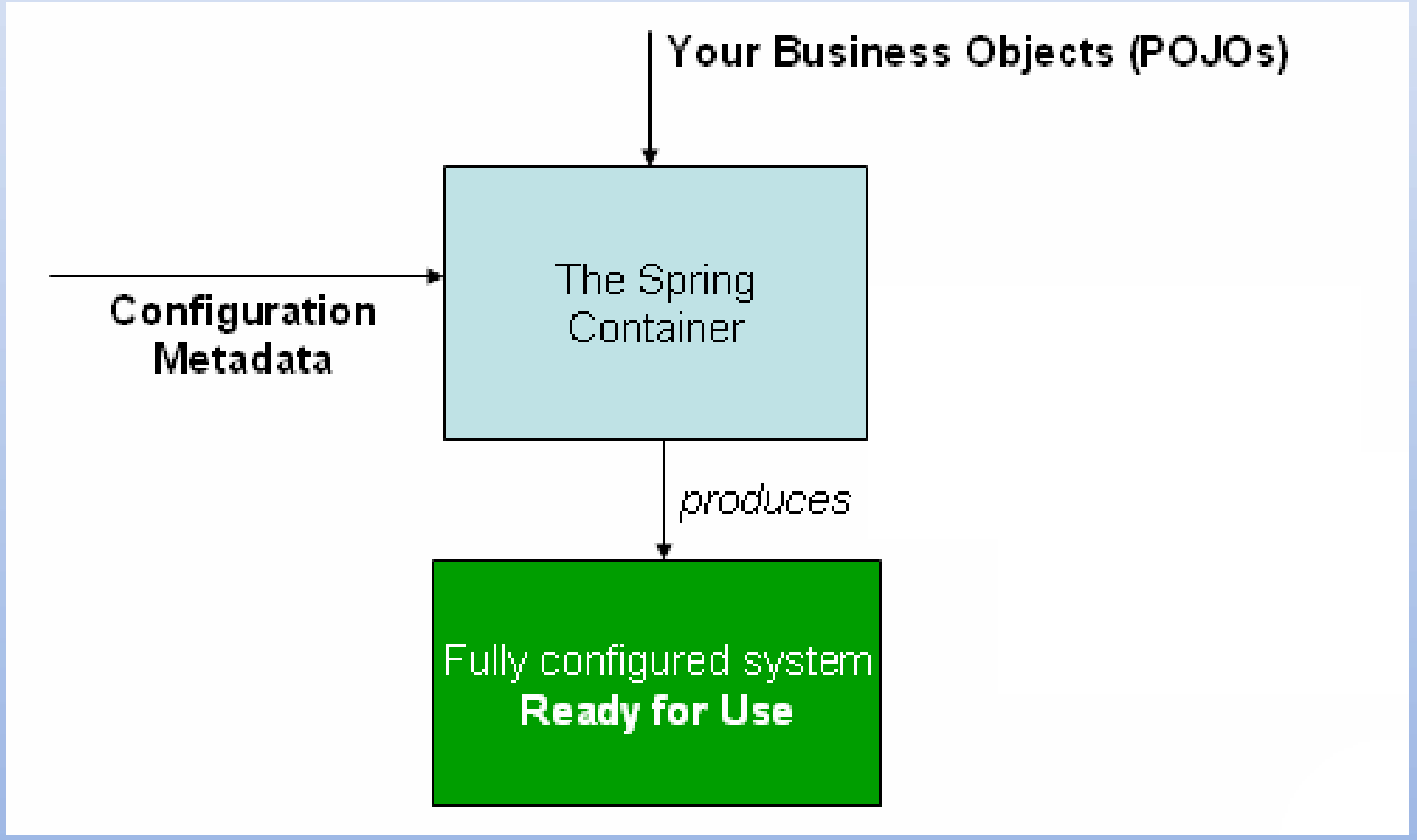
Spring core+beans.

http://docs.spring.io/spring/docs/current/spring-framework-reference/html/index.html

Spring Framework

- **Spring Framework** (или коротко **Spring**)- реализует и продвигает принцип инверсии управления (IOC) или внедрения зависимости (DI) и является по сути IOC-контейнером.

Spring Framework IOC



Spring Configuration

- XML-based configuration;
- Annotation-based configuration;
- Java-based configuration.

XML-based configuration

```
<beans xmlns="http://www.springframework.org/schema/beans" ... >

    <bean id="config" class="org.springframework.beans.factory.config.PropertyPlaceholderConfigurer">
        <property name="locations">
            <list><value>classpath:/config.properties</value></list>
        </property>
    </bean>
    <context:component-scan base-package="nedis.study.jee.components.impl"/>
    <context:component-scan base-package="nedis.study.jee.dao.impl.hibernate"/>
    <context:component-scan base-package="nedis.study.jee.services.impl"/>
    <bean id="dataSource" class="org.postgresql.jdbc2.optional.PoolingDataSource">
        <property name="dataSourceName" value="htppl-ds" />
        <property name="serverName" value="${db.serverName}" />
        <property name="databaseName" value="${db.databaseName}" />
        <property name="user" value="${db.username}" />
        <property name="password" value="${db.password}" />
        <property name="initialConnections" value="10" />
        <property name="maxConnections" value="200" />
    </bean>
    <bean name="sessionFactory" class="org.springframework.orm.hibernate4.LocalSessionFactoryBean">
        <property name="configLocation" value="classpath:/hibernate.cfg.xml"/>
        <property name="dataSource" ref="dataSource"/>
    </bean>
    <bean id="transactionManager" class="org.springframework.orm.hibernate4.HibernateTransactionManager">
        <property name="sessionFactory" ref="sessionFactory"/>
    </bean>
    <tx:annotation-driven transaction-manager="transactionManager"/>
</beans>
```

Annotation-based configuration

```java
@Service("commonService")
@Transactional(readOnly=true)
@Scope(ConfigurableBeanFactory.SCOPE_SINGLETON)
public class CommonServiceImpl implements CommonService {

    @Autowired
    private AccountDao accountDao;

    @Autowired
    @Qualifier("hibernateRoleDao")
    private RoleDao roleDao;

    @Autowired
    private AccountRoleDao accountRoleDao;

    @Value("${nedis.jee.study.supportEmailAddress}")
    private String supportEmailAddress;

    @Autowired
    private EntityBuilder entityBuilder;

    @Autowired
    private EmailService emailService;

    public CommonServiceImpl() {
        super();
    }
```

Java-based configuration

```java
@Configuration
public class ApplicationJeeConfig {

    @Bean(name="config")
    @Scope(ConfigurableBeanFactory.SCOPE_SINGLETON)
    public PropertyPlaceholderConfigurer getConfig(){
        PropertyPlaceholderConfigurer c = new PropertyPlaceholderConfigurer();
        c.setLocation(new ClassPathResource("/config.properties"));
        return c;
    }

    @Bean(name="dataSource")
    public DataSource getDataSource(){
        PoolingDataSource ds = new PoolingDataSource();
        ds.setDatabaseName("xgbua_jeenedis");
        //TODO All config should be here
        return ds;
    }

    @Bean(name="sessionFactory")
    public SessionFactory getSessionFactory(){
        LocalSessionFactoryBean s = new LocalSessionFactoryBean();
        s.setDataSource(getDataSource());
        s.setConfigLocation(new ClassPathResource("/hibernate.cfg.xml"));
        return s.getObject();
    }
}
```

Создание IoC контейнера

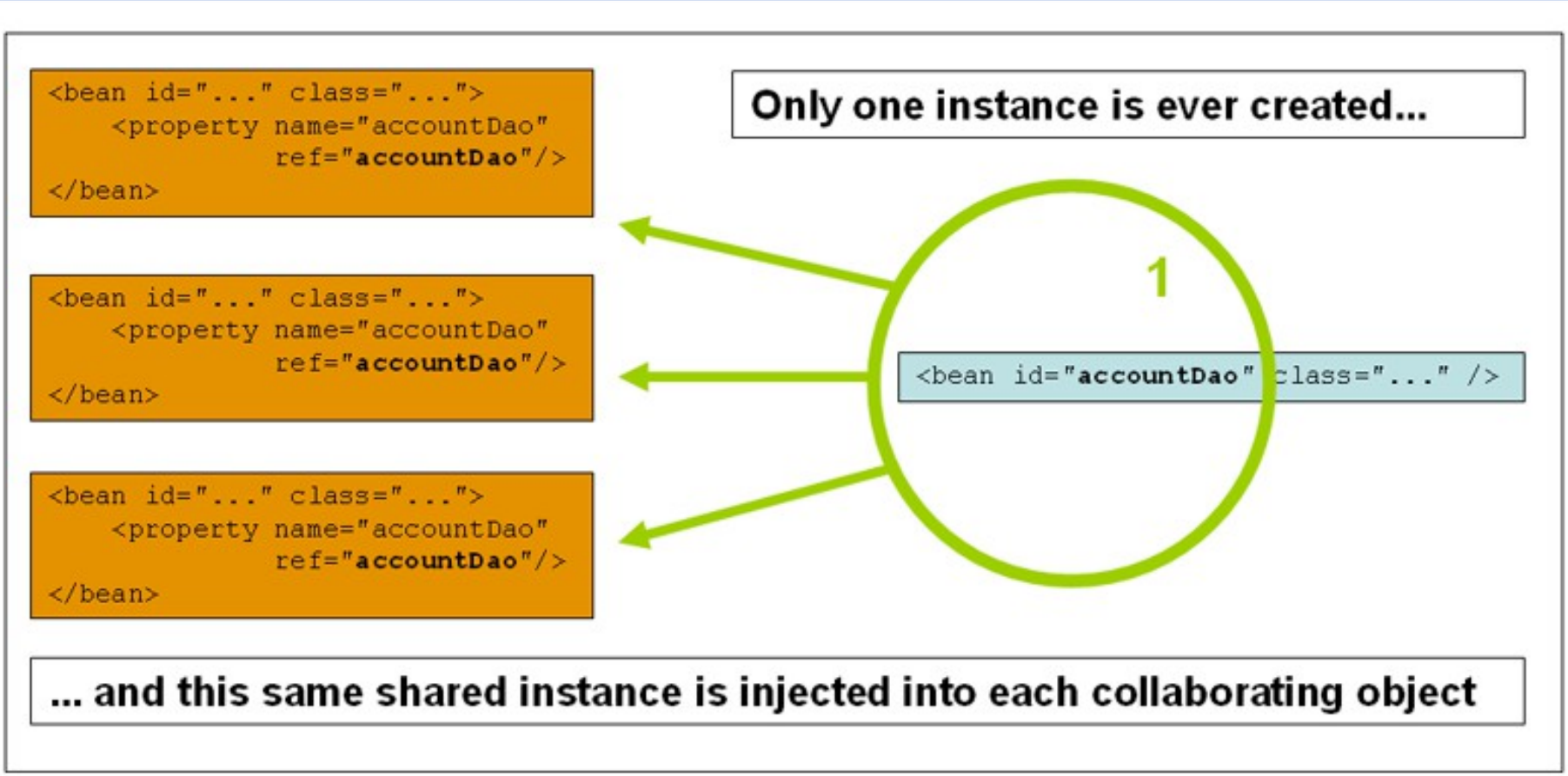
```
ApplicationContext context = new ClassPathXmlApplicationContext(new String[] {"applicationContext.xml"});
CommonService commonService = context.getBean(CommonService.class);
System.out.println("Support email="+commonService.getSupportEmailAddress());
System.out.println("All roles: "+commonService.listAllRoles());
```

```
ApplicationContext context = new AnnotationConfigApplicationContext(ApplicationJeeConfig.class);
CommonService commonService = context.getBean(CommonService.class);
System.out.println("Support email="+commonService.getSupportEmailAddress());
System.out.println("All roles: "+commonService.listAllRoles());
```

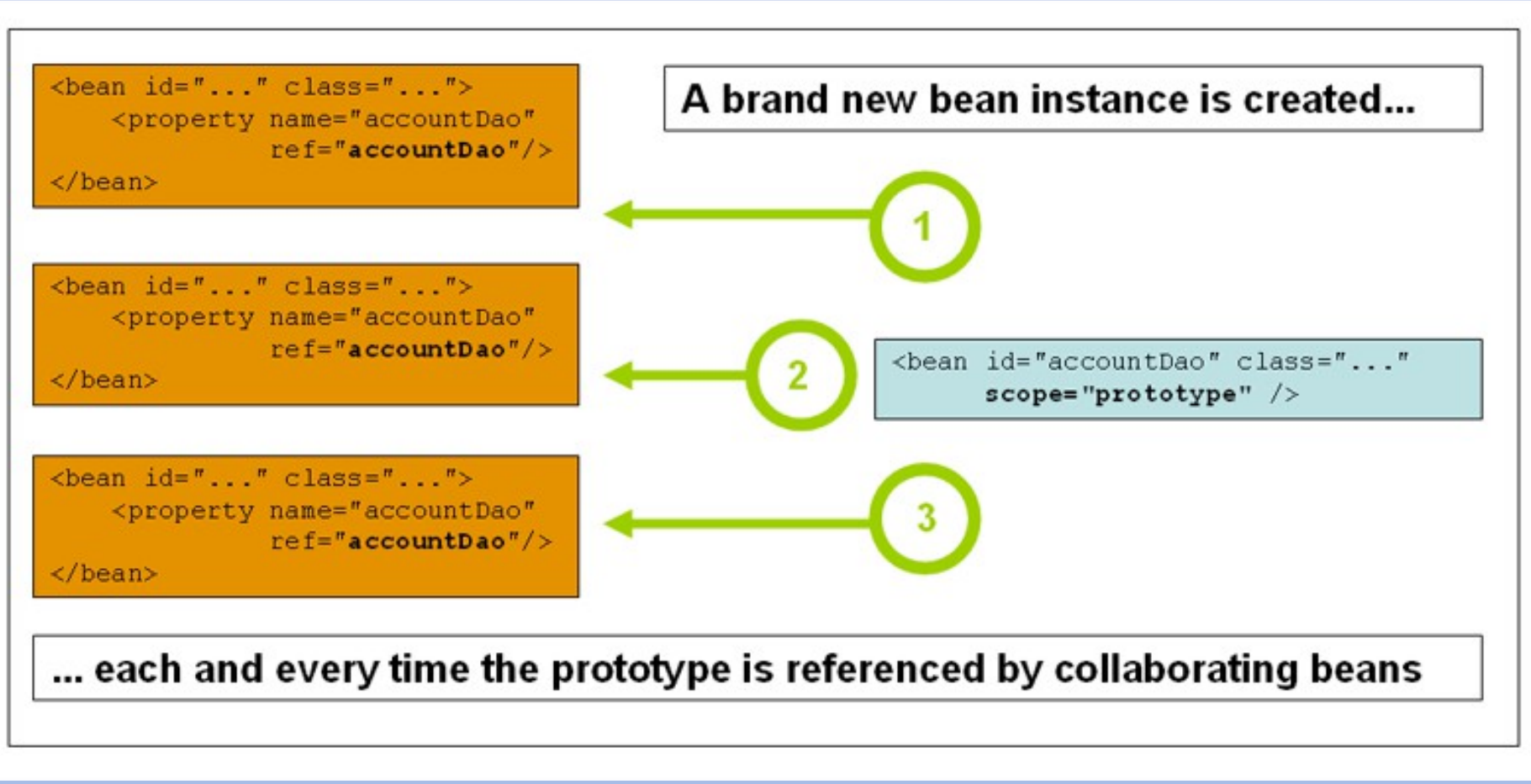
Bean scopes

Scope	Description
singleton	(Default) Scopes a single bean definition to a single object instance per Spring IoC container.
prototype	Scopes a single bean definition to any number of object instances.
request	Scopes a single bean definition to the lifecycle of a single HTTP request; that is, each HTTP request has its own instance of a bean created off the back of a single bean definition. Only valid in the context of a web-aware Spring ApplicationContext.
session	Scopes a single bean definition to the lifecycle of an HTTP Session. Only valid in the context of a web-aware Spring ApplicationContext.
global session	Scopes a single bean definition to the lifecycle of a global HTTP Session. Typically only valid when used in a portlet context. Only valid in the context of a web-aware Spring ApplicationContext.
application	Scopes a single bean definition to the lifecycle of a ServletContext. Only valid in the context of a web-aware Spring ApplicationContext.

The singleton scope



The prototype scope



Request, Session, Global session

DispatcherServlet,
RequestContextListener
RequestContextFilter

docs.spring.io/spring/docs/current/spring-framework-reference/html/beans.html#beans-factory-scopes-other

```
<bean id="userPreferences" class="com.foo.UserPreferences" scope="session">
  <aop:scoped-proxy/>
</bean>

<bean id="userManager" class="com.foo.UserManager">
  <property name="userPreferences" ref="userPreferences"/>
</bean>
```

Custom scope

docs.spring.io/spring/docs/current/spring-framework-reference/html/beans.html#beans-factory-scopes-custom-using

```
package org.springframework.beans.factory.config;

public interface Scope {

    Object get(String name, ObjectFactory<?> objectFactory);

    Object remove(String name);

    void registerDestructionCallback(String name, Runnable callback);

    Object resolveContextualObject(String key);

    String getConversationId();
}
```

Lifecycle callbacks

Init:

Methods annotated with **@PostConstruct**

afterPropertiesSet() as defined by the InitializingBean callback interface

A custom configured **init()** method

Destroy:

Methods annotated with **@PreDestroy**

destroy() as defined by the DisposableBean callback interface

A custom configured **destroy()** method

The destroy-method attribute of a <bean> element can be assigned a special (inferred) value which instructs Spring to automatically detect a public close or shutdown method on the specific bean class (any class that implements java.lang.AutoCloseable or java.io.Closeable would therefore match). This special (inferred) value can also be set on the default-destroy-method attribute of a <beans> element to apply this behavior to an entire set of beans. Note that this is the default behavior with Java config.

Aware interfaces

Name	Injected Dependency
ApplicationContextAware	Declaring ApplicationContext
ApplicationEventPublisherAware	Event publisher of the enclosing ApplicationContext
BeanClassLoaderAware	Class loader used to load the bean classes.
BeanFactoryAware	Declaring BeanFactory
BeanNameAware	Name of the declaring bean
BootstrapContextAware	Resource adapter BootstrapContext the container runs in. Typically available only in JCA aware ApplicationContexts
LoadTimeWeaverAware	Defined *weaver* for processing class definition at load time
MessageSourceAware	Configured strategy for resolving messages (with support for parametrization and internationalization)
NotificationPublisherAware	Spring JMX notification publisher
PortletConfigAware	Current PortletConfig the container runs in. Valid only in a web-aware Spring ApplicationContext
PortletContextAware	Current PortletContext the container runs in. Valid only in a web-aware Spring ApplicationContext
ResourceLoaderAware	Configured loader for low-level access to resources
ServletConfigAware	Current ServletConfig the container runs in. Valid only in a web-aware Spring ApplicationContext
ServletContextAware	Current ServletContext the container runs in. Valid only in a web-aware Spring ApplicationContext

FactoryBean

http://docs.spring.io/spring/docs/current/spring-framework-reference/html/beans.html#beans-factory-extension-factorybean

```
package org.springframework.beans.factory.config;

public interface FactoryBean<T> {

    T getObject() throws Exception;

    Class<?> getObjectType();

    boolean isSingleton();
}
```

Spring Annotations

@Required
@Autowired
@PersistenceContext
@Qualifier("main")
@Qualifier public @*interface* Genre { String value(); }
JSR-250 **@Resource**
@PostConstruct and @PreDestroy
@Component
@Repository
@Service
@Controller
@Scope
@ javax.inject.Inject
@ javax.inject.Named
<context:component-scan base-package="org.example"/>

Spring annotations vs. standard annotations

Spring	javax.inject.*	javax.inject restrictions / comments
@Autowired	@Inject	@Inject has no 'required' attribute; can be used with Java 8's Optional instead.
@Component	@Named	JSR-330 does not provide a composable model, just a way to identify named components.
@Scope("singleton")	@Singleton	The JSR-330 default scope is like Spring's prototype. However, in order to keep it consistent with Spring's general defaults, a JSR-330 bean declared in the Spring container is a singleton by default. In order to use a scope other than singleton, you should use Spring's @Scope annotation. javax.inject also provides a @Scope annotation. Nevertheless, this one is only intended to be used for creating your own annotations.
@Qualifier	@Qualifier / @Named	javax.inject.Qualifier is just a meta-annotation for building custom qualifiers. Concrete String qualifiers (like Spring's @Qualifier with a value) can be associated through javax.inject.Named.
@Value	-	no equivalent
@Required	-	no equivalent
@Lazy	-	no equivalent
ObjectFactory	Provider	javax.inject.Provider is a direct alternative to Spring's ObjectFactory, just with a shorter get() method name. It can also be used in combination with Spring's @Autowired or with non-annotated constructors and setter methods.

BeanFactory and ApplicationContext

Feature	BeanFactory	ApplicationContext
Bean instantiation/wiring	Yes	Yes
Automatic BeanPostProcessor registration	No	Yes
Automatic BeanFactoryPostProcessor registration	No	Yes
Convenient MessageSource access (for i18n)	No	Yes
ApplicationEvent publication	No	Yes

Resource

Prefix	Example	Explanation
classpath:	classpath:com/myapp/config.xml	Loaded from the classpath.
file:	file:/data/config.xml	Loaded as a URL, from the filesystem. [1]
http:	http://myserver/logo.png	Loaded as a URL.
(none)	/data/config.xml	Depends on the underlying ApplicationContext.