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

Lấy ra bean kiểu gì ?