    public static void main(String[] args) {
        Thread thread = new Thread(DemoA::run,"线程1");
        thread.start();
    }

    private static void run() {
        System.out.println("当前线程" + Thread.currentThread().getName() + "是" +
(Thread.currentThread().isDaemon() ? "守护线程" : "用户线程"));
        while (true) ;
    }
}
```

3.不管守护线程结束没有，只要用户线程结束，程序就会退出

```
public class DemoA {

    public static void main(String[] args) {
        Thread thread = new Thread(DemoA::run,"线程1");
        //将线程1设置为守护线程
        thread.setDaemon(true);
        thread.start();
    }

    private static void run() {
        System.out.println("当前线程" + Thread.currentThread().getName() + "是" +
(Thread.currentThread().isDaemon() ? "守护线程" : "用户线程"));
        while (true) ;
    }
}
```

2.回首FutureTask

1.FutureTask存在的问题

```
public class Demob {

    public static void main(String[] args)throws Exception {
        FutureTask<String> futureTask = new FutureTask<>(Demob::call);
        new Thread(futureTask,"t1").start();
        //异步结果集的展现 ---- 会阻塞
        //System.out.println(futureTask.get());
        //轮询的方式去问 --- 消耗cpu资源
        while (true) {
            if (futureTask.isDone()){
                System.out.println(futureTask.get());
                break;
            }
        }

        System.out.println("main结束");
    }

    private static String call() {
        try {
            TimeUnit.SECONDS.sleep(1);
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
        return "yhd";
    }
}
```

从上面的代码可以看出FutureTask存在的问题：
1.get()异步结果集的展现 ---- 会阻塞
2.futureTask.isDone()轮询的方式去问 --- 消耗cpu资源

2.想完成一些复杂的任务

1.应对Future的完成时间，完成了可以告诉我，也就是我们的回调通知。
2.将两个异步计算合成一个异步计算，这两个异步计算互相独立，同时第二个又依赖第一个的结果。
3.当Future集合中某个任务最快结束时，返回结果。
4.等待Future集合中的所有任务都完成。

二，CompletableFuture

1.介绍

异步函数式编程，Future接口的扩展与增强版。
可以通过回调的方式处理计算结果，并且提供了转换和组合CompletableFuture的方法。
1.完成通知回调
2.类似Linux管道符的形式，可以异步的计算，上一个计算结果可以给下一个，结合lambda表达式和函数式编程串起来转换和组合获得结果。

2.实例化

如果不指定线程池，默认的线程都是守护线程

```
public class DemoC {

    private static ThreadPoolExecutor pool=new ThreadPoolExecutor(1,5,1,
TimeUnit.SECONDS,new ArrayBlockingQueue<>(1));

    public static void main(String[] args) throws Exception{
        //创建一个有返回结果的实例
        CompletableFuture<String> supplyAsync = CompletableFuture.supplyAsync(()
-> "yhd", pool);
        //创建一个没有返回结果的实例
        CompletableFuture<Void> runAsync =
CompletableFuture.runAsync(System.out::println);
        pool.shutdown();
    }
}
```

3.异步+回调

whenComplete 和 join 的区别：
一个时执行完返回结果，一个是主动获取结果

whenComplete：是执行当前任务的线程执行继续执行 whenComplete 的任务。
whenCompleteAsync：是执行把 whenCompleteAsync 这个任务继续提交给线程池来进行执行。
方法不以Async结尾，意味着Action使用相同的线程执行，而Async可能会使用其他线程执行（如果是使用相同的线程池，也可能会被同一个线程选中执行）

```
public class DemoC {

    private static ThreadPoolExecutor pool=new ThreadPoolExecutor(1,5,1,
TimeUnit.SECONDS,new ArrayBlockingQueue<>(1));

    public static void main(String[] args) throws Exception{
        //创建一个有返回结果的实例  计算完成时回调   发生异常时执行
        CompletableFuture.supplyAsync(() -> "yhd",
pool).whenComplete(DemoC::accept).exceptionally(Throwable::toString);
        pool.shutdown();
    }

    private static void accept(String result, Throwable e) {
        System.out.println(result);
    }
}
```

4.结果处理

handle 是执行任务完成时对结果的处理。
handle 是在任务完成后再执行，还可以处理异常的任务。

```
/**
 * @author yhd
 * @createtime 2020/11/15 23:10
 */
public class DemoB {

    public static void main(String[] args) {
```

```
        CompletableFuture.supplyAsync(()->"尹会
东").handle(DemoB::apply).whenCompleteAsync(((s, throwable) ->
System.out.println(s)));
    }

    private static String apply(String s, Throwable throwable) {
        if (null != throwable)
            return throwable.getMessage();
        return s + " 牛逼！";
    }
}
```

5.线程串行化

thenApply 方法：当一个线程依赖另一个线程时，获取上一个任务返回的结果，并返回当前任务的返回值。
thenAccept方法：消费处理结果。接收任务的处理结果，并消费处理，无返回结果。
thenRun方法：只要上面的任务执行完成，就开始执行thenRun，只是处理完任务后，执行 thenRun的后续操作
带有Async默认是异步执行的。这里所谓的异步指的是不在当前线程内执行。

```
public class DemoE {

    public static void main(String[] args) {
        CompletableFuture<Integer> thenApply = CompletableFuture.supplyAsync(()
-> 1).thenApply(integer -> 1);
        CompletableFuture<Void> thenAccept = CompletableFuture.supplyAsync(() ->
1).thenAccept(System.out::println);
        CompletableFuture<Void> thenRun = CompletableFuture.supplyAsync(() ->
1).thenRun(System.out::println);
        System.out.println(thenApply.join());
    }
}
```

6.任务组合

1.两任务组合 - 都要完成

两个任务必须都完成，触发该任务。

thenCombine：组合两个future，获取两个future任务的返回结果，并返回当前任务的返回值
thenAcceptBoth：组合两个future，获取两个future任务的返回结果，然后处理任务，没有返回值。
runAfterBoth：组合两个future，不需要获取future的结果，只需两个future处理完任务后，处理该任务。

```
public class DemoF {

    public static void main(String[] args) {
        CompletableFuture
                .supplyAsync(() -> "hello")
                .thenApplyAsync(t -> t + " world!")
                .thenCombineAsync(
                        CompletableFuture
                                .completedFuture(" CompletableFuture"), (t, u) -
> t + u)
```

```
                .whenComplete(DemoF::accept);
    }

    private static void accept(String t, Throwable u) {
        System.out.println(t);
    }
}
```

2.两任务组合 - 一个完成

当两个任务中，任意一个future任务完成的时候，执行任务。
applyToEither：两个任务有一个执行完成，获取它的返回值，处理任务并有新的返回值。
acceptEither：两个任务有一个执行完成，获取它的返回值，处理任务，没有新的返回值。
runAfterEither：两个任务有一个执行完成，不需要获取future的结果，处理任务，也没有返回值

3.多任务组合

allOf：等待所有任务完成
anyOf：只要有一个任务完成

7.计算接口性能

1.一淘

```
/**
 * @author yhd
 * @createtime 2020/11/15 15:11
 * 查询多个电商网站同一个商品的价格
 * 同步和异步两种方式
 */
public class DemoD {

    static List<Gmall> gmalls= Arrays.asList(
                new Gmall("京东"),
                new Gmall("拼多多"),
                new Gmall("淘宝"),
                new Gmall("唯品会"),
                new Gmall("分期乐"));


    /**
     * 同步
     * @param gmalls
     * @param productName
     * @return
     */
    public static List<String> getPriceSync(List<Gmall> gmalls,String
productName){
        return  gmalls.stream().map(gmall->String.format(productName+"in %s price
is %d
",gmall.getGmallName(),gmall.getPriceByProductName(productName))).collect(Collec
tors.toList());
    }

    /**
     * 异步
```

```
     * @param gmalls
     * @param productName
     * @return
     */
    public static List<String> getPriceAsync(List<Gmall> gmalls,String
productName){
        return gmalls.stream().map(gmall->CompletableFuture.supplyAsync(()-
>String.format(productName+"in %s price is %d
",gmall.getGmallName(),gmall.getPriceByProductName(productName)))).collect(Colle
ctors.toList()).stream().map(CompletableFuture::join).collect(Collectors.toList(
));
    }

    public static void main(String[] args) {

        long startTime = System.currentTimeMillis();
        getPriceSync(gmalls,"金鳞岂是池中物").forEach(System.out::println);
        long endTime = System.currentTimeMillis();
        System.out.println("同步"+ (endTime-startTime));

        System.out.println("------------------------");

        long s = System.currentTimeMillis();
        getPriceAsync(gmalls,"金鳞岂是池中物").forEach(System.out::println);
        long e = System.currentTimeMillis();
        System.out.println("异步"+ (e-s));
    }
}
@Data
@NoArgsConstructor
@AllArgsConstructor
class Gmall{
    private String gmallName;

    public Integer getPriceByProductName(String productName){
        try {
            TimeUnit.SECONDS.sleep(1);
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
        return (int)(Math.random()*100)+1;
    }
}
```

结果:

```
==================================
金鳞岂是池中物in 京东 price is 100
金鳞岂是池中物in 拼多多 price is 12
金鳞岂是池中物in 淘宝 price is 25
金鳞岂是池中物in 唯品会 price is 35
金鳞岂是池中物in 分期乐 price is 60
同步5081
------------------------
金鳞岂是池中物in 京东 price is 57
金鳞岂是池中物in 拼多多 price is 68
金鳞岂是池中物in 淘宝 price is 84
```

```
金鳞岂是池中物in 唯品会 price is 16
金鳞岂是池中物in 分期乐 price is 20
异步1387
==================================
```

2.检索

```
/**
 * @author yhd
 * @createtime 2020/11/15 23:40
 */
public class DemoC {

    //模拟数据库
    private static Map<String,Company> db=new HashMap<>();
    //模拟查询条件
    private static List<String> persons;
    //初始化数据
    static {
        persons=Arrays.asList("马云","马化腾","李彦宏","张朝阳","刘强东","王兴");
        db.put("马云",new Company("马云","阿里巴巴"));
        db.put("马化腾",new Company("马化腾","腾讯"));
        db.put("李彦宏",new Company("李彦宏","百度"));
        db.put("张朝阳",new Company("张朝阳","搜狐"));
        db.put("刘强东",new Company("刘强东","京东"));
        db.put("王兴",new Company("王兴","美团"));
    }
    //模拟去数据库查询
    public static Company selectDB(String key){
        try {
            TimeUnit.SECONDS.sleep(1);
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
        return db.get(key);
    }


    public static void main(String[] args) {
        Sync();
        System.out.println();
        Async();
        System.out.println();
        System.out.println(result());
    }
    //同步查询
    public static void Sync(){
        Long startTime=System.currentTimeMillis();
        persons.stream().map(key-
>selectDB(key).getWorks()).collect(Collectors.toList()).forEach(System.out::prin
tln);
        Long endTime=System.currentTimeMillis();
        System.out.println(endTime-startTime);
    }
    //异步查询
    public static void Async(){
        Long startTime=System.currentTimeMillis();
```

```
        persons.stream().map(key->CompletableFuture.supplyAsync(()->
>selectDB(key).getWorks())).collect(Collectors.toList()).stream().map(Completabl
eFuture::join).collect(Collectors.toList()).forEach(System.out::println);
        Long endTime=System.currentTimeMillis();
        System.out.println(endTime-startTime);
    }
    //转换成map返回
    public static Map<String,String> result(){
        return persons.stream().map(key -> CompletableFuture.supplyAsync(() ->
selectDB(key))).collect(Collectors.toList()).stream().map(CompletableFuture::joi
n).collect(Collectors.toMap(Company::getPerson,Company::getWorks));
    }

}
@Data
@NoArgsConstructor
@AllArgsConstructor
class Company{
    private String person;

    private String works;
}
```

结果

```
阿里巴巴
腾讯
百度
搜狐
京东
美团
6053

阿里巴巴
腾讯
百度
搜狐
京东
美团
1406

{张朝阳=搜狐, 马云=阿里巴巴, 王兴=美团, 李彦宏=百度, 刘强东=京东, 马化腾=腾讯}
```

3.商品详情

```
/**
 * @author yhd
 * @createtime 2020/11/16 0:45
 */
public class DemoD {

    private static Map<String, Result> result = new HashMap<>();
    private static ThreadPoolExecutor pool = new ThreadPoolExecutor(1, 5, 1,
TimeUnit.SECONDS, new ArrayBlockingQueue<>(1));
```

```
    public static void main(String[] args) {
        Long s = System.currentTimeMillis();
        CompletableFuture<Result> skuInfoFuture =
CompletableFuture.supplyAsync((DemoD::get), pool);
        CompletableFuture<Void> skuPriceFuture =
CompletableFuture.runAsync(DemoD::run, pool);
        CompletableFuture<Result> skuImgsFuture =
skuInfoFuture.thenApplyAsync(DemoD::apply, pool);
        CompletableFuture<Result> spuInfoFuture =
skuInfoFuture.thenApplyAsync(DemoD::apply2, pool);
        CompletableFuture.allOf(skuInfoFuture, skuPriceFuture, skuImgsFuture,
spuInfoFuture).join();
        System.out.println(result);
        Long e = System.currentTimeMillis();
        System.out.println("查詢總計耗時: "+(e-s));
        pool.shutdown();
    }

    /**
     * 查询sku基本信息
     */
    public static Result getSkuInfo(Integer key) {
        sleep();
        return new Result(1, "skuInfo");
    }

    /**
     * 查询sku图片信息
     */
    public static Result getSkuImgs(Integer key) {
        sleep();
        return new Result(1, "skuImgs");
    }

    /**
     * 查询spu销售属性
     */
    public static Result getSpuInfo(Integer key) {
        sleep();
        return new Result(1, "spuInfo");
    }

    /**
     * 查询sku价格
     */
    public static Result getSkuPrice(Integer key) {
        sleep();
        return new Result(1, "skuPrice");
    }

    public static void sleep() {
        try {
            TimeUnit.SECONDS.sleep(2);
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
    }
```

```
    private static Result get() {
        Result skuInfo = getSkuInfo(1);
        result.put(skuInfo.getName(), skuInfo);
        return skuInfo;
    }

    private static void run() {
        Result skuPrice = getSkuPrice(1);
        result.put(skuPrice.getName(), skuPrice);
    }

    private static Result apply(Result res) {
        Result skuImgs = getSkuImgs(res.getId());
        result.put(skuImgs.getName(), skuImgs);
        return res;
    }

    private static Result apply2(Result res) {
        Result spuInfo = getSpuInfo(res.getId());
        result.put(spuInfo.getName(), spuInfo);
        return res;
    }
}

@Data
@NoArgsConstructor
@AllArgsConstructor
class Result {
    private Integer id;

    private String name;
}
```