new AnnotationConfigApplicationContext(TestConfig.class);

AnnotationConfigApplicationContext(Class<?>... annotatedClasses)

this();

this(); 初始化BeanDefinition容器和IOC容器

public AnnotationConfigApplicationContext() {}

public GenericApplicationContext() { this.beanFactory = new DefaultListableBeanFactory();}

public DefaultListableBeanFactory() {super();}

先加载父类构造器

public AbstractAutowireCapableBeanFactory() { super();

public AbstractBeanFactory() {}

FactoryBeanRegistrySupport

初始化IOC容器

DefaultSingletonBeanRegistry
private final Map<String, Object> singletonObjects = new ConcurrentHashMap<String, Object>(256);

FactoryBeanRegistrySupport
private final Map<String, Object> factoryBeanObjectCache = new ConcurrentHashMap<String, Object>(16);

AbstractBeanFactory
private final List<BeanPostProcessor> beanPostProcessors = new ArrayList<BeanPostProcessor>();

AbstractAutowireCapableBeanFactory
private final Map<String, BeanWrapper> factoryBeanInstanceCache = new ConcurrentHashMap<String, BeanWrapper>(16);
public AbstractAutowireCapableBeanFactory() { super();}

初始化BeanDefinition容

DefaultListableBeanFactory
private final Map<String, BeanDefinition> beanDefinitionMap = new ConcurrentHashMap<String, BeanDefinition>(256);
private volatile List<String> beanDefinitionNames = new ArrayList<String>(256);

this(); 注册BeanDefinition容器

public AnnotationConfigApplicationContext() { this.reader = new AnnotatedBeanDefinitionReader(this);}

public AnnotatedBeanDefinitionReader(BeanDefinitionRegistry registry) { this(registry, getOrCreateEnvironment(registry)); }

public AnnotatedBeanDefinitionReader(BeanDefinitionRegistry registry, Environment environment) { AnnotationConfigUtils.registerAnnotationConfigProcessors(this.registry); }

AnnotationConfigUtils
public static void registerAnnotationConfigProcessors(BeanDefinitionRegistry registry) { registerAnnotationConfigProcessors(registry, null);}

AnnotationConfigUtils#registerAnnotationConfigProcessors(org.springframework.beans.factory.support.BeanDefinitionRegistry, java.lang.Object)
beanDefs.add(registerPostProcessor(registry, def, CONFIGURATION_ANNOTATION_PROCESSOR_BEAN_NAME));

AnnotationConfigUtils#registerPostProcessor
registry.registerBeanDefinition(beanName, definition);

GenericApplicationContext#registerBeanDefinition

注册BeanDefinition容

DefaultListableBeanFactory#registerBeanDefinition
this.beanDefinitionMap.put(beanName, beanDefinition);
this.beanDefinitionNames.add(beanName);

register(annotatedClasses); 注册BeanDefinition容器

AnnotationConfigApplicationContext#register

AnnotatedBeanDefinitionReader#register

AnnotatedBeanDefinitionReader#registerBean

AnnotatedBeanDefinitionReader#registerBean(java.lang.Class<?>, java.lang.String, java.lang.Class<? extends java.lang.annotation.Annotation>...)

BeanDefinitionReaderUtils#registerBeanDefinitio
n

GenericApplicationContext#registerBeanDefinition

DefaultListableBeanFactory#registerBeanDefinition
this.beanDefinitionMap.put(beanName, beanDefinition);
this.beanDefinitionNames.add(beanName);

refresh();将对象实例放到IOC容器中

AbstractApplicationContext#finishBeanFactoryInitialization

ConfigurableListableBeanFactory#preInstantiateSingletons

AbstractBeanFactory#getBean(java.lang.String)

AbstractBeanFactory#doGetBean

DefaultSingletonBeanRegistry#getSingleton(java.lang.String, org.springframework.beans.factory.ObjectFactory<?>)
singletonObject = singletonFactory.getObject();

AbstractBeanFactory#createBean

AbstractAutowireCapableBeanFactory#doCreateBean

AbstractAutowireCapableBeanFactory#createBeanInstance

AbstractAutowireCapableBeanFactory#instantiateBean

InstantiationStrategy#instantiate(org.springframework.beans.factory.support.RootBeanDefinition, java.lang.String, org.springframework.beans.factory.BeanFactory)

BeanUtils#instantiateClass(java.lang.reflect.Constructor<T>, java.lang.Object...)

java.lang.reflect.Constructor#newInstance

将对象实例放到IOC容器中

DefaultSingletonBeanRegistry#addSingleton
protected void addSingleton(String beanName, Object singletonObject) {
synchronized (this.singletonObjects) {
this.singletonObjects.put(beanName, singletonObject );
this.singletonFactories.remove(beanName);
this.earlySingletonObjects.remove(beanName);
this.registeredSingletons.add(beanName);
}
}