Spring lesson 3

xieqiaoyun

Why Spring?

简化java开发环境

- 可以非侵入式扩展POJO
- 通过依赖注入、面向接口编程解耦
- 面向切面编程，分离关注点
- 通过切面和模版，消除多处重复的样板代码，提升组件可复用性

两个重要基础概念

依赖注入：

依赖第三方(如ApplicationContext容器)注入

AOP：

面向切面编程，分离关注点，提升组件可复用性

依赖注入

```java
public class AutoWiredBean {
    private String myName;
    private HelloBean helloBean;
    public AutoWiredBean(){
        System.out.println(new Date() + "created "+ this);
    }
    public void sayHello(){
        System.out.println(new Date() + " hellow "+myName +" @ "+this);
    }
    public void setMyName(String myName){
        this.myName = myName;
    }
    @Autowired
    public void setHelloBean(HelloBean helloBean){
        this.helloBean = helloBean;
    }
}
```

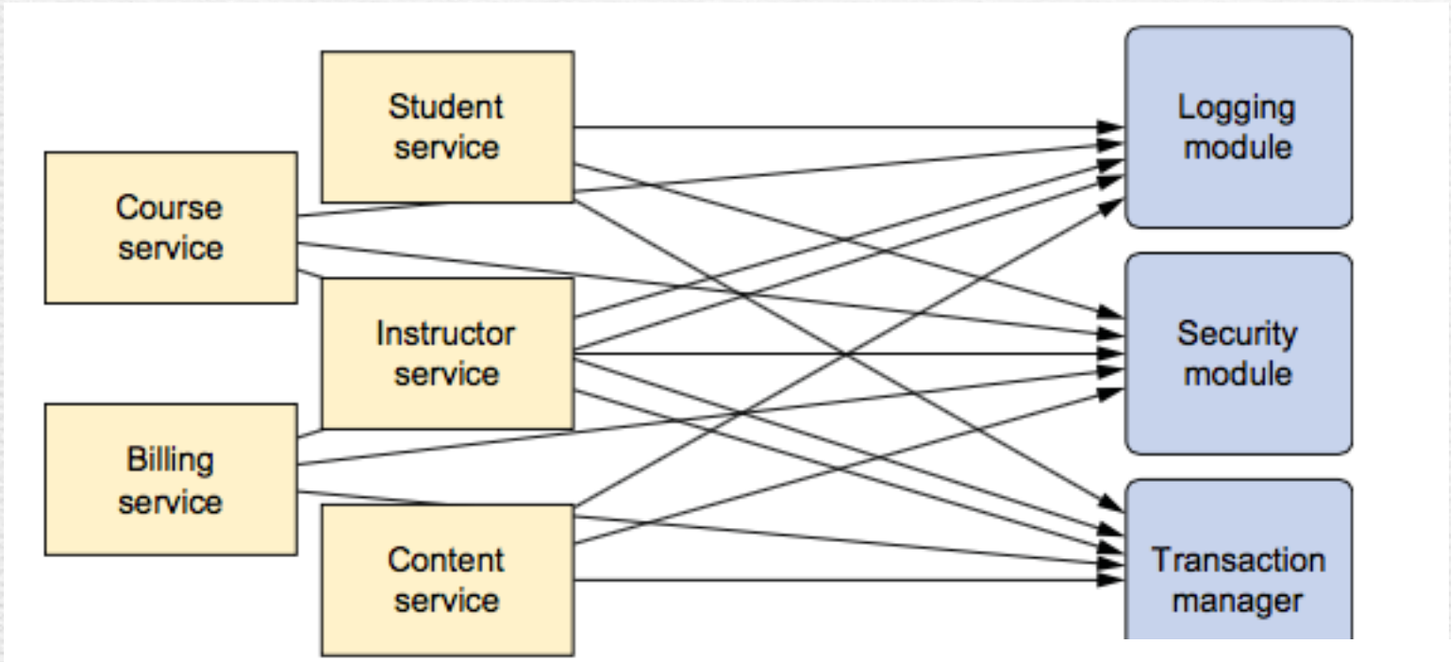
注入

AutoWiredBean

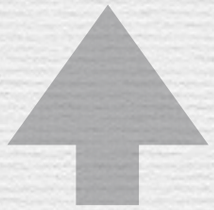
还可以通过

AOP

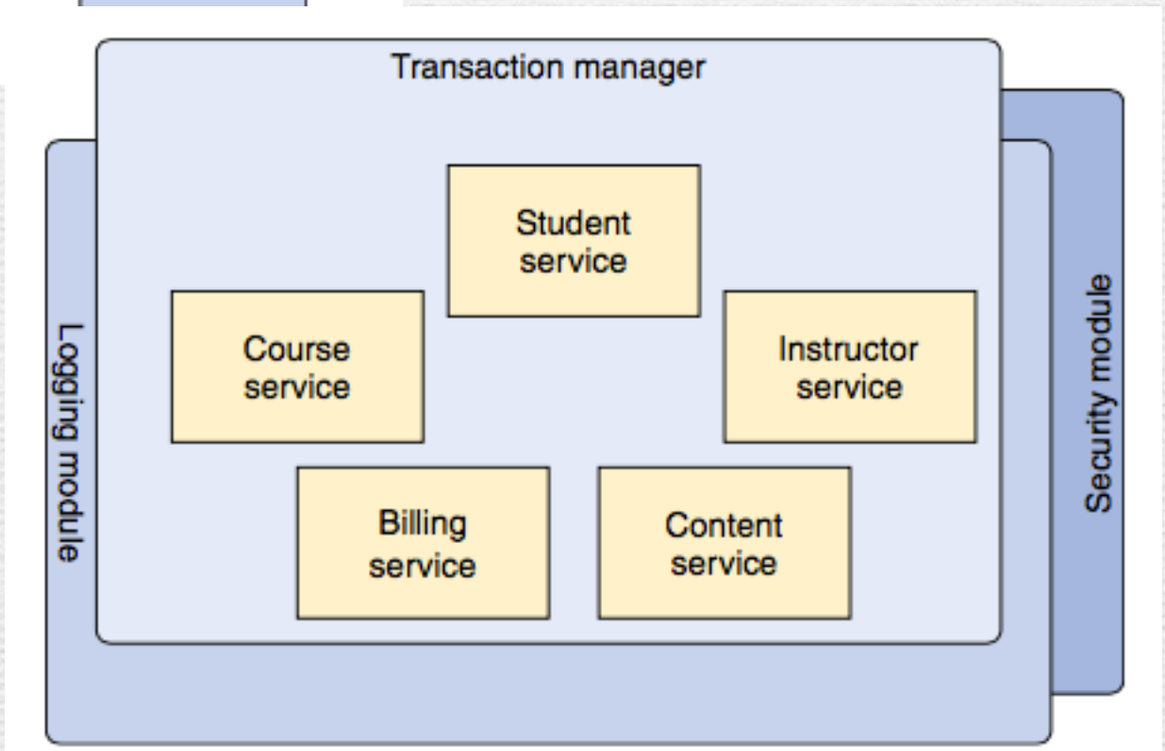
AOP增加了两个复杂度：横跨多个组件、修改原有方法核心逻辑



这么看明朗了。。。



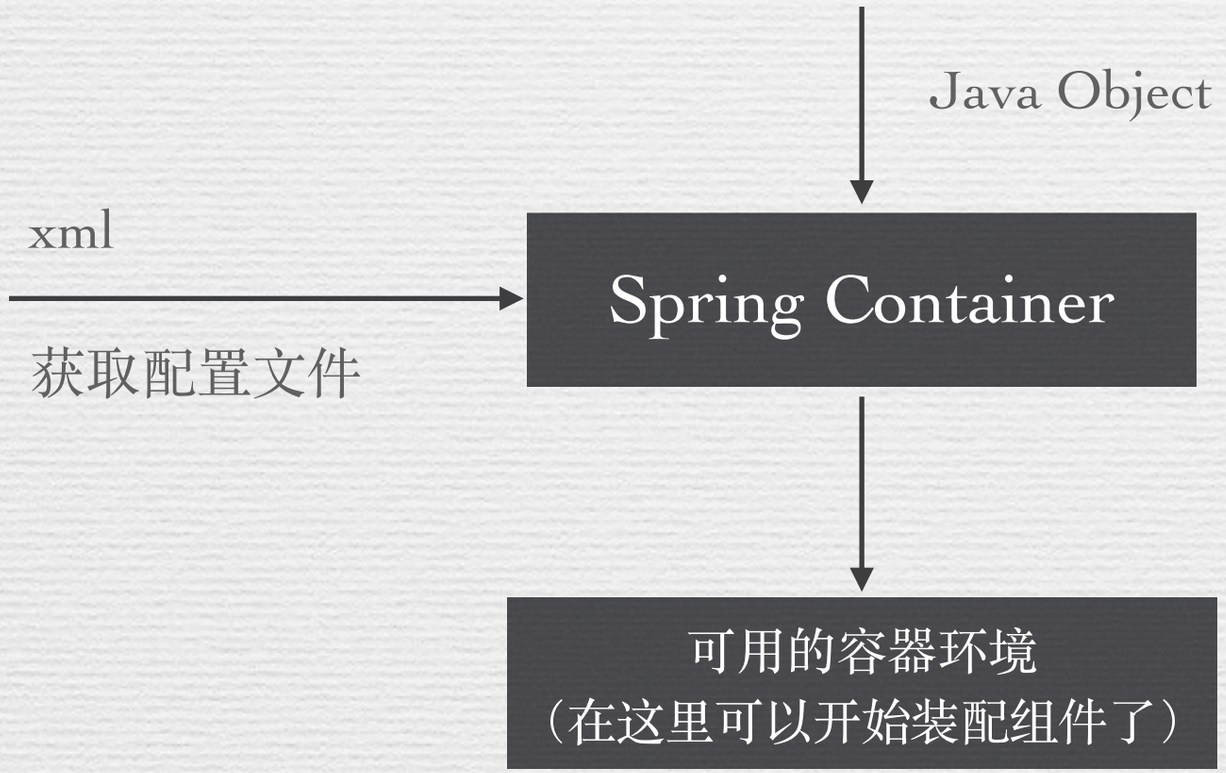
这么看很纠结。。。

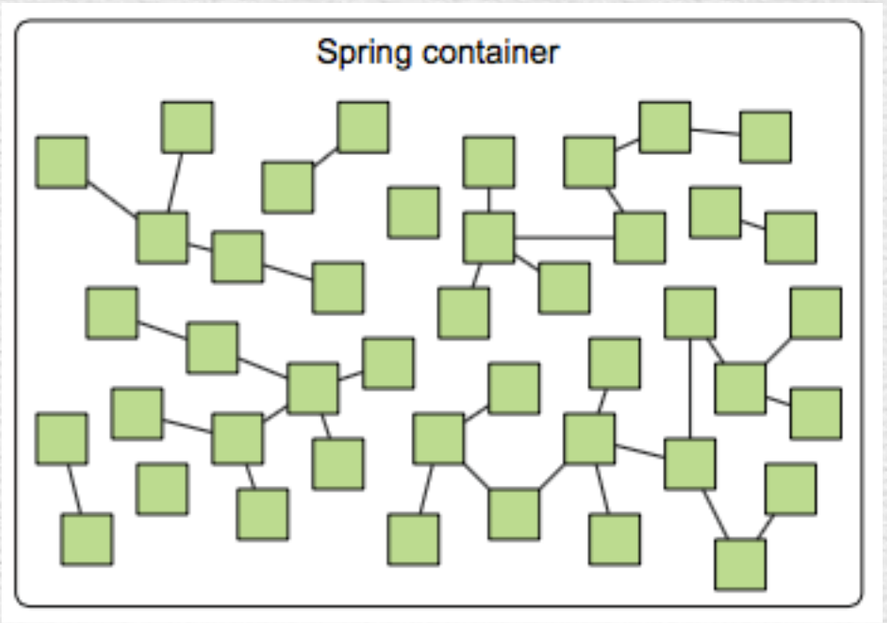


AOP

什么是容器

负责加载bean定义配置，并根据bean定义创建、装配bean（即创建它们之间的连接）





容器的五个实现类

- AnnotationConfigApplicationContext
- AnnotationConfigWebApplicationContext
- ClassPathXmlApplicationContext
- FileSystemXmlApplicationContext
- XmlWebApplicationContext

均实现ApplicationContext接口，加载不同Bean定义文件

容器初始化

通过加载XML bean定义

```java
public class HelloWorldBean {
    private String myName;
    public HelloWorldBean(){
        System.out.println(new Date() + "created '
    }
    public void sayHello(){
        System.out.println(new Date() + " hellow '
    }
    public void setMyName(String myName){
        this.myName = myName;
    }
}
```

```java
public class main {
    public static void main(String[] args) throws InterruptedException{
        System.out.println(new Date() + " begin");
        ApplicationContext ctx =
                new ClassPathXmlApplicationContext( configLocation: "application.xml");
        //ApplicationContext ctx =
        // new FileSystemXmlApplicationContext("hw03/src/main/resources/application.x

        HelloWorldBean helloWorldBean = (HelloWorldBean) ctx.getBean( s: "helloBean");
        System.out.println(new Date()+" get bean "+helloWorldBean);

        helloWorldBean.sayHello();
    }
```

Java-Base 定义

```java
@Component("annotationBean")
public class AnnotationBean {
    private String myName;
    public AnnotationBean(){
        System.out.println("AnnotationBean creat
    }
    public void setMyName(String name){
        this.myName = name;
    }
    public void sayHello(){
        System.out.println("hello annotation bea
    }
```

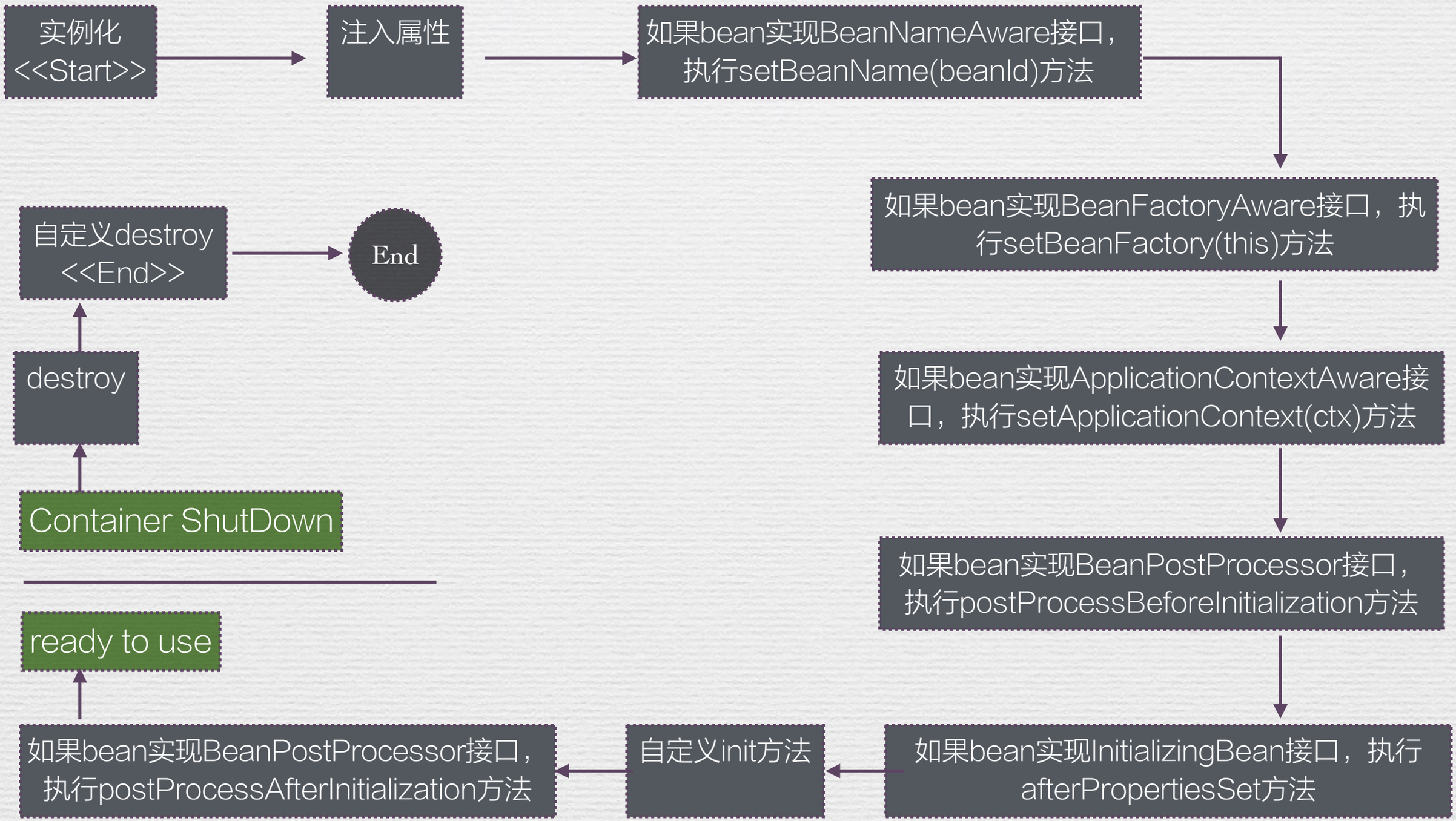
```java
@Configuration
@ComponentScan("xqy.bean")
public class main3 {
    public static void main(String[] args) throws InterruptedException {
        System.out.println(new Date() + " begin");
        ApplicationContext ctx = new AnnotationConfigApplicationContext(main3.class);

        AnnotationBean bean = (AnnotationBean) ctx.getBean( s: "annotationBean");
        bean.setMyName("i am annotation bean");
        System.out.println(new Date()+" get bean "+bean);

        bean.sayHello();
    }
}
```

Spring Bean组件

Bean

- class
- name
- scope
- Constructor argument
- properties
- auto wiring mode
- lazy-initialization mode
- initialization method
- destruction method

Bean生命周期


实例化
<<Start>>
注入属性
如果bean实现BeanNameAware接口，
执行setBeanName(beanId)方法
如果bean实现BeanFactoryAware接口，执
行setBeanFactory(this)方法
如果bean实现ApplicationContextAware接
口，执行setApplicationContext(ctx)方法
如果bean实现BeanPostProcessor接口，
执行postProcessBeforeInitialization方法
如果bean实现InitializingBean接口，执行
afterPropertiesSet方法
自定义init方法
如果bean实现BeanPostProcessor接口，
执行postProcessAfterInitialization方法
ready to use
Container ShutDown
destroy
自定义destroy
<<End>>
End

两个角度装配bean

- Component scanning：从上下文扫描(@ComponentScan)
- Auto wiring: 从bean依赖 (@AutoWired)

Bean作用域

ConfigurableBeanFactory.SCOPE_SINGLETON

全局作用域只有一个

```
@Component
@Scope(ConfigurableBeanFactory.SCOPE_PROTOTYPE)
public class Notepad { ... }
```

调用一次new一个

```
@Component
@Scope(
   value=WebApplicationContext.SCOPE_SESSION,
   proxyMode=ScopedProxyMode.INTERFACES)
public ShoppingCart cart() {
...
}
```

每次会话生产一个

```
@Component
public class StoreService {
  @Autowired
  public void setShoppingCart(ShoppingCart
shoppingCart) {
    this.shoppingCart = shoppingCart;
  }
... }
```

ConfigurableBeanFactory.SCOPE_REQUEST

每次请求new一个

常用注解

@ComponentScan

@Configuration

@Component

@Bean

@Autowired

@Scope

@Named

@Inject

@PropertySource

@Value

@Require

@Profile

@Primary

@Quanlifier

@Conditional

@RunWith

@ContextConfiguration

@Rule

@Test

统一配置信息管理：Profile

Profile剥离程序和配置，可用于环境切换、数据库连接、系统权限等。。。

配置Active Profile：

- DispatcherServlet的初始化参数
- Web 应用 上下文参数
- JNDI 实体
- 环境变量
- JVM系统属性
- @ActiveProfiles注解

Profile Sample

```
public interface DataSource {
    List<User> getAll();
    Map<String, String> getAcount();
}
```

生产数据

```
public class ProductionDataSource implements
DataSource {
    public List<User> getAll() {
        List<User> users = new ArrayList<User>();
        users.add(new User("pro01","123", true, new
Date()));
        users.add(new User("pro02","123",false, new
Date()));
        users.add(new User("pro03","123",false,new
Date()));
        users.add(new User("pro04","123",true,new
Date()));
        return users;
    }
    public Map<String, String> getAcount() {
        Map<String,String> m = new HashMap();
        m.put("username","admin");
        m.put("password","9527");
        return m;
    }
}
```

测试数据

```
public class TestDataSource implements DataSource{
    public List<User> getAll() {
        List<User> users = new ArrayList<User>();
        users.add(new User("test01","456", true, new
Date()));
        users.add(new User("test02","456",false, new
Date()));
        users.add(new User("test03","456",true,new
Date()));
        users.add(new User("test04","456",true,new
Date()));
        return users;
    }
    public Map<String, String> getAcount() {
        Map<String,String> m = new HashMap<>();
        m.put("username","test");
        m.put("password","3125");
        return m;
    }
}
```

Profile Sample

```
@Configuration
public class DataConfig {
    @Bean(name="dataSource")
    @Profile("test")
    public DataSource getTestData(){
        return new TestDataSource();
    }

    @Bean(name="dataSource")
    @Profile("production")
    public DataSource getProductionData(){
        return new ProductionDataSource();
    }
}
```

```
AnnotationConfigApplicationContext ctx =
new AnnotationConfigApplicationContext();

ctx.getEnvironment().setActiveProfiles("test");
                                    激活测试配置信息
ctx.register(DataConfig.class);

ctx.register(ServiceCofig.class);

ctx.refresh();
```

Thank you

–xieqiaoyun

作业如果还未写完的，后面还会继续哒