# 学习攻略 | 如何才能学好并发编程？

2019-02-26 王宝令

并发编程并不是一门相对独立的学科，而是一个综合学科。并发编程相关的概念和技术看上非常零散，相关度也很低，总给你一种这样的感觉：我已经学习很多相关技术了，可还是搞不定并发编程。那如何才能学习好并发编程呢？

其实很简单，只要你能从两个方面突破一下就可以了。一个是“跳出来，看全景”，另一个是“钻进去，看本质”。

## 跳出来，看全景

我们先说“跳出来”。你应该也知道，学习最忌讳的就是“盲人摸象”，只看到局部，而没有看到全局。所以，你需要从一个个单一的知识和技术中“跳出来”，高屋建瓴地看并发编程。当然，这**首要之事就是你建立起一张全景图**。

不过，并发编程相关的知识和技术还真是错综复杂，时至今日也还没有一张普遍认可的全景图，也许这正是很多人在并发编程方面难以突破的原因吧。好在经过多年摸爬滚打，我自己已经“勾勒”出了一张全景图，不一定科学，但是在某种程度上我想它还是可以指导你学好并发编程的。

在我看来，并发编程领域可以抽象成**三个核心问题：分工、同步和互斥**。

### 1. 分工

所谓分工，类似于现实中一个组织完成一个项目，项目经理要拆分任务，安排合适的成员去完成。

在并发编程领域，你就是项目经理，线程就是项目组成员。任务分解和分工对于项目成败非常关键，不过在并发领域里，分工更重要，它直接决定了并发程序的性能。在现实世界里，分工是很复杂的，著名数学家华罗庚曾用“烧水泡茶”的例子通俗地讲解了统筹方法（一种安排工作进程的数学方法），“烧水泡茶”这么简单的事情都这么多说道，更何况是并发编程里的工程问题呢。

既然分工很重要又很复杂，那一定有前辈努力尝试解决过，并且也一定有成果。的确，在并发编程领域这方面的成果还是很丰硕的。Java SDK并发包里的Executor、Fork/Join、Future本质上都是一种分工方法。除此之外，并发编程领域还总结了一些设计模式，基本上都是和分工方法相关的，例如生产者-消费者、Thread-Per-Message、Worker Thread模式等都是用来指导你如何分工的。

学习这部分内容，最佳的方式就是和现实世界做对比。例如生产者-消费者模式，可以类比一下餐馆里的大厨和服务员，大厨就是生产者，负责做菜，做完放到出菜口，而服务员就是消费者，把做好的菜给你端过来。不过，我们经常会发现，出菜口有时候一下子出了好几个菜，服务员是可以把这一批菜同时端给你的。其实这就是生产者-消费者模式的一个优点，生产者一个一个地生产数据，而消费者可以批处理，这样就提高了性能。

### 2. 同步

分好工之后，就是具体执行了。在项目执行过程中，任务之间是有依赖的，一个任务结束后，依赖它的后续任务就可以开工了，后续工作怎么知道可以开工了呢？这个就是靠沟通协作了，这是一项很重要的工作。

在并发编程领域里的同步，主要指的就是线程间的协作，本质上和现实生活中的协作没区别，不过是**一个线程执行完了一个任务，如何通知执行后续任务的线程开工**而已。

协作一般是和分工相关的。Java SDK并发包里的Executor、Fork/Join、Future本质上都是分工方法，但同时也能解决线程协作的问题。例如，用Future可以发起一个异步调用，当主线程通过get()方法取结果时，主线程就会等待，当异步执行的结果返回时，get()方法就自动返回了。主线程和异步线程之间的协作，Future工具类已经帮我们解决了。除此之外，Java SDK里提供的CountDownLatch、CyclicBarrier、Phaser、Exchanger也都是用来解决线程协作问题的。

不过还有很多场景，是需要你自己来处理线程之间的协作的。

工作中遇到的线程协作问题，基本上都可以描述为这样的一个问题：**当某个条件不满足时，线程需要等待，当某个条件满足时，线程需要被唤醒执行**。例如，在生产者-消费者模型里，也有类似的描述，“当队列满时，生产者线程等待，当队列不满时，生产者线程需要被唤醒执行；当队列空时，消费者线程等待，当队列不空时，消费者线程需要被唤醒执行。”

在Java并发编程领域，解决协作问题的核心技术是**管程**，上面提到的所有线程协作技术底层都是利用管程解决的。管程是一种解决并发问题的通用模型，除了能解决线程协作问题，还能解决

下面我们将要介绍的互斥问题。可以这么说，**管程是解决并发问题的万能钥匙**。

所以说，这部分内容的学习，关键是理解管程模型，学好它就可以解决所有问题。其次是了解Java SDK并发包提供的几个线程协作的工具类的应用场景，用好它们可以妥妥地提高你的工作效率。

### 3. 互斥

分工、同步主要强调的是性能，但并发程序里还有一部分是关于正确性的，用专业术语叫“**线程安全**”。并发程序里，当多个线程同时访问同一个共享变量的时候，结果是不确定的。不确定，则意味着可能正确，也可能错误，事先是不知道的。而导致不确定的主要源头是可见性问题、有序性问题和原子性问题，为了解决这三个问题，Java语言引入了内存模型，内存模型提供了一系列的规则，利用这些规则，我们可以避免可见性问题、有序性问题，但是还不足以完全解决线程安全问题。解决线程安全问题的核心方案还是互斥。

**所谓互斥，指的是同一时刻，只允许一个线程访问共享变量。**

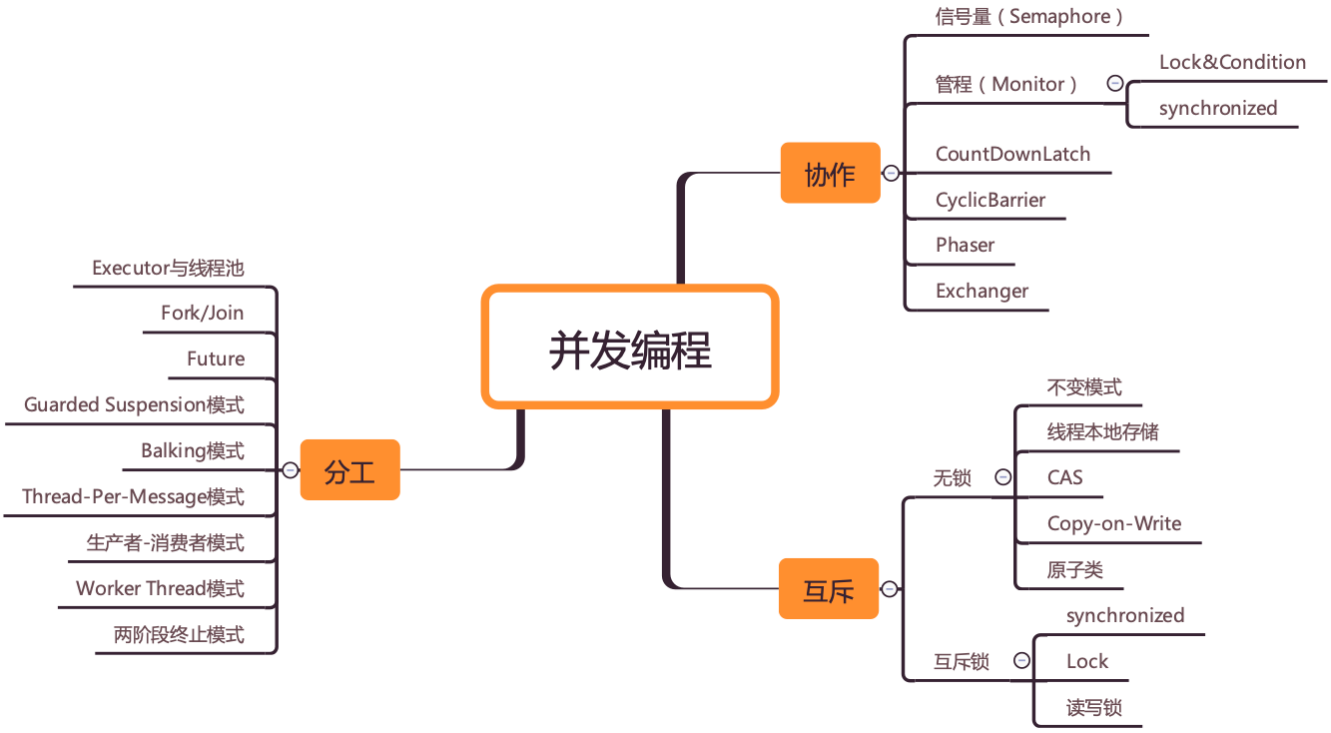
实现互斥的核心技术就是锁，Java语言里synchronized、SDK里的各种Lock都能解决互斥问题。虽说锁解决了安全性问题，但同时也带来了性能问题，那如何保证安全性的同时又尽量提高性能呢？可以分场景优化，Java SDK里提供的ReadWriteLock、StampedLock就可以优化读多写少场景下锁的性能。还可以使用无锁的数据结构，例如Java SDK里提供的原子类都是基于无锁技术实现的。

除此之外，还有一些其他的方案，原理是不共享变量或者变量只允许读。这方面，Java提供了Thread Local和final关键字，还有一种Copy-on-write的模式。

使用锁除了要注意性能问题外，还需要注意死锁问题。

这部分内容比较复杂，往往还是跨领域的，例如要理解可见性，就需要了解一些CPU和缓存的知识；要理解原子性，就需要理解一些操作系统的知识；很多无锁算法的实现往往也需要理解CPU缓存。这部分内容的学习，需要博览群书，在大脑里建立起CPU、内存、I/O执行的模拟器。这样遇到问题就能得心应手了。

跳出来，看全景，可以让你的知识成体系，所学知识也融汇贯通起来，由点成线，由线及面，画出自己的知识全景图。




并发编程全景图之思维导图

## 钻进去，看本质

但是光跳出来还不够，还需要下一步，就是在某个问题上钻进去，深入理解，找到本质。

就拿我个人来说，我已经烦透了去讲述或被讲述一堆概念和结论，而不分析这些概念和结论是怎么来的，以及它们是用来解决什么问题的。在大学里，这样的教材很流行，直接导致了芸芸学子成绩很高，但解决问题的能力很差。其实，知其然知其所以然，才算真的学明白了。

我属于理论派，**我认为工程上的解决方案，一定要有理论做基础**。所以在学习并发编程的过程中，我都会探索它背后的理论是什么。比如，当看到Java SDK里面的条件变量Condition的时候，我会下意识地问，“它是从哪儿来的？是Java的特有概念，还是一个通用的编程概念？”当我知道它来自管程的时候，我又会问，“管程被提出的背景和解决的问题是什么？”这样一路探索下来，我发现Java语言里的并发技术基本都是有理论基础的，并且这些理论在其他编程语言里也有类似的实现。所以我认为，技术的本质是背后的理论模型。

## 总结

当初我学习Java并发编程的时候，试图上来就看Java SDK的并发包，但是很快就放弃了。原因是我觉得东西太多，眼花缭乱的，虽然借助网络上的技术文章，感觉都看懂了，但是很快就又忘了。实际应用的时候大脑也一片空白，根本不知道从哪里下手，有时候好不容易解决了个问题，也不知道这个方案是不是合适的。

我知道根本原因是，我的并发知识还没有成体系。

我想，要让自己的知识成体系，一定要挖掘Java SDK并发包背后的设计理念。Java SDK并发包是并发大师Doug Lea设计的，他一定不是随意设计的，一定是深思熟虑的，其背后是Doug Lea对并发问题的深刻认识。可惜这个设计的思想目前并没有相关的论文，所以只能自己琢磨了。

分工、同步和互斥的全景图，是我对并发问题的个人总结，不一定正确，但是可以帮助我快速建立解决并发问题的思路，梳理并发编程的知识，加深认识。我将其分享给你，希望对你也有用。

对于某个具体的技术，我建议你探索它背后的理论本质，理论的应用面更宽，一项优秀的理论往往在多个语言中都有体现，在多个不同领域都有应用。所以探求理论本质，既能加深对技术本身的理解，也能拓展知识深度和广度，这是个一举多得的方法。这方面，希望我们一起探讨，共同进步。

欢迎在留言区跟我分享你的经历与想法。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。



精选留言

常江舟

122

从性能角度讲，我们为了提高执行一定计算机任务的效率，所以IO等待的时候不能让cpu闲着，所以我们把任务拆分交替执行，有了分时操作系统，出现了并发，后来cpu多核了又有了并行计算。这里也就是作者说的[分工]。分工以后我们为了进一步提升效率和更加灵活地达到目的，所以我们要对任务进行组织编排，也就是对线程组织编排。于是线程之间需要通信，于是操

作系统提供了一些让进程，线程之间通信的方式。也就是作者说的[同步]。但是事物总不是完美的。并发和通信带来了较高的编程复杂度，同时也出现了多线程并发操作共享资源的问题。于是天下大势，分久必合，我们又要将对共享资源的访问串行化。所以我们根据现实世界的做法设计了了锁，信号量等等来补充这套体系。也就是作者所说的[互斥]!

综上，这一切均为提高性能的手段和对其所产生问题的解决方案。

2019-02-26

作者回复

正解

2019-02-26

Minecraft

68

并发编程学习 第一天 明天去面试 祝我好运

2019-02-26

Jerry银银

53

这篇文章看了四五篇，写得真好，收获也很多。
文中提到了两点真是发人深省：
1. 方法论层面：「跳出来，看全景」 和 「钻进去，看本质」，这两条方法论，我想是适合很多领域的学习的。
2. 并发领域的「全景图」。
对于「全景图」，我之前也有一直在构建，可是因为知识储备不够，确实很难构建出来。稍微了解过并发领域知识的人都知道，里面的知识点、概念多而散：线程安全、锁、同步、异步、阻塞、非阻塞、死锁、队列(为什么并发要跟队列扯上关系)、闭锁、信号量、活锁等等。如果单个去学这些知识点，单个去练习，如果没有「主线」，后期很容易忘。我思考再思考，也总结了一下学习并发的主线：
首先，得理解并发的重要性，为什么需要并发？对于这个问题，只需要放在潜意识里面，只需要两个字：性能！其它的细节，再去慢慢拓展。
然后，既然并发很重要，而并发处理的是任务，接下就是：对任务的抽象、拆解、分工执行。而线程模型，只是其中的一种模型，还有多进程、协程。Java使用的是多线程模型，对应到具体的代码就是：Thread, Runnable, Task，执行任务有：Exectors。 引出了线程，有势必存在着线程安全性的问题，因为多线程访问，数据存在着不一致的问题。
再然后，大的任务被拆解多个小的子任务，小的子任务被各自执行，不难想象，子任务之间肯定存在着依赖关系，所以需要协调，那如何协调呢？也不难想到，锁是非常直接的方式(Monitor原理)，但是只用锁，协调的费力度太高，在并发的世界里面，又有了一些其它的更抽象的工具：闭锁、屏障、队列以及其它的一些并发容器等；好了，协调的工作不难处理了。可是协调也会有出错的时候，这就有了死锁、活锁等问题，大师围绕着这个问题继续优化协调工具，尽量让使用者不容易出现这些活跃性问题；
到此，「并发」的历史还在演化：如果一遇到并发问题，就直接上锁，倒也没有什么大问题，可是追求性能是人类的天性。计算机大师就在思考，能不不加锁也能实现并发，还不容易出错，于是就有了：CAS、copy-on-write等技术思想，这就是实现了「无锁」并发；
可是，事情到此还没有完。如果以上这些个东西，都需要每个程序员自己去弄，然后自己保证正确性，那程序员真累死了，哪还有时间、精力创造这么多美好的应用！于是，计算机大师又

开始思考，能不能抽象出统一「模型」，可能这就有了类似于「Java内存模型」这样的东西。

---

借用宝令老师的语言，以上「是我对并发问题的个人总结，不一定正确，但是可以帮助我快速建立解决并发问题的思路，梳理并发编程的知识，加深认识。我将其分享给你，希望对你也有用」。

2019-03-06

作者回复

我觉得你比我总结的好

2019-03-06

Jerry银银 37

之前看薛兆丰的《经济学通识》，他总结到，人类面临着四大基本约束：东西不够，生命有限，互相依赖，需要协调。当我看到这句话的时候，我猛然间意识到：计算机也同样面临着这四大基本约束。

在计算中，CPU、内存、IO、硬盘、带宽等，这些资源也都有不够的时候，而每个线程的也有着自己的生命周期，并且它们之间又是相互依赖的，也同样需要协调。

有了上面的这种想法，我觉得我学习计算机的知识有了章法可循。

2019-03-06

作者回复

好厉害

2019-03-06

crazypokerk 23

感觉确实如老师所说的，知识不成体系，就像是奶酪，看着是一块，实则满眼孔洞，加油！

2019-02-26

作者回复

这个比喻我是服了

2019-02-28



Handongyang 19

想给老师提一个建议，就是在开篇用一个问题来引出本篇所要讲述的内容，然后在结尾时的总结之前回答开篇的问题。最后，在总结之后再设计并提出一个问题，让大家来讨论和回答。每一课之后的激烈讨论将是最有意思的，望老师考虑一下，谢谢！

2019-02-26

作者回复

你的建议非常好，我努力向这个方向前进

2019-02-28

我会得到 16

全局思维加单点突破，这种方式屡试不爽。希望令哥沉住气不着急，好好打磨，慢慢更新，搞出精品，打造业界标杆

2019-02-26

作者回复

借你吉言

2019-02-28

凌

15

令哥，你就坐我对面，让我如何评论啊！呵呵

2019-02-26

作者回复

使劲夸就行了，我不介意

2019-02-28

墙角

11

正如老师所说，并发编程涉及的知识面比较广，无奈大学阶段没有学好，老师帮忙推荐下和并发编程相关的书籍。只有有了一定的知识铺垫，才能更好的理解并发编程。感谢！

2019-03-06

作者回复

《Java并发编程实战》作者阵容可谓大师云集，也包括Doug Lea

《Java并发编程的艺术》讲解并发包内部实现原理，能读明白，内功大增

《图解Java多线程设计模式》并发编程设计模式方面的经典书籍

《操作系统：精髓与设计原理》经典操作系统教材

http://ifeve.com 国内专业并发编程网站

http://www.cs.umd.edu/~pugh/java/memoryModel/ 很多并发编程的早期资料都在这里

2019-03-06

王二宝

8

我和老师的观念是一样的，如果碰到自己一直搞不定的问题时，我的应对方法也是：从两个方面突破。一个是“跳出来，看全景”，另一个是“钻进去，看本质”。

2019-02-26

作者回复

同感

2019-02-28

冉

7

个人一点建议因大家基础功底不一且并发知识面广，若专栏篇幅受限的话，还望讲的过程中给出一些相关链接知识，方便理解，望采纳！

2019-03-01

作者回复

多谢建议，必须采纳！

2019-03-01

梅云霞

7

总结

当初我学习 Java 并发编程的时候，试图上来就看 Java SDK 的并发包，但是很快就放弃了。原因是我觉得东西太多，眼花缭乱的，虽然借助网络上的技术文章，感觉都看懂了，但是很快就又忘了。实际应用的时候大脑也一片空白，根本不知道从哪里下手，有时候好不容易解决了个问题，也不知道这个方案是不是合适的。
说出了我的处境！如何记住Java SDK 的并发包就像记住ABC呢

2019-02-26

作者回复

买个专栏啊

2019-02-28

minggushen

6

老师想请教您一个问题，目前公司需要进行分表操作，单表2亿数据，每年的增量也是两亿。有没有什么理论基础支持我分片的片数，以及是否需要分库以及其他注意事项。如果没有的话，老师按照您的经验，应该分成多少个片呢？目前是用的哈希对128取模进行的，分成128个表，是否合适呢。

2019-02-26

作者回复

建议先做个冷热分离吧，如果不能做，建议分库，分片规则很重要，要结合业务，具体问题具体分析。回头我再出个分布式计算的专栏.....

2019-02-28

Joker

5

这是我在极客学院购买的第一份产品，也是因为纯洁的微笑才买的这份产品，希望自己能坚持下去，在并发编程这方面有点突破

2019-02-28

编辑回复

极客时间，哥哥

2019-02-28

发条橙子 。

3

总结：

并发编程需要构造出一个全景图 。 只要分为三大点 ： 分工、协作、互斥。

先将一个大的逻辑按不同的工作去分配给不同的线程 ， 这些线程可以同时进行 ， 也可以一个线程结束后再进行下一个进程，这时就需要线程间的协作 ， 最后如果是多个线程同时进行并且会访问同一个共享资源时就需要对这个资源加锁以便造成资源的不一致 ， 这就是互斥

当全景图有了就需要深入到各个点， java sdk 的并发包中提供了很多工具帮我们处理上面三大点， 比如分工的fork/join 、协作的 future 、 互斥的各种锁以及无锁原子类等 。

这只是表面 ， 我们更要深入其中了解其背后的理论，比如很多 sdk并发包中的工具都是基于管程的思想 。 了解了这个思想才能举一反三 ， 站在更高的视野去理解并发的本质

另外，希望老师也可以着重讲一些管程这类的原理概念，我也是第一次听到这个词甚是感兴趣，并且老师说道并发的处理大多涉及操作系统相关的知识，也希望老师能推荐一些书籍或者文章资料便于我们更深入的理解学习

2019-02-26

作者回复

专栏有专门一期讲管程，每一期都会有相关推荐

2019-02-28

木易走刀口 3

好东西值得认真学习

2019-02-26

作者回复

感谢杨总支持！

2019-02-28

邋遢的流浪剑客 3

过年的时候看了一遍java并发编程的艺术，感觉有点晕，正好跟着老师的课在深入理解一下

2019-02-26

作者回复

那本书属于高段位的，适合学完这个专栏后再看

2019-02-28

遠い道の先で 2

1、跳出来看全景，钻进去看本质。

-在进入一个新领域学习时，建立一张学习线路的全景图，由点成线由线成面，贯穿整个学习过程。

-在学到某个具体问题时，钻进去看本质，了解技术背后的理论模型，了解当初这个理论产生的环境时什么，主要解决什么问题。

2、一些重要的知识前任一定有所研究并有相应的结果，可以先查阅目前最可靠的解决方案，提高自己的基线。

3、分工：将一个大的任务（项目）拆分成若干个小任务，并安排适合的成员去执行。

4、同步：每个小任务间可能存在相互依赖，同步需要做的是在前置任务完成后，通知后置任务启动。

5、互斥：互斥主要解决正确性问题。互斥要求同一时间，只允许一个线程访问共享变量。

6、管程monitor是解决并非问题的万？能？钥匙。（这个完全不理解）

7、一项优秀的理论往往在多个语言中都有体现，学习过程中应注重理论的学习。

———

以上是听课过程中的笔记，下面说说自己的感悟：

之前听过吴军老师的课程，讲到民间科学家为何难以作出成绩，是因为基线不高。老师在课中有讲到我们可以通过学习优秀前辈研究和成果去学习并发编程。非常同意两位老师的观点。

跳出来看全景，钻进去看本质。这个观点很新颖我也非常认同，在日常的碎片化学习往往会让我们这学一点那些一点，感觉好像学了很多，但又感觉不知道学习什么。

在学习前制定一张学习地图，就像一棵树先有树干再有树枝，会成长的枝繁叶茂的。

平时遇到一个新的知识，我们可能一开始只知道怎么用，没有去探究它的本质，可能导致最终我们只会基础的使用而没有办法举一反三，下次遇到了还是新问题。
这是第一次在极客时间购买课程，希望自己能跟上每一堂课，和大家一起成长。

2019-03-13

作者回复

眼光不错哈哈

2019-03-13

Weixiao

2

老师你好，看一些英文资料，你文中表达的互斥的含义，好像对应这个synchronize这个单词，这个单词被翻译成中文是“同步”，但是英文里说到sync这个概念的时候却在表达你文中所说的“互斥”的概念。你对并发的三个概念划分表示认同，但是中文的叫法，和之前看过的资料有一些不同，望老师指点。

2019-03-04

作者回复

同步在不同的语境里，意思完全不一样。

函数调用有同步调用、异步调用；

并发里，稍微专业的说法是指线程间的通信

synchronized也被翻译成同步关键字。同步有时候是线程安全的代名词，总有人说“这个方法没有正确同步”；

所以没有必要纠结他纠结这个。不同语境能正确理解就行了。

2019-03-04

才才

2

问个问题，分工，同步这俩个概念感觉相似，有本质区别吗

2019-02-26

作者回复

分工主要是拆分任务，要找到瓶颈，设计如何用多线程解决，这个偏设计。同步主要是线程间如何通信，这个偏实现。例如网络io有瓶颈，你可以用一连接一线程来分工，也可以用一组线程监听事件，一组线程处理事件来分工。前一种是不需要同步的，线程间不需要通信。但是后一种需要，因为两组线程要通信

2019-02-26