fcp-jj-
mark:3024:0:0:0:q75.awebp#?w=782&h=551&s=62446&e=png&b=fffefe)

毕竟是大版本更新，New Features 可以说是非常的多，一眼望去好几十个，鼠标都得划好几下。

心想这么多新特性，得学到啥时候去啊。

![](https://p3-juejin.byteimg.com/tos-cn-i-
k3u1fbpfcp/e62999483a9141a1bd71f45d0a3af0e2~tplv-k3u1fbpfcp-jj-
mark:3024:0:0:0:q75.awebp#?w=440&h=391&s=269436&e=png&b=c8b8
cb)

突然划到看到这个时候，我眼睛都直了：

![](https://p3-juejin.byteimg.com/tos-cn-i-
k3u1fbpfcp/fb99d51428bc4fb58c14569015a0ce4a~tplv-k3u1fbpfcp-jj-
mark:3024:0:0:0:q75.awebp#?w=1034&h=235&s=33883&e=png&b=ffffff)

在服务启动时，异步初始化 beans。

不是说好不支持吗？怎么突然变卦了呢？

于是我点到这个 New Features 后面的链接，准备一探究竟：

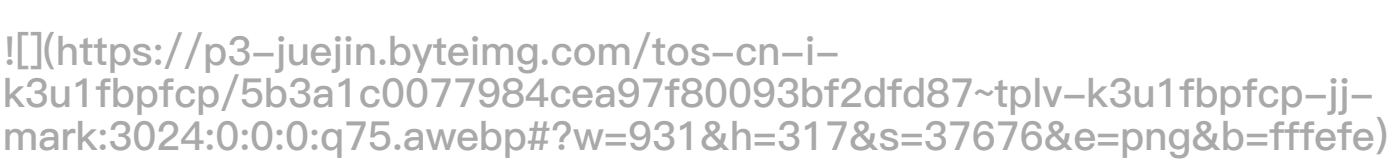
> [github.com/spring-proj...](http://cxyroad.com/
”https://github.com/spring-projects/spring-framework/issues/19487”)

![](https://p3-juejin.byteimg.com/tos-cn-i-
k3u1fbpfcp/5d524eb0c7234f89b9d4ffe8337852f8~tplv-k3u1fbpfcp-jj-
mark:3024:0:0:0:q75.awebp#?w=937&h=657&s=71488&e=png&b=ffffff)

这个 issue 是 2016 年提出来的，提问的这个哥们给出了一个自己实际的案例

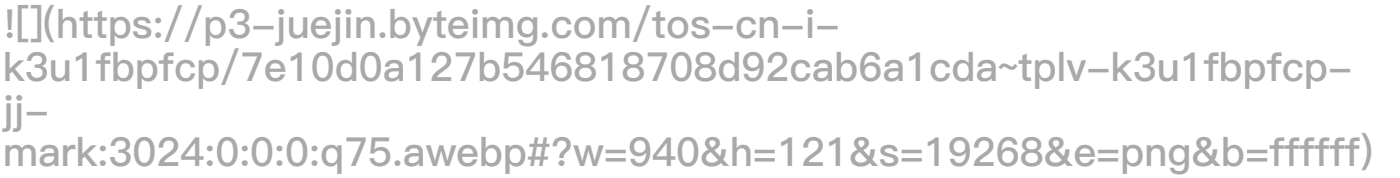
，然后还是想要官方能够支持 Bean 的异步初始化。

在今年 2 月的时候，这个下面有一个官方回答：



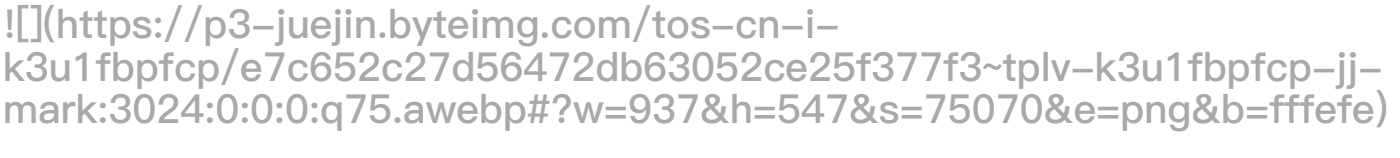

把链接指引到了 13410 这个 issue 里面。

而 13410 就是我们前面提到的这个 2011 年提出的 issue：




所以兜兜转转，还是回到了最开始的地方。

两年过去了，这个问题下最新的回答是 2024 年 2 月 28 日，也是来自官方的回答：




这个回答可以说非常关键了，是整个 Bean 的异步初始化的实现思路，我带你盘一下关键点，强烈建议你自己去看看，并且根据这部分的描述找到对应的代码。

在这个回答里面提到说会引入 backgroundInit 标识，以及在 @Bean 里面加入 bootstrap=BACKGROUND 枚举，通过这样的方式来支持 Bean 的异步初始化。

会在 preInstantiateSingletons 方法中，覆盖每个加了 BACKGROUND 的 Bean 的整个 getBean 步骤。

因为是异步处理，相应的 Future 会存储起来，这样依赖的 Bean 就会自动等待 Bean 实例完成。

此外，所有常规的后台初始化都会在 preInstantiateSingletons 结束时强制完成。只有被额外标记为 @Lazy 的 Bean 才允许稍后完成（直到第一次实际访问）。

最后这个回答中还强调了一点：因为是异步化操作，所以项目中还需要搞一个叫做 bootstrapExecutor 的线程池，来支持这个事情。

没有，那就异步化不了。

尝鲜
--

气氛都烘托到这里了，那高低得给你整一个 Demo 跑跑才行啊。

目前 Spring 6.2.0 版本还没正式发布，最新的 SpringBoot 里面也还没有集成 Spring 6.2.0 版本。

所以我们不能通过新建一个 SpringBoot 项目来尝鲜，得搞一个纯粹的 Spring 项目。

没想到歪师傅写到这里的时候遇到了一个卡点：怎么去创建一个 Spring 项目来着?

![](https://p3-juejin.bytei ebug”的表现之一。

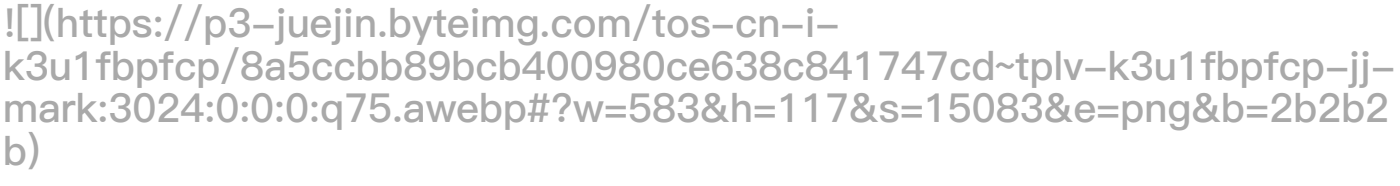
然后点过去一看，是要从 beanFactory 里面拿出一个叫做 bootstrapExecutor 的 bean 放进去：

bootstrapExecutor，是写死在源码里面的，所以你换另外一个 xxxExecutor，源码也不识别啊。

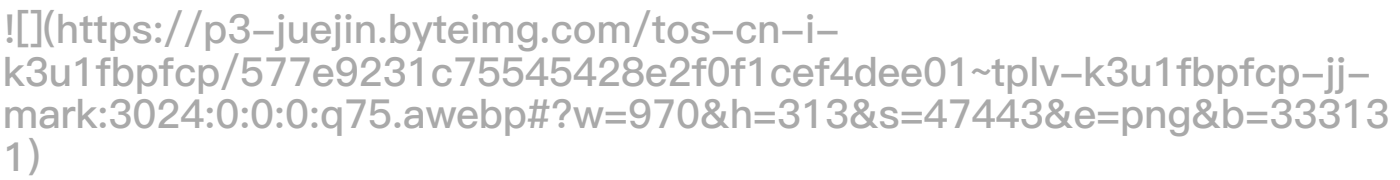
另外一个方式就是正向去找。

首先我们知道 BACKGROUND 是我们的一个“抓手”，而这个抓手在源码中也只有一个地方被调用：

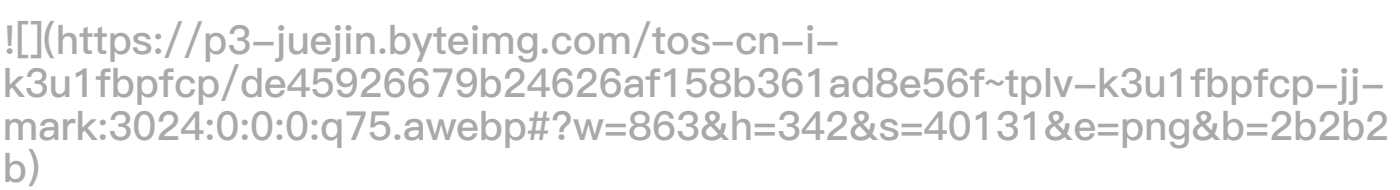

点过去之后发现这里是把 backgroundInit 设置为 true：




然后看 backgroundInit 标识被使用的地方：




又可以找到这里来：




这不就和前面呼吁上了。

这部分真的是太好 debug 了，我不骗你，你自己玩去吧。

思考
--

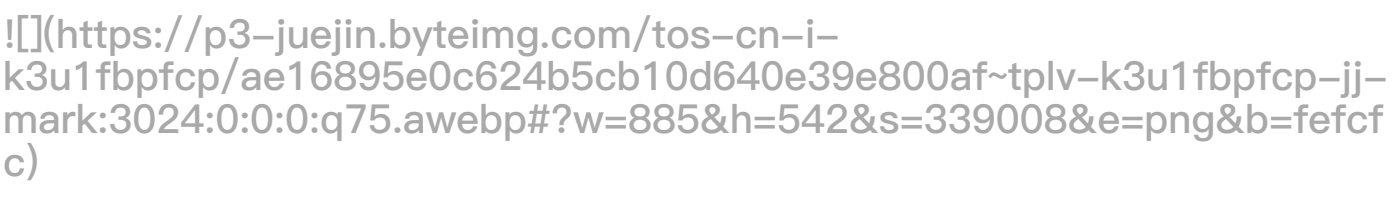
在大概摸清楚具体实现之后，歪师傅开始思考另外一个问题：Spring 为什么要支持 Bean 的异步初始化?

异步化，核心目标是为了加速项目启动，减少项目启动时间嘛。

按照官方最开始的说法，项目启动慢，应该是用户找到启动慢的根本原因，而不是想着异步化这个治标不治本的方法。

比如在前面的 issue 里面，有个老哥说：我这边有个应用启动花了 2 小时 30 分...

在 2011 年，官方是这样回复的：




他们的核心观点还是：在 Spring 容器中并行化 Bean 初始化的好处对于少数使用 Spring 的应用程序来说是非常重要的，而坏处是不可避免的 Bug、增加的复杂性和意想不到的副作用，这些可能会影响所有使用 Spring 的应用程序，恐怕这不是一个有吸引力的前景。

言外之意就是：我不改。

官方希望看到的是用户去寻找启动慢的真正原因。

用户希望的是官方提供一个异步化的方法先来解决当前的问题。

官方和用户都知道这是一个治标不治本的方案。

官方觉得没有必要，或者“太 low”，这样的代码不应该出现在我们的项目中，因为用户没有按照我的预期去使用对应代码。

用户觉得我不管治标还是治本，只要能解决问题就行。

这个时候就出现了分歧。

这个分歧甚至长达 13 年之久。在这期间官方和用户反复拉扯，都难以达成一致。

终于，在 6.2 版本里面，官方还是妥协了，Bean 的异步初始化终于还是落地了。

13 年的时间已经足够长了，长到 Spring 的用户群体已经爆炸式的增长，官方不得不足够重视用户反复提起的需求。

即使这个需求在官方看来是不合理的，这个解决方案看起来是不优雅的，但是由于用户需要，所以不得不提供。

你看这个场景像不像是你在工作中接到了一个自认为不合理的需求，但是却不得不去实施一样。

或者像不像在你精心搭建的系统中，必须加入一坨你觉得很难接受的代码。

就像你刚刚开始工作的时候，甚至有一点代码洁癖。

然后随着需求的叠加、时间的推移、日复一日的重复之后，开始变成“又不是不能用”。

没关系，都是会变的。