Debug từ @SpringBootApplication

Đặt debug ở @SpringBootApplication

Dòng thứ 318 của lớp SpringApplication thực hiện hàm run(String… arg)

Dòng thứ 395 của lớp SpringApplication thực hiện hàm prepareContext() trong đó thực hiện các hàm :

* postProcessApplicationContext – dòng thứ 597
* Đăng ký bean name generator theo các tên được cấu hình trong @Configuration. Sử dụng để tạo tên cho các bean
* Đăng ký resourceLoader cho context theo 2 kiểu : DefaultResourceLoader và GenericApplicationContext. Tải các lớp .class từ bộ nhớ để JVM thực thi.
* Đăng ký conversion service cho bean factory theo environment. Sử dụng để chuyển đổi giữa các kiểu dữ liệu
* addAotGeneratedInitializerIfNecessary – dòng thứ 400
* Tạo danh sách A head of Time của các Context Initializers, đảm bào các AoTContextInitializers – tối ưu hóa hiệu suất khởi động luôn được chạy trước các ApplicationContextInitializer – thay đổi trạng thái, cấu hình
* applyInitializers – dòng thứ 622
* Áp dụng bất kỳ ApplicationContextInitializers nào cho context trước khi nó được làm mới.

Cũng có thể tham khảo hàm refresh của lớp ConfigurableApplicationContext – là phương thức khởi động, áp dụng các cấu hình từ schema, dựa vào Java, XML, tệp thuộc tính. Nó hủy các singleton đã được tạo nếu có lỗi.

Thông báo context đã được chuẩn bị xong, đóng bootstrapContext và ghi log khởi động nếu cần.

Dòng thứ 431 của hàm prepareContext() của lớp SpringApplication thực hiện hàm load()

Dòng thứ 699 của hàm load() của lớp SpringApplication thực hiện hàm load()

Dòng thứ 127 của hàm load() của lớp BeanDefinitionLoader thực hiện load(source).

Dòng thứ 144 của hàm load() của lớp BeanDefinitionLoader gọi tới hàm load() của lớp BeanDefinitionLoader, thực hiện hàm scan().

Dòng thứ 252 của hàm doScan của lớp ClassPathBeanDefinitionScanner thực hiện chức năng quét và tạo tên các bean (Component scan). Tại thời điểm 14/8/2024 có 31 bean được khởi tạo.

Sau đó, tại dòng thứ 68 của hàm scan() của lớp ComponentScanAnnotationParser trả về kết quả là 1 set chứa tên các bean được quét qua.

Tại dòng thứ 306 của hàm doProcessConfigurationClass của lớp ConfigurationClassParser sẽ trả về kết quả là 1 set chứa tên các bean được quét bởi các package được đánh dấu bằng @ComponentScan.

Dòng thứ 410 và 412 đăng ký singleton bean cho hai bean lớn là springApplicationArguments – thực hiện tiêm thuộc tính khi khởi tạo bean A với thông tin –server.port vào trong 1 bean khác thông qua class ApplicactionArguments và springBootBanner – in ra tên của ứng dụng khi chạy.

Đăng ký các bean khác :

org.springframework.beans.factory.BeanFactory

Dòng thứ 15 interface BeanFactory : getBean(String name)

Dòng thứ 130 abtract AbstractBeanFactory : doGetBean(String name, …)

* Lấy được bean name
* Không lấy được shared Instance ?

Dòng thứ 270 abstract AbstractBeanFactory : doGetBean()

* Kiểm tra và không tìm thấy bean definitions tồn tại trong factory
* Thực hiện hàm markBeanAsCreated() dòng thứ 1760 sử dụng synchronized ? \*để đồng bộ các yêu cầu khởi tạo bean

Dòng thứ 300 abstract AbstractBeanFactory đã khởi tạo được RootBeanDefinition với các thuộc tính như sau

scope=singleton;

abstract=false;

lazyInit=null;

autowireMode=0;

dependencyCheck=0;

autowireCandidate=true;

primary=false;

factoryBeanName=null;

factoryMethodName=null;

initMethodNames=null;

destroyMethodNames=null

Dòng thứ 334 abstract AbstractBeanFactory : tạo instance cho bean. Kiểm tra và thấy rootBeanDefinition là singleton

Dòng thứ 403 abstract AbstractBeanFactory thực hiện hàm adaptBeanInstance (String,Object object, requiredType) ?

Dòng thứ 393 BeanDefinitionValueResolver thực hiện kiểm tra tên bean cung cấp có phải là duy nhất hay không. Nếu nó chưa là duy nhất, tăng biến đếm cho tới khi tên là duy nhất ?

Dòng thứ 341 thực hiện hàm resolveReference kiểm tra bean có tồn tại hay không bằng hàm ref.isToParesn. Nếu là bean cha, tìm kiếm bean cha trong factory

Nếu không, tìm trong factory hiện tại

Dòng thứ 365 thực hiện hàm getBean(resolveName) trong lớp BeanDefinitionValueResolver để lấy bean.

Dòng thứ 367 thực hiện hàm registerDependentBean(resolveName, beanName) trả về thông tin sau

Dòng thứ 369 kiểm tra bean được tạo có phải kiểu NullBean hay không ?, nếu có thì gán bean = null.

Dòng thứ 131 của lớp BeanDefinitionValueResolver thực hiện hàm resolveValueIfNecessary() với mục đích kiểm tra và xử lý các giá trị bean khác nhau, đảm bảo bean có kiểu dữ liệu phù hợp trước khi được sử dụng.

Dòng thứ 665 của lớp DefaultListableBeanFactory thực hiện hàm getBeansOfType trả về một map chứa các cặp key – value bao gồm tên bean và kiểu dữ liệu của bean

Dòng thứ 1385 của lớp AbstractAutowireCapableBeanFactory thực hiện hàm populateBean. Trong hàm này ta sẽ kiểm tra beanwrapper. Trong hàm này ta có thể tiêm phụ thuộc dựa theo tên hoặc theo kiểu dữ liệu của bean, sử dụng AUTOWIRE\_BY\_NAME, AUTOWIRE\_BY\_TYPE.

Dòng thứ 553 của lớp AbstractAutowireCapableBeanFactory thực hiện hàm doCreateBean, tạo ra bean được chỉ định và trả về instance mới của bean.

Dòng thứ 556 tới dòng thứ 568 là hàm thực hiện gán instance cho bean. Kiểm tra nếu như instance của bean đã tồn tại trong cache chưa, nếu tồn tại thì lấy instance trong cache gán cho wrapper bằng hàm instanceWrapper = this.factoryBeanInstanceCache.remove(beanName);

Nếu chưa tồn tại trong cache thì khởi tạo 1 instance mới cho wrapper đó.

Sau cùng, lấy instance của wrapper gán cho bean.

Dòng thứ 570 tới dòng thứ 582 là hàm thực hiện xử lý tuần tự cho 1 bean toàn cục. Khi có nhiều hơn 1 luồng cùng truy cập và gọi tới 1 bean toàn cục thì ta cần xử lý đồng bộ cho nó. Tránh xử lý lại trong tương lai.

Dòng thứ 584 tới dòng thứ 594 là hàm thực hiện xử lý tham chiếu tuần hoàn. Trong khi khởi tạo 1 bean này, ta có thể gán instance của bean vào trong factory bean để các bean khác có thể truy cập và sử dụng bean đó. Ví dụ trong @Service A và @Service B có A và B quan hệ 1 nhiều hoặc nhiều nhiều tới nhau.

Dòng thứ 522 của lớp AbstractAutowireCapableBeanFactory thực hiện hàm createBean, trả về bean được tạo từ hàm doCreateBean

Dòng thứ 337 của lớp AbstractBeanFactory trả về kết quả dựa vào hàm createBean.

Dòng thứ 234 của lớp DefaultSingletonBeanRegistry thực hiện hàm getSingleton bằng singletonFactory.getObject()

Dòng thứ 335 của lớp AbstractBeanFactory gán giá trị của hàm getSingleton() cho sharedInstance. BeanInstance sẽ được gán bằng giá trị hàm getObjectForBeanInstance().

Việc lọc và sắp xếp các postProcessorName rất quan trọng, vì chúng được khởi tạo là 1 bean singleton, nên thứ tự khởi tạo ảnh hưởng trực tiếp đến việc cấu hình bean.

Dòng 342 của lớp ConfigurationClassParser có sự thay đổi về resource

Tìm hiểu về sự thay đổi của resource loader

Lấy ra bean kiểu gì ? sử dụng hàm getBean, lấy bean ra từ beanfactory

Trong hàm prepareContext , sau khi thực hiện close Bootstrap context, biến resourceLoader vẫn bị null.

kết thúc hàm prepare context, resourceloader vẫn bị null

context có 2 giá trị scanner ClassPathBeanDefinitionScanner dùng để quét các file được gán annotation @Component, @Service, @Repository và giá trị reader AnnotatedBeanDefinitionReader đăng ký các bean được định nghĩa thủ công trong code

Luồng trong code như sau :

1. Các listener được khởi tạo sẽ quét các thư viện ngoài và khởi tạo bean thông qua classpath. ở đây là META-INF/spring.factories . Debug ở dòng 326 của hàm run() ở lớp SpringApplication
2. ở trong listener có 1 field là ucp – URLClassPath là nơi chứa các path đã được quét bằng listener
3. …
4. đầu tiên đi từ hàm refreshContext(context) của lớp SpringApp dòng thứ 452
5. sau đó đi tới hàm refresh(context) của lớp SpringApp dòng thứ 753
6. sau đó đi tới hàm refresh() của lớp ServletWebServerApplicationContext dòng thứ 144
7. sau đó đi tới hàm refresh() của lớp AbstractApplicationContext dòng thứ 585 - tiền xử lý BeanDefinition

* hàm này thực hiện chuẩn bị context cho refresh. Tạo và chuẩn bị bean factory. Gọi các BeanFactoryPostProcessor đã đăng ký, đăng ký các BeanPostProcessor. Sau đó kiểm tra và đăng ký các beanlistener, khởi tạo các singleton còn lại. hoàn thành refresh

1. sau đó đi tới hàm invokeBeanFactoryPostProcessors của lớp AbtractApplicationContext dòng thứ 788

* thực hiện trước khi gọi tới khởi tạo singleton. Các bean trong BeanFactoryPostProcessor được khởi tạo theo thứ tự rõ ràng đã được chỉ định.

1. sau đó đi tới hàm invokeBeanFactoryPostProcessors của lớp PostProcessorRegistrationDelegate dòng thứ 68

* gọi và có thể thay đổi cấu hình bean trước khi được tạo, phân loại thành các loại giao diện như PriorityOrdered và Ordered

1. sau đó gọi tới hàm doGetBean của lớp AbtractBeanFactory dòng thứ 205

* tạo và trả về 1 bean trong spring bao gồm kiểm tra cache, tạo instance, xử lý các trường hợp đặc biệt như vòng tròn tham chiếu, scope của bean và xử lý ngoại lệ

1. sau đó gọi tới hàm getSingleton của lớp DefaultSingletonBeanRegistry dòng thứ 112

* hàm này sẽ trả về đối tượng singleton của bean bằng tên của nó, tạo và đăng ký singleton mới nếu nó chưa được đăng ký trước đó

1. sau đó gọi tới hàm createBean của lớp AbstractAutowireCapableBeanFactory dòng thứ 485

* tạo một instance của bean, điền dữ liệu vào instance của bean, áp dụng các post-processor

1. sau đó gọi tới hàm populateBean của lớp AbtractAutowireCapableBeanFactory dòng thứ 1385

* điền giá trị cho các thuộc tính của bean

1. sau đó gọi tới hàm applyPropertyValues của lớp AbtractAutowireCapableBeanFactory dòng thứ 1655

* áp dụng các giá trị được cung cấp cho các bean và xử lý liên kết tới các bean khác trong bean factory

1. sau đó gọi tới hàm resolveValueIfNecessary của lớp BeanDefinitionValueResolver dòng thứ 131

* giải quyết các giá trị trong quá trình tạo bean

1. sau đó gọi tới hàm resolveReference của lớp BeanDefinitionValueResolver dòng thứ 337

* xử lý giải quyết các liên kết tới các bean khác trong bean factory

1. sau đó gọi tới hàm getBean của lớp AbstractBeanFactory dòng thứ 200

Hàm getBean được gọi ra sau doScan. Hàm getBean có thể được gọi ra ở bất kỳ đâu, còn hàm doScan được gọi ra sau khi khởi tạo container, quét các component. Câu hỏi đặt ra :  
Classloader lưu ở đâu trong các phân vùng bộ nhớ của máy tính ? BeanFactory được lưu ở trong heap của máy, thuộc phần RAM, thực hiện chức năng lưu trữ và khởi tạo các bean khi được gọi tới, còn classloader thì sao ?

Heap của JVM là nơi phân bổ các đối tượng của java

Classloader cũng được lưu trong heap của máy

tại sao classloader được lưu ở ram mà không phải CPU ?

vì classloader cần truy cập liên tục vào các thông tin của các đối tượng cần truyền tải, và đảm bảo tính tạm thời – khi chạy phần mềm thì classloader chỉ truyền tải thông tin trong quá trình chạy, sau khi ứng dụng kết thúc thông tin này có thể bị xoá đi.

Cách load các thư viện ở ngoài vào trong như thế nào ? (mysql, các csdl, )

Mysql driver …