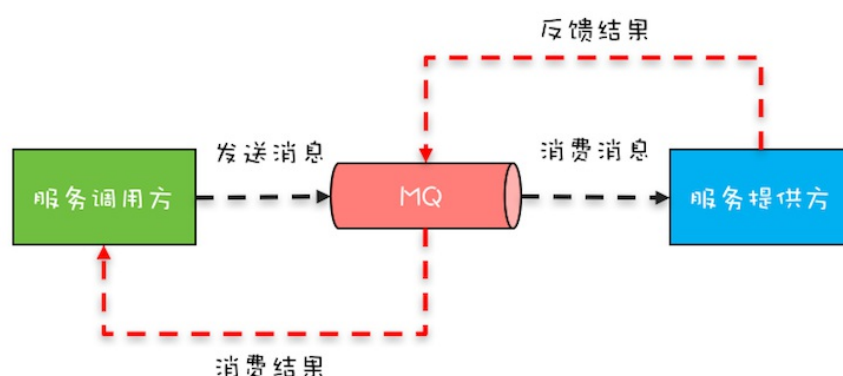


31 | Guarded Suspension模式：等待唤醒机制的规范实现

2019-05-09 王宝令



前不久，同事小灰工作中遇到一个问题，他开发了一个Web项目：Web版的文件浏览器，通过它用户可以在浏览器里查看服务器上的目录和文件。这个项目依赖运维部门提供的文件浏览服务，而这个文件浏览服务只支持消息队列（MQ）方式接入。消息队列在互联网大厂中用的非常多，主要用作流量削峰和系统解耦。在这种接入方式中，发送消息和消费结果这两个操作之间是异步的，你可以参考下面的示意图来理解。



消息队列（MQ）示意图

在小灰的这个Web项目中，用户通过浏览器发过来一个请求，会被转换成一个异步消息发送给MQ，等MQ返回结果后，再将这个结果返回至浏览器。小灰同学的问题是：给MQ发送消息的线程是处理Web请求的线程T1，但消费MQ结果的线程并不是线程T1，那线程T1如何等待MQ的返

回结果呢？为了便于你理解这个场景，我将其代码化了，示例代码如下。

```
class Message{
    String id;
    String content;
}

//该方法可以发送消息
void send(Message msg){
    //省略相关代码
}

//MQ消息返回后会调用该方法
//该方法的执行线程不同于
//发送消息的线程
void onMessage(Message msg){
    //省略相关代码
}

//处理浏览器发来的请求
Respond handleWebReq(){
    //创建一消息
    Message msg1 = new
        Message("1","{...}");
    //发送消息
    send(msg1);
    //如何等待MQ返回的消息呢？
    String result = ...;
}
```

看到这里，相信你一定有点似曾相识的感觉，这不就是前面我们在 [《15 | Lock和Condition（下）：Dubbo如何用管程实现异步转同步？》](#) 中曾介绍过的异步转同步问题吗？仔细分析，的确是这样，不过在那一篇文章中我们只是介绍了最终方案，让你知其然，但是并没有介绍这个方案是如何设计出来的，今天咱们再仔细聊聊这个问题，让你知其所以然，遇到类似问题也能自己设计出方案来。

Guarded Suspension模式

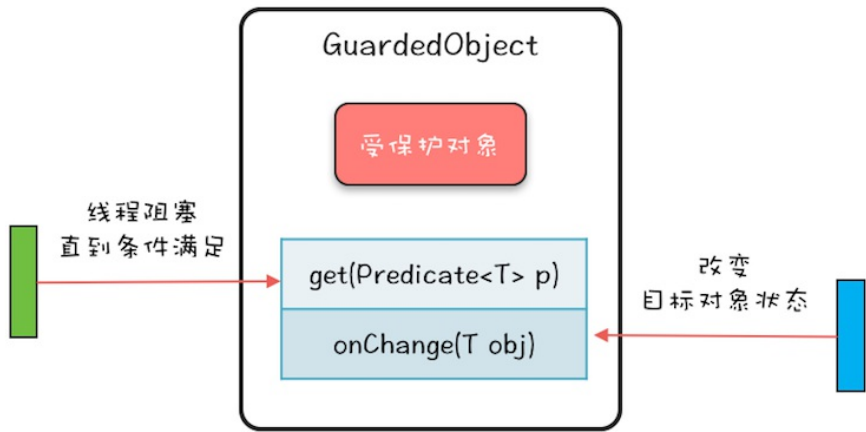
上面小灰遇到的问题，在现实世界里比比皆是，只是我们一不小心就忽略了。比如，项目组团建要外出聚餐，我们提前预订了一个包间，然后兴冲冲地奔过去，到那儿后大堂经理看了一眼包间，发现服务员正在收拾，就会告诉我们：“您预订的包间服务员正在收拾，请您稍等片刻。”过

了一会，大堂经理发现包间已经收拾完了，于是马上带我们去包间就餐。

我们等待包间收拾完这个过程和小灰遇到的等待MQ返回消息本质上是一样的，都是等待一个条件满足：就餐需要等待包间收拾完，小灰的程序里要等待MQ返回消息。

那我们来看看现实世界里是如何解决这类问题的呢？现实世界里大堂经理这个角色很重要，我们是否等待，完全是由他来协调的。通过类比，相信你也一定有思路了：我们的程序里，也需要这样一个大堂经理。的确是这样，那程序世界里的大堂经理该如何设计呢？其实设计方案前人早就搞定了，而且还将其总结成了一个设计模式：**Guarded Suspension**。所谓Guarded Suspension，直译过来就是“保护性地暂停”。那下面我们就来看看，Guarded Suspension模式是如何模拟大堂经理进行保护性地暂停的。

下图就是Guarded Suspension模式的结构图，非常简单，一个对象GuardedObject，内部有一个成员变量——受保护的對象，以及两个成员方法——get(Predicate<T> p)和onChanged(T obj)方法。其中，对象GuardedObject就是我们前面提到的大堂经理，受保护对象就是餐厅里面的包间；受保护对象的get()方法对应的是我们的就餐，就餐的前提条件是包间已经收拾好了，参数p就是用来描述这个前提条件的；受保护对象的onChanged()方法对应的是服务员把包间收拾好了，通过onChanged()方法可以fire一个事件，而这个事件往往能改变前提条件p的计算结果。下图中，左侧的绿色线程就是需要就餐的顾客，而右侧的蓝色线程就是收拾包间的服务员。



Guarded Suspension模式结构图

GuardedObject的内部实现非常简单，是管程的一个经典用法，你可以参考下面的示例代码，核心是：**get()**方法通过条件变量的**await()**方法实现等待，**onChanged()**方法通过条件变量的**signalAll()**方法实现唤醒功能。逻辑还是很简单的，所以这里就不再详细介绍了。

```
class GuardedObject<T>{
    //受保护的對象
    T obj;
    final Lock lock =
        new ReentrantLock();
    final Condition done =
        lock.newCondition();
    final int timeout=1;
    //获取受保护对象
    T get(Predicate<T> p) {
        lock.lock();
        try {
            //MESA管程推荐写法
            while(!p.test(obj)){
                done.await(timeout,
                    TimeUnit.SECONDS);
            }
        }catch(InterruptedException e){
            throw new RuntimeException(e);
        }finally{
            lock.unlock();
        }
        //返回非空的受保护对象
        return obj;
    }
    //事件通知方法
    void onChanged(T obj) {
        lock.lock();
        try {
            this.obj = obj;
            done.signalAll();
        } finally {
            lock.unlock();
        }
    }
}
```

扩展Guarded Suspension模式

上面我们介绍了Guarded Suspension模式及其实现，这个模式能够模拟现实世界里大堂经理的角色，那现在我们再来看看这个“大堂经理”能否解决小灰同学遇到的问题。

Guarded Suspension模式里GuardedObject有两个核心方法，一个是get()方法，一个是onChanged()方法。很显然，在处理Web请求的方法handleWebReq()中，可以调用GuardedObject的get()方法来实现等待；在MQ消息的消费方法onMessage()中，可以调用GuardedObject的onChanged()方法来实现唤醒。

```
//处理浏览器发来的请求
Respond handleWebReq(){
    //创建一消息
    Message msg1 = new
        Message("1","{...}");
    //发送消息
    send(msg1);
    //利用GuardedObject实现等待
    GuardedObject<Message> go
        =new GuardObjec<>();
    Message r = go.get(
        t->t != null);
}
void onMessage(Message msg){
    //如何找到匹配的go?
    GuardedObject<Message> go=???
    go.onChanged(msg);
}
```

但是在实现的时候会遇到一个问题，handleWebReq()里面创建了GuardedObject对象的实例go，并调用其get()方等待结果，那在onMessage()方法中，如何才能找到匹配的GuardedObject对象呢？这个过程类似服务员告诉大堂经理某某包间已经收拾好了，大堂经理如何根据包间找到就餐的人。现实世界里，大堂经理的头脑中，有包间和就餐人之间的关系图，所以服务员说完之后大堂经理立刻就能把就餐人找出来。

我们可以参考大堂经理识别就餐人的办法，来扩展一下Guarded Suspension模式，从而使它能够很方便地解决小灰同学的问题。在小灰的程序中，每个发送到MQ的消息，都有一个唯一性的属性id，所以我们可以维护一个MQ消息id和GuardedObject对象实例的关系，这个关系可以类比

大堂经理大脑里维护的包间和就餐人的关系。

有了这个关系，我们来看看具体如何实现。下面的示例代码是扩展Guarded Suspension模式的实现，扩展后的GuardedObject内部维护了一个Map，其Key是MQ消息id，而Value是GuardedObject对象实例，同时增加了静态方法create()和fireEvent(); create()方法用来创建一个GuardedObject对象实例，并根据key值将其加入到Map中，而fireEvent()方法则是模拟的大堂经理根据包间找就餐人的逻辑。

```
class GuardedObject<T>{
    //受保护的對象
    T obj;
    final Lock lock =
        new ReentrantLock();
    final Condition done =
        lock.newCondition();
    final int timeout=2;
    //保存所有 GuardedObject
    final static Map<Object, GuardedObject>
    gos=new ConcurrentHashMap<>();
    //静态方法创建 GuardedObject
    static <K> GuardedObject
        create(K key){
        GuardedObject go=new GuardedObject();
        gos.put(key, go);
        return go;
    }
    static <K, T> void
        fireEvent(K key, T obj){
        GuardedObject go=gos.remove(key);
        if (go != null){
            go.onChanged(obj);
        }
    }
    //获取受保护对象
    T get(Predicate<T> p) {
        lock.lock();
        try {
            //MESA管程推荐写法
```

```
while(!p.test(obj)){
    done.await(timeout,
        TimeUnit.SECONDS);
}
}catch(InterruptedException e){
    throw new RuntimeException(e);
}finally{
    lock.unlock();
}
//返回非空的受保护对象
return obj;
}

//事件通知方法
void onChanged(T obj) {
    lock.lock();
    try {
        this.obj = obj;
        done.signalAll();
    } finally {
        lock.unlock();
    }
}
}
```

这样利用扩展后的**GuardedObject**来解决小灰同学的问题就很简单了，具体代码如下所示。

```
//处理浏览器发来的请求
Respond handleWebReq(){
    int id=序号生成器.get();
    //创建一消息
    Message msg1 = new
        Message(id,"{...}");
    //创建GuardedObject实例
    GuardedObject<Message> go=
        GuardedObject.create(id);
    //发送消息
    send(msg1);
    //等待MQ消息
    Message r = go.get(
        t->t != null);
}
void onMessage(Message msg){
    //唤醒等待的线程
    GuardedObject.fireEvent(
        msg.id, msg);
}
```

总结

Guarded Suspension模式本质上是一种等待唤醒机制的实现，只不过**Guarded Suspension**模式将其规范化了。规范化的好处是你无需重头思考如何实现，也无需担心实现程序的可理解性问题，同时也能避免一不小心写出个Bug来。但**Guarded Suspension**模式在解决实际问题的時候，往往还是需要扩展的，扩展的方式有很多，本篇文章就直接对**GuardedObject**的功能进行了增强，**Dubbo**中**DefaultFuture**这个类也是采用的这种方式，你可以对比着来看，相信对**DefaultFuture**的实现原理会理解得更透彻。当然，你也可以创建新的类来实现对**Guarded Suspension**模式的扩展。

Guarded Suspension模式也常被称作**Guarded Wait**模式、**Spin Lock**模式（因为使用了**while**循环去等待），这些名字都很形象，不过它还有一个更形象的非官方名字：多线程版本的**if**。单线程场景中，**if**语句是不需要等待的，因为在只有一个线程的条件下，如果这个线程被阻塞，那就没有其他活动线程了，这意味着**if**判断条件的结果也不会发生变化了。但是多线程场景中，等待就变得有意义了，这种场景下，**if**判断条件的结果是可能发生变化的。所以，用“多线程版本的**if**”来理解这个模式会更简单。

课后思考

有同学觉得用`done.await()`还要加锁，太啰嗦，还不如直接使用`sleep()`方法，下面是他的实现，你觉得他的写法正确吗？

```
//获取受保护对象
T get(Predicate<T> p) {
    try {
        while(!p.test(obj)){
            TimeUnit.SECONDS
                .sleep(timeout);
        }
    } catch (InterruptedException e){
        throw new RuntimeException(e);
    }
    //返回非空的受保护对象
    return obj;
}

//事件通知方法
void onChanged(T obj) {
    this.obj = obj;
}
```

欢迎在留言区与我分享你的想法，也欢迎你在留言区记录你的思考过程。感谢阅读，如果你觉得这篇文章对你有帮助的话，也欢迎把它分享给更多的朋友。

Java 并发编程实战

全面系统提升你的并发编程能力

王宝令

资深架构师



新版升级：点击「👤请朋友读」，20位好友免费读，邀请订阅更有**现金**奖励。

精选留言



青莲

👍 10

sleep 无法被唤醒，只能时间到后自己恢复运行，当真正的条件满足了，时间未到，接着睡眠，无性能可言

2019-05-09



刘章周

👍 6

当从消息队列接收消息失败时，while循环会一直执行下去，永远不会结束，回占用大量资源。

2019-05-09

作者回复

👍

2019-05-09



zhangwei

👍 5

老师，我有个疑问，希望帮忙解答。如果Web应用是集群的，A节点处理HTTP请求后发了MQ，B节点的onMessage消费了回执消息，那么A节点怎么把结果响应给客户端呢？疑问好久了，希望老师给个思路，谢谢！

2019-05-18

作者回复

我了解有人是这么做的：把回执消息放到redis的list中，按照ip重新分组之后从redis中再次消费。

也可以按照ip建立不同的topic。

2019-05-18



Felix Envy

👍 5

老师，感觉如果有方法调用了GuardedObject.create方法但是没有任何其他线程调用fireEvent方法会造成内存泄漏啊，这种情况需要考虑吗？

2019-05-12

作者回复

需要，等待超时后要把他移除。

2019-05-13



Mr. Brooks

👍 3

没有锁也无法保证内存可见性吧

2019-05-10

作者回复

需要

2019-05-13



张三

👍 2

接入微信支付支付宝支付里边，也需要提供一个回调函数，onChange()就是一个回调函数吧，不过微信支付支付宝支付是异步回调，是不是也可以改成这种？微信支付支付宝里的其它第三方支付是不是就是这种模式，因为支付成功之后跳转到它们自己的页面，而不是微信支付支付宝官方的支付成功界面

2019-05-10

作者回复

这个回调函数和mq的回调函数从服务接入方的角度看是一样的

2019-05-12



zero

👍 2

wait会释放占有的资源，sleep不会释放

2019-05-09



null

👍 1

老师，您好！

我想到了一个场景：线程 t1 提交了消息 m1，线程 t2 提交了消息 m2，此时都在 get() 方法处等待结果返回。m2 先被处理完，this.obj 对应的是消息 m2 的结果，调用 fireEvent() 唤醒 t1 和 t2，t1 竞争到锁资源，消费了 m2 的结果 this.obj。

如果存在这种场景，再维护一个 ConcurrentHashMap，key 是 msg.id，value 是对应的 obj，是否就能解决结果这问题？

谢谢老师！

2019-06-03

作者回复

只要唤醒的时候能找到正确的线程就可以，不知道你的方法是不是能做到

2019-06-03



Rancood

1

这个模式了解了，但是实例中业务有点懵，`handleWebReq`方法最终拿到的是自己发送出去的`message`消息；是不是应该在`onMessage`方法唤醒等待线程之前进行业务处理，生成新的`Message`消息`newMsg`，然后把`newMsg`传到`fireEvent`里面，这样拿到的是反馈的结果。请老师指点一下

2019-05-22



晓杰

1

用`sleep`的话只能等睡眠时间到了之后再返回`while`循环条件去判断，但是`wait`相当于和`signal`组成等待唤醒的机制，这样满足条件的概率更大一些，性能也更好

2019-05-09



朱晋君

1

如果以文中的最后一段示例代码来看，每一个请求生成一个`id`，对应一个`GuardedObject`，并没有线程安全问题。我觉得可以去掉锁。

但是加`sleep`的话，没有办法唤醒，只能等到超时。

2019-05-09

作者回复

`await`和`notify`获取锁才能调用，所以不能去掉锁

2019-05-09



linqw

0

总结：`Guarded Suspension`模式，要解决的是，发送消息的线程和消费消息结果的线程不是同一个，但是消息结果又需要由发送的线程进行处理，为此需要为每个消息创建出类似大堂经理，生活中一般是只有一个大堂经理，但是在编程世界里需要为每个分配一个大堂经理，大堂经理主要做的事情就是发送线程发送完消息时，将其阻塞，提供消息结果的回调接口，通知阻塞的发送线程消费消息结果。

课后习题：1、使用`sleep`如果消息结果已经返回，还需等到`sleep`超时，才能继续执行2、使用加锁的`await`方法可以保证可见性，如果使用`sleep`的话，需要给`obj`加上`volatile`3、感觉在`sent (message)`成功后才阻塞，不然有可能`mq`接收消息失败，`while`循环会一直执行下去，`sent`失败，直接响应提示给前端。

2019-05-25



cricket1981

0

想问一下分布式环境下，异步转同步的方法有哪些？例如，数据服务部署多个`instance`，客户在`Web UI`上点击外部数据源试用，后端通过一个数据服务`instance`请求外部数据源，外部数据源会异步回调结果(`LB`地址)返回，怎么样将结果显示在请求数据服务的`Web UI`上？客户试用的过程是同步的，但请求外部数据源操作流程是异步的。谢谢！

2019-05-24

作者回复

`websocket`可以吗，没了解到你说的难点在哪里

2019-05-24



andy

0

我有个疑惑就是，这里所说的MQ可以是RabbitMQ或者是其他类型的MQ吗？还是说这个MQ其实就JAVA中的一个数据结构？

2019-05-20

作者回复

前者

2019-05-20



三木禾

0

老师，您这代码能不能写的完整一点啊，前面有个onMessage ,后面有个onChange

2019-05-19

作者回复

onmessage调用 onchange,他们是两个类里的方法

2019-05-20



Zach_

0

await(timeout); 本来是去等待队列里面呆timeout长的时间，如果到timeout之前没有被signal就出队;如果被signal了就是机制的正常运行。最终都return obj;

sleep(timeout); 守护条件不成立就sleep timeout, 还不成立再sleep timeout, ... 如果在发生死锁或者onchange失败的情况下， get()方法会因为状态改变的失败而一直 判断 sleep下去

老师我这里添加了死锁，不知道说的对不对啊，恳请老师回复。

还有，为什么我没办法用这种独到的眼力去套用一些框架代码的相似处啊？是知道的太少了嘛？

2019-05-15



Zach_

0

额 我看有个同学说 可见性也无法保证。

锁是保证可见性的嘛？ 我记得锁是保证原子性的哇， 可见性不是volatile变量来保证的嘛？

2019-05-15



Geek_ebda96

0

sleep方法，sleep期间，不会释放对象的锁，线程T1执行get，获取的lock在sleep期间他的锁不会释放，如果t2线程去执行onchanged方法，这时候获取不到lock的锁的，会导致程序死锁，await方法，线程T1执行get,在等待期间，他会释放掉锁，这时t2执行onchanged，可以获取到lock然后给obj赋值，get1就可以获取到最新的obj

对于reentrantlock，执行lock后，如果不释放unlock，执行了await，其他线程还是可以获取到lock

2019-05-14



佑儿

👍 0

学习心得：每一种设计模式都适应某种场景，个人认为Guarded Suspension模式的根本在于get数据的时候一定能够获取，如果获取不到就等待，保护性暂停也就是这个意思，阻塞队列大部分都是这种模式。

2019-05-14



Kaleidoscoper

👍 0

想问下老师，Create和fireEvent方法不加锁可以么，像Create方法不加锁指令重排优化，先返回object再加进map里会不会有问题

2019-05-14

作者回复

线程安全的容器能避免你说的这种情况

2019-05-15