Spring6.2最新特性

1. 支持loc 容器 bean 异步并行的创建

这可以显着减少启动时间。

https://github.com/spring-projects/spring-framework/issues/13410

1.1. 使用方式

1. 配置需要异步创建的bean

```
@Bean(bootstrap = Bean.Bootstrap.BACKGROUND)
```

1. 配置线程池

```
@Bean
public Executor bootstrapExecutor(){
    return  Executors.newCachedThreadPool();
}
```

1.2. 测试：

1.2.1. 批量注册异步bean

```
@Component
public class BatchBeanDefinitionRegister implements BeanDefinitionRegistryPostProcessor, BeanFactoryAware {
    @Override
    public void postProcessBeanDefinitionRegistry(BeanDefinitionRegistry registry) throws BeansException {
        for (int i=0;i<10;i++){
            RootBeanDefinition rootBeanDefinition = new RootBeanDefinition(AService.class);
            rootBeanDefinition.setBackgroundInit(true);
            registry.registerBeanDefinition("AService"+i,rootBeanDefinition);
        }
    }
```

1.2.2. 为了更好演示让bean睡几秒

```
public class AService {
    @Autowired
    BService bService;

    public AService() throws InterruptedException {
        Thread.sleep(5000);
    }
}
```

1.2.3. 可以分别测试用bootstrapExecutor和不用的用时时间

```
public static void main(String[] args) {
    long beginTime = System.currentTimeMillis();
    AnnotationConfigApplicationContext ioc = new AnnotationConfigApplicationContext(Main2.class);
    long endTime = System.currentTimeMillis();
    System.out.println(endTime-beginTime);
    AService aService = (AService) ioc.getBean("AService9");
    System.out.println(aService);
}
```

可以发现如果bean很多， 确实速度提升了~!!!✿✿ヽ(°▽°)ノ✿

源码：

org.springframework.beans.factory.support.DefaultListableBeanFactory#preInstantiateSingleton

涉及到CompletableFuture不会可以参考：https://www.processon.com/view/link/6638e5d996857d67d2e3e998?cid=6618dbe5c04c1e02ff4f0754

1.3. 问题

1.3.1. 问题1： 如果异步beanA正在通过ioc容器加载创建， 此时另外一个线程2获取beanA.会怎么样?

线程2会阻塞吗?

在异步beanA创建时， 会往三级缓存中加入future.join();，

```
addSingletonFactory(beanName, () -> {
                                        try {
                                                future.join();
                                        }
                                        catch (CompletionException ex) {
ReflectionUtils.rethrowRuntimeException(ex.getCause());
                                        }
                                        return future;  // not to be exposed, just to lead to ClassCastException in case of mismatch
                                });
```

线程2在获取bean时， 会等待异步beanA线程执行完毕， 再去一级缓存中捞一遍

```
ObjectFactory<?> singletonFactory = this.singletonFactories.get(beanName);
    if (singletonFactory != null) {
        // 如果backgroundInit beanA正在创建， 此时有其他线程获取beanA或回调future.join 等待
        singletonObject = singletonFactory.getObject();
        // Singleton could have been added or removed in the meantime.
        if (this.singletonFactories.remove(beanName) != null) {
            this.earlySingletonObjects.put(beanName, singletonObject);
        }
        else {
            singletonObject = this.singletonObjects.get(beanName);
        }
    }
```

spring5的时候， 源码在创建的时候会上锁， 按道理第2个线程获取不到锁啊。 其实 ...

在spring6为了支持并行createBean， 不再进行上锁了，所以getsingleton中2个线程都可以trylock

tryLock()返回值表示的是用来尝试获取锁：成功获取则返回true；获取失败则返回false，**这个方法无论如何都会立即返回**。不会像synchronized一样，一个线程获取锁之后，其他锁只能等待那个线程释放之后才能有获取锁。

1.3.2. 问题2：如果同一个Bean懒加载一起创建会怎么样?

这个时候不会往三级缓存存入 join 。 而且也不会阻塞

为了支持并行createBean， 在创建bean时不再使用互斥锁

```
// 创建前标记当前bean正在创建
// 由于现版本并发创建允许并行， 所以可能存在同一个bean在正在创建， 其他线程报错， 以前版本是不存在的因为以前是互斥锁
// 但是这种情况极少  因为很少实际情况级别不会一起创建一个bean,除非程序员人为
// 相当于悲观锁（之前互斥等待）改乐观锁（现在不互斥比较替换，异常跳出）了。
beforeSingletonCreation(beanName);
```

在标记正在创建的bean时会报错:

```
if (!this.inCreationCheckExclusions.contains(beanName) &&
!this.singletonsCurrentlyInCreation.add(beanName)) {
                        throw new BeanCurrentlyInCreationException(beanName);
                }
```

相当于悲观锁（之前互斥等待）改乐观锁（现在不互斥比较替换，异常跳出）了。

1.3.3. 问题3： 如果BeanA\BeanB 都是异步lazy bean， 并且循环依赖， 2个线程，线程1 getBean(A) 线程2 getBean(B) ， 会互相join（死锁）？？？？？

不会， 因为在创建bean的时候， 实例化后会把三级缓存给覆盖掉！

2. defaultCandidate

https://github.com/spring-projects/spring-framework/issues/26528

当声明了bean, 不想被依赖注入， 可以设置该属性

解决问题：

当声明了bean， 有些时候不想被DI， 以前做不到，现在可以：

```
@Bean(defaultCandidate = true)
```

Spring-AI

完整示例代码

https://gitee.com/xscodeit/ai-openai-examples.git

官网：

https://docs.spring.io/spring-ai/reference/index.html

整合chatgpt

前置准备

1. open-ai-key:

https://api.xty.app/register?aff=PuZD

https://eylink.cn/

淘宝搜： open ai key

2. 魔法软件， 由于国家相关法律规定，建议大家自行准备。

实现

创建完后会发现加入了依赖：

```xml
<dependencyManagement>
    <dependencies>
       <dependency>
          <groupId>org.springframework.ai</groupId>
          <artifactId>spring-ai-bom</artifactId>
          <version>1.0.0-SNAPSHOT</version>
          <type>pom</type>
          <scope>import</scope>
       </dependency>
    </dependencies>
</dependencyManagement>

<dependencies>
    <dependency>
       <groupId>org.springframework.boot</groupId>
       <artifactId>spring-boot-starter-web</artifactId>
    </dependency>


    <dependency>
       <groupId>org.springframework.ai</groupId>
       <artifactId>spring-ai-openai-spring-boot-starter</artifactId>
    </dependency>

    <dependency>
       <groupId>org.springframework.boot</groupId>
       <artifactId>spring-boot-starter-test</artifactId>
       <scope>test</scope>
    </dependency>

    <dependency>
       <groupId>org.projectlombok</groupId>
       <artifactId>lombok</artifactId>
    </dependency>

</dependencies>
```

设置代理, 如果你请求的大模型的api接口不是国内的。需要将程序设置代理:

```
 public static void main(String[] args) {
    // 设置代理
    String proxy = "127.0.0.1";  // 如果代理在你本机就127.0.0.1 ,   如果代理是其
    他服务器相应设置
    int port = 7890;   //设置科学上网代理的端口,
    System.setProperty("proxyType", "4");
    System.setProperty("proxyPort", Integer.toString(port));
    System.setProperty("proxyHost", proxy);
    System.setProperty("proxySet", "true");

    SpringApplication.run(Application.class, args);

}
```

设置key:

```
spring:
  ai:
    openai:
      api-key: ${OPEN_AI_KEY}
      base-url: ${OPEN_AI_URL}
```

示例代码:

```
1
2  private final ChatClient chatClient;
3  private final OpenAiChatModel chatClient2;
4  private final OpenAiImageModel imageClient;
5  private final OpenAiAudioTranscriptionModel audioClient;
6  private final  OpenAiAudioApi openAiAudioApi;
7
8
9  @Value("${OPEN_AI_KEY}")
10 private String openAiKey;
11
12
13 @GetMapping("/ai/simple")
14 public Map<String, String> completion(@RequestParam(value = "message", defaultValue =
   "给我讲个笑话") String message) {
15     System.out.println(openAiKey);
16     var value=chatClient.prompt()
17           .user(message).call().content();
18     return Map.of("generation",value );
19 }
20
21 @GetMapping(value="/ai/stream",produces="text/sse;charset=UTF-8")
22 public Flux<String> stream(@RequestParam(value = "message", defaultValue = "给我讲个笑
   话") String message ) {
23     System.out.println(openAiKey);
24     return chatClient.prompt()
25           .user(message)
26           .stream()
27           .content();
28 }
29
30
31 @GetMapping(value="/ai/img",produces="text/html")
32 public String image(@RequestParam(value = "message", defaultValue = "猫") String
   message ) {
33
34     ImageResponse response = imageClient.call(
35           new ImagePrompt(message,
```

```
                OpenAiImageOptions.builder()
                        .withQuality("hd")
                        .withN(1)
                        .withModel(OpenAiImageApi.ImageModel.DALL_E_2.getValue())
                        // dall-e-2 256
                        .withHeight(256)
                        .withWidth(256).build()));

    String url = response.getResult().getOutput().getUrl();
    System.out.println(url);
    return "<img src='"+url+"'/>";

}


@GetMapping(value="/ai/audit2text")
public String audit2text() {

    var transcriptionOptions = OpenAiAudioTranscriptionOptions.builder()
          .withResponseFormat(OpenAiAudioApi.TranscriptResponseFormat.TEXT)
          .withTemperature(0f)
          .build();

    // flac、mp3、mp4、mpeg、mpga、m4a、ogg、wav 或 webm。
    var audioFile = new ClassPathResource("/hello.mp3");

    AudioTranscriptionPrompt transcriptionRequest = new
AudioTranscriptionPrompt(audioFile, transcriptionOptions);
    AudioTranscriptionResponse response = audioClient.call(transcriptionRequest);

    //openAiAudioApi.createTranscription()
    return response.getResult().getOutput();
}


@GetMapping(value="/ai/text2audit")
public String text2audit() {

```

```
    ResponseEntity<byte[]> speech = openAiAudioApi.createSpeech(
          OpenAiAudioApi.SpeechRequest.builder().
                withVoice(OpenAiAudioApi.SpeechRequest.Voice.ONYX).
                withInput("你好，我是徐庶").build());
    byte[] body = speech.getBody();



    // 将byte[]存为 mp3文件
    try {
       writeByteArrayToMp3(body, System.getProperty("user.dir"));
    } catch (IOException e) {
       throw new RuntimeException(e);
    }

    return "ok";
}

public static void writeByteArrayToMp3(byte[] audioBytes, String outputFilePath) throws
 IOException {
    // 创建FileOutputStream实例
    FileOutputStream fos = new FileOutputStream(outputFilePath+"/xushu.mp3");

    // 将字节数组写入文件
    fos.write(audioBytes);

    // 关闭文件输出流
    fos.close();
}


@GetMapping(value="/ai/mutil")
public String mutilModel(String message,String imgUrl) throws IOException {

    byte[] imageData = new ClassPathResource("/test.png").getContentAsByteArray();

    var userMessage = new UserMessage(
          "这个图片你看出什么?", // content
          List.of(new Media(MimeTypeUtils.IMAGE_PNG, imageData))); // media

```

```
    ChatResponse response = chatClient.call(new Prompt(userMessage,
          OpenAiChatOptions.builder()
                .withModel(OpenAiApi.ChatModel.GPT_4_TURBO_PREVIEW.getValue())
                .build()));

    return  response.getResult().getOutput().getContent();
}

```

function-call

AI本身是不具备实时消息能力的， 比如问“现在北京的天气是什么”， AI是不知道的， 这个时候我们需要通过接口来帮助AI完成,大致流程：

function-call实现的代码：

1. 在发送文本时， 同时设置Funcation； 关键代码： `.withFunction("getWaitTime")`

```
ChatResponse response = chatClient.call(new Prompt(
        List.of(userMessage),
        OpenAiChatOptions.builder()
                .withFunction("getWaitTime")
                .build()));
```

2. 当代码执行完call时（AI响应之后）， 会再调用 `getWaitTime` 对应的bean的apply方法。

```
@Bean
public Function<WaitTimeService.Request, WaitTimeService.Response> getWaitTime() {
    return new WaitTimeService();
}
```

3. 并且会把 `getWaitTime` 该bean的Request作为funcation-call的返回参数， 即可在 `apply` 方法中获取Request对应的参数

```java
@Override
public Response apply(Request request) {
    String name =  request.name();
    String location =  request.location();
     // todo...
    return new Response(location+"有10个！");
}

public record Request(String name,String location) {}
public record Response(String weather) {}
```

随后Response会再次丢给AI组织语言， 进行响应，最终

源码：

看源码之前， 了解下该接口需要哪些参数：https://platform.openai.com/docs/api-reference/chat/create

请求： userMessage=“长沙有多少叫徐庶的”

调用call方法， 执行openai的远程api请求

```java
@Override
public ChatResponse call(Prompt prompt) {

    ChatCompletionRequest request = createRequest(prompt, false);

    return this.retryTemplate.execute(ctx -> {

        ResponseEntity<ChatCompletion> completionEntity =
this.callWithFunctionSupport(request);

        var chatCompletion = completionEntity.getBody();
        if (chatCompletion == null) {
            logger.warn("No chat completion returned for prompt: {}", prompt);
            return new ChatResponse(List.of());
        }

        RateLimit rateLimits =
OpenAiResponseHeaderExtractor.extractAiResponseHeaders(completionEntity);

        List<Generation> generations = chatCompletion.choices().stream().map(choice -> {
            return new Generation(choice.message().content(), toMap(chatCompletion.id(),
choice))

.withGenerationMetadata(ChatGenerationMetadata.from(choice.finishReason().name(),
null));
        }).toList();

        return new ChatResponse(generations,

OpenAiChatResponseMetadata.from(completionEntity.getBody()).withRateLimit(rateLimits));
    });
```

2. 通过 createRequest 将withFunction参数解析到request.tools

```java
ChatCompletionRequest request = createRequest(prompt, false);
```

tools属性：包含了

i. 函数名，也是需要调用 的bean的name：getWaitTime

ii. properties是告诉AI，要返回的向量的名字

iii. @Description

iv. 函数描述，函数功能的描述，模型使用它来选择何时以及如何调用函数。

关于open-ai对tools参数的说明：https://platform.openai.com/docs/api-reference/chat/create#chat-create-tools

---

3. 来到 `callWithFunctionSupport` 真正调用远程api接口

```
protected Resp callWithFunctionSupport(Req request) {
    Resp response = this.doChatCompletion(request);
    return this.handleFunctionCallOrReturn(request, response);
}
```

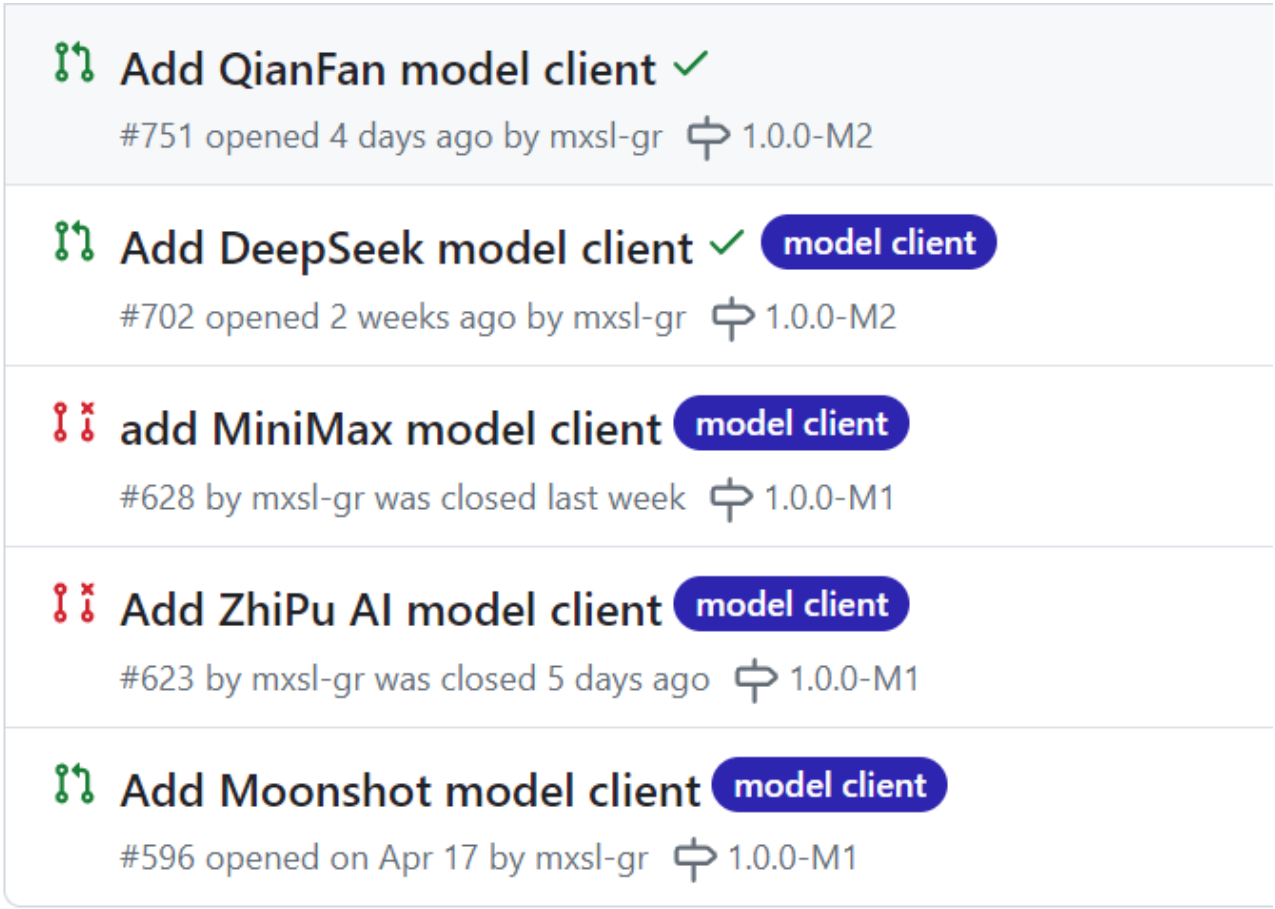
这里2句关键代码：

1. `this.doChatCompletion(request);` 方法会正常请求**chat completion** 接口并且会带上funcation-call参数并携带tools属性， 并且返回对话中的funcation-call所需参数（即 `WaitTimeService.Request` 的参数）
2. `handleFunctionCallOrReturn` 执行Function-callback方法, 此时会调用 `WaitTimeService.apply` 方法
   a. 拿到之前解析的functionCallback即
   b. 将arguments从Json转换为对象调用 `WaitTimeService.apply`
   c. 将返回的数据再次请求大模型

流程图：
https://www.processon.com/view/link/6654257b40e88034a42b837c?cid=6652ddc65a676876a64350e5

SpringAI社区非常活跃

在后续的版本都会更新国内常用的大模型
https://github.com/spring-projects/spring-ai/issues?q=Moonshot



有道云链接：https://note.youdao.com/s/9t8IUhiR