Practical Spring

— Annotated IoC Container & Template Mechanism

William Mo

morningspace@gmail.com

Agenda

- Spring IoC Container
- Template Mechanism

Inversion of Control

- IoC is a common characteristic of framework
  - Framework, not library
  - It gives frameworks the power to serve as extensible skeletons
- Hollywood's Law: Don't call us, we'll call you
- Dependency Inversion Principle
- How do we use IoC in practice:
  - closure, callback, observer, template method

What Aspect of Control does IoC Container Invert?

- A specific inversion of control, on how we lookup a component implementation.
- Using a specific assemblor to assemble dependencies instead of instantiating directly by component itself
  - Dependency Assembling or Dependency Management
  - Spring, Pico Container, etc.

Dependency Injection

- As IoC is too generic, we use Dependency Injection
- There're several styles of DI:
  - Constructor Injection(T3)
  - Setter Injection(T2)
  - Interface Injection(T1)
  - Getter Injection
- Dependency Injection is a useful alternative to Service Locator.

A Brief Look at Spring IoC Container

- Use Application Context or Bean Factory to manage and configure beans(mostly, POJOs)
- Read bean definitions from XML or properties files
- Removing the need for custom singletons and ad hoc configuration.
- Application Context also provide some additional features such as i18n, event publication, etc.

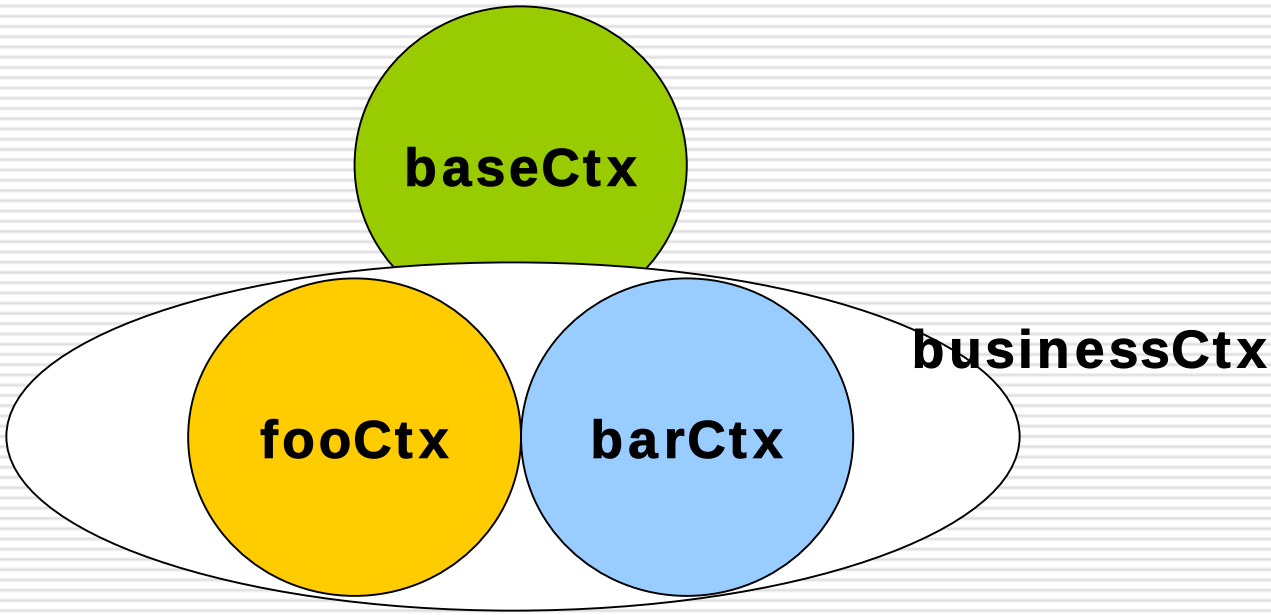
A Closer Look at Bean Definition

- Some interesting features:
  - hierarchical bean definition
  - Nestable bean definition
  - primitive property, collection property
  - property placeholder configurer
  - bean definition import
  - …

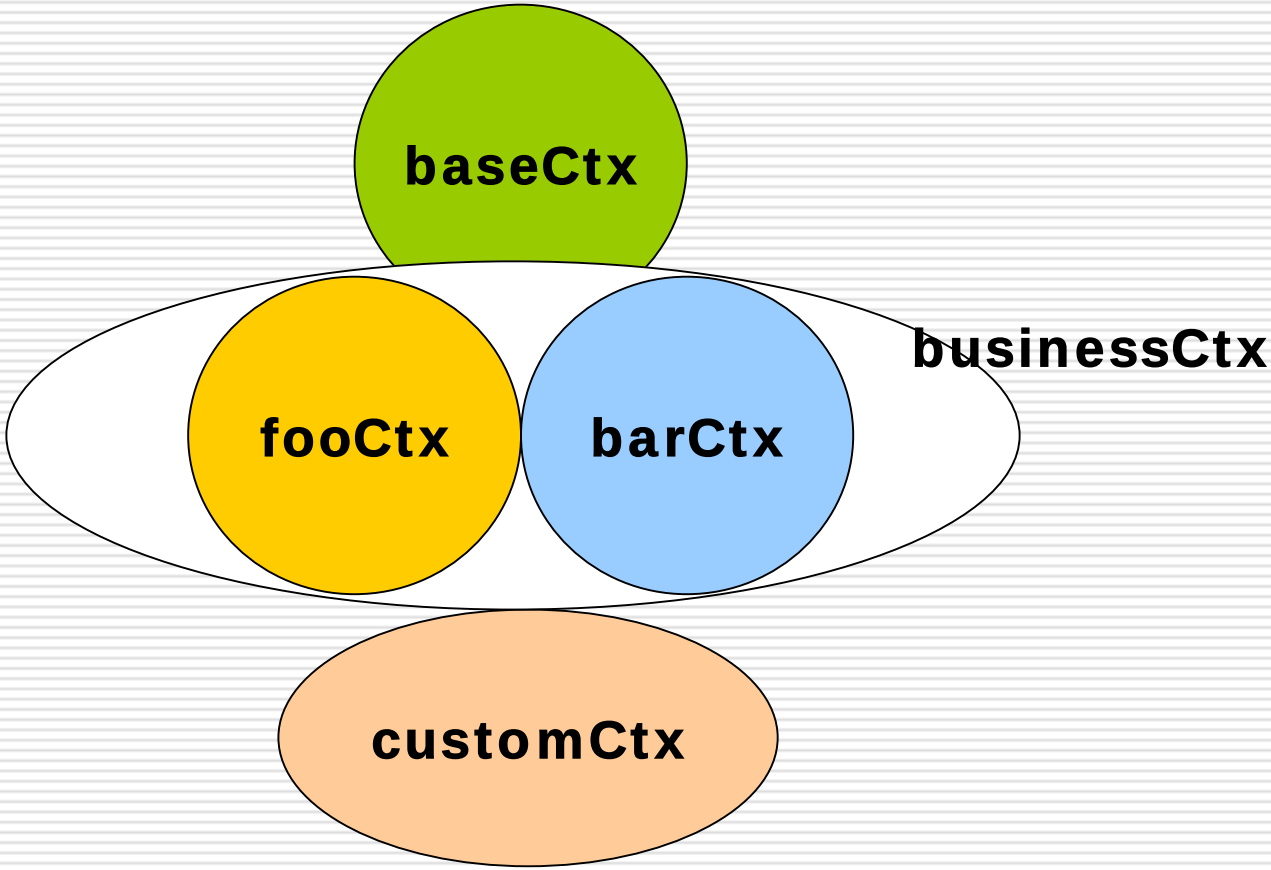
Two User Stories

- Story A: Singleton-style, logical, hierarchical application context
  - We have several bean definition files separated in different sub-projects
  - Each definition file will be put into different distributed achieve
  - They build up as a whole context logically, and also hierarchically
  - We need a singleton context which can be accessed within any part of application
  - Other people can reuse and derive our context, while no need to modify existed bean definitions

Singleton-style Logical Hierarchical Application Context



Singleton-style Logical Hierarchical Application Context



Singleton-style Logical Hierarchical Application Context

- Use ContextSingletonBeanFactoryLocator. It can build up a whole context from one or more bean definition fragments. e.g.

```
BeanFactoryLocator locator =
    ContextSingletonBeanFactoryLocator.getInstance(
        "classpath:beanRefContext.xml");
BeanFactory bf =
     locator.useBeanFactory("businessCtx").getFactory();
```

Test List

- The Context returned by Context…Locator should include all beans if we specify child context
- The Context returned by Context…Locator should include base bean only when we specify parent context
- The Context returned by Context…Locator should include all beans plus custom beans if we specify custom context
- …

Introduce a singleton helper

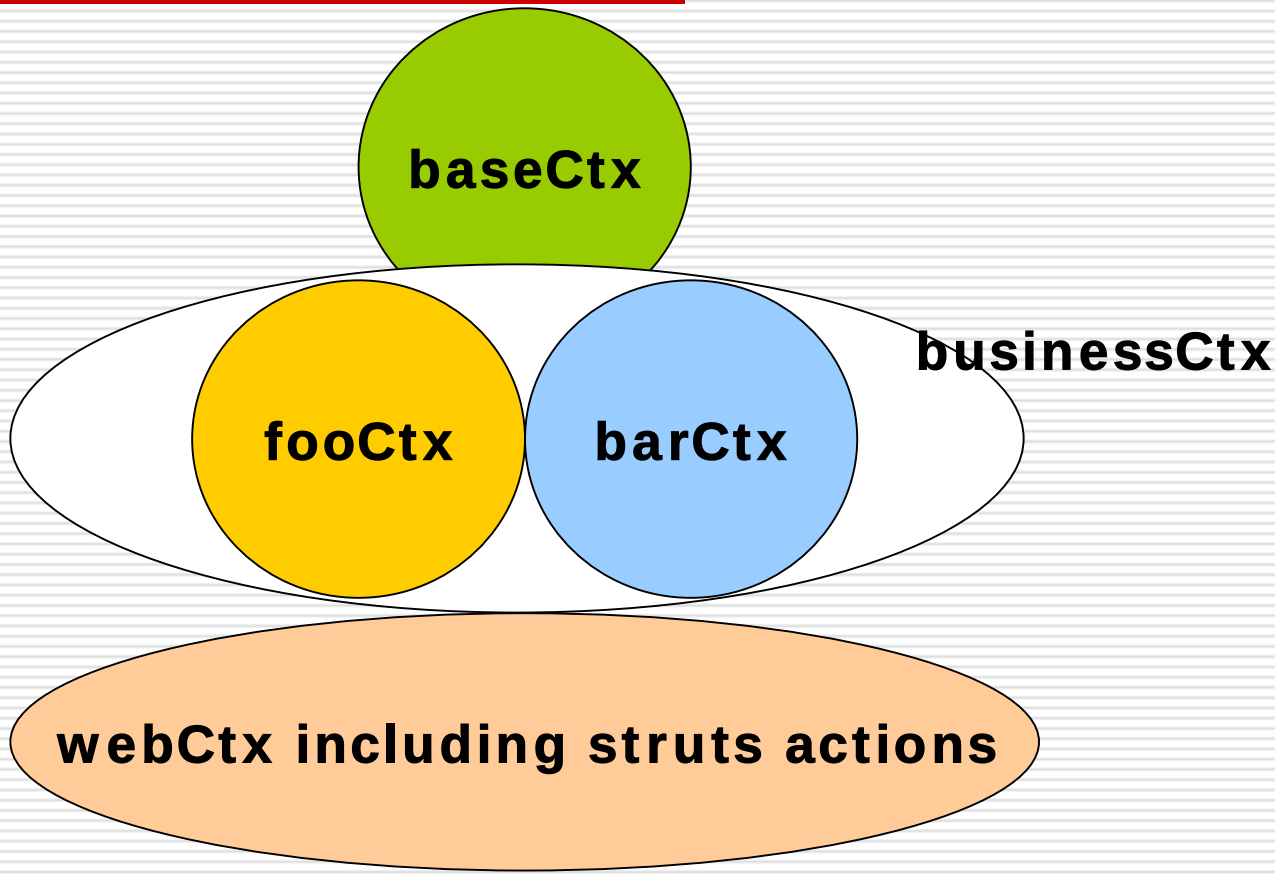
- Use a helper class to wrap the bean factory locating logic

```
ContextUtil.getContext(
    "classpath:beanRefContext.xml“, ”businessCtx”);
ContextUtil.getContext();
```

A Second User Story

- Story B: Integrate middle tie context with web tie context
  - We have a struts based web application
  - We want to use IoC features in struts actions
  - We also want to use a single whole context which across web tie and middle tie
  - Meanwhile, we shall avoid the “paradoxical singleton” problem

Integrate Middle Tie Context With Web Tie Context



Integrate Middle Tie Context With Web Tie Context

- We can use ContextLoaderListener or ContextLoaderServlet to hook application context into the servlet context.
- We can use ContextLoaderPlugin and DelegatingActionProxy to define Struts actions as beans, wiring them with middle tier components defined in the root context.

Integrate Middle Tie Context With Web Tie Context

- We could customize ContextLoader to integrate the middle tier context and avoid the “paradoxical singleton” problem
- There’re may customized ways
  - We could retrieve created web application context from servlet context and put it into a singleton helper

Test List

- The web context should include beans defined within web context only, if we ignore the locator parameters in web.xml
- The web context should include all beans if we provide the locator parameters
- The web context should include struts actions filled with related beans if we provide the locator parameters
- According to singleton bean, the web context and ContextUtil should return the same bean

A Brief Summary

- What Spring gives us:
  - Convenient, out-of-the-box building blocks
  - Flexible, easy to customized
  - Consistent, using a consistent way to reduce learning curve
  - Testable
- With Spring IoC container, we separate the bean configuration from use.
- We can assemble components from different layers transparently

Something More to Mention

- When we use singleton factory locator. Does it mean another Service Locator?
  - Spring team suggest the use of singleton factory locator is unnecessary except for a small amount of Glue Code.
  - A better way is using Base Class to hide the coupling logic.
    - Spring JPetStore
    - ActionSupport.

The Fundamental Design Principle

- Where we put the dependency assembling code.
- The choice between Service Locator and Dependency Injection is less important than the principle of separating service configuration from the use of services in an application (Factory, Singleton)
- Singleton is not always hard to use. Due to testability, we can customize ContextUtil if necessary. e.g.

```
ContextUtil.load(new TestContextUtil());
ContextUtil.getContext();
```

Agenda

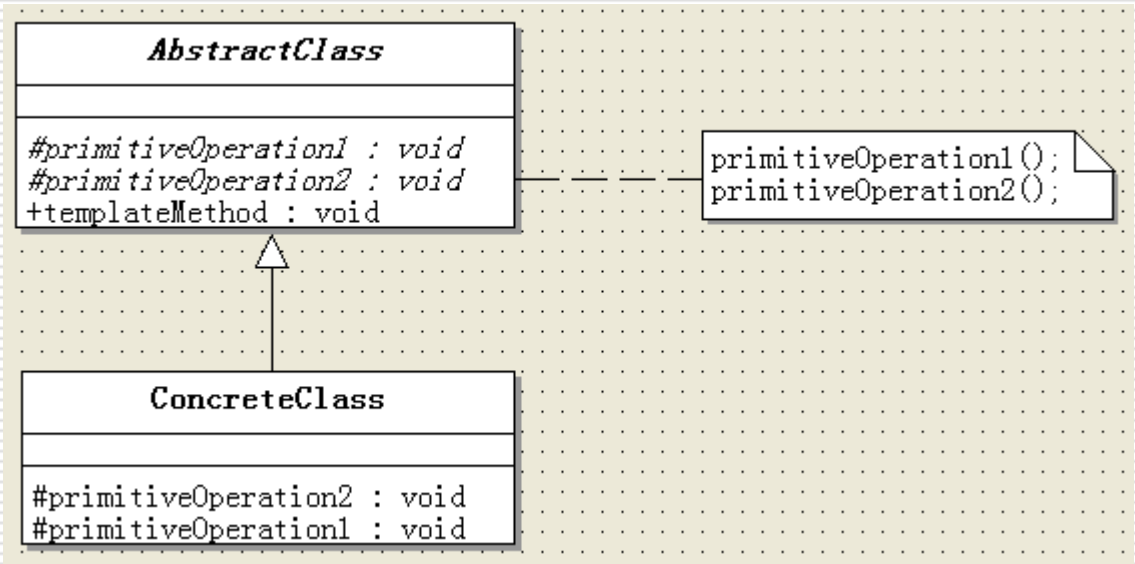
- **Spring IoC Container**
- ***Template mechanism***

Template Method

- Definition: The super-class defines the flow of control, while subclasses extend overriding methods or implementing abstract methods to do the extension.
  - The framework does the calling, your code reacts. (Also Inversion of Control)

Template Method

- Class Diagram



Specific Domain — Resource Access

- Resource: JDBC, LDAP, JNDI, JMS etc.
- Difficulty to obtain resource.
  - There is no consistent way to access
  - So much exceptions to handle and vendor specific
  - Forget to release resource may lead to serious resource leakage, especially for newcomers
  - …

The Solution Given by Spring

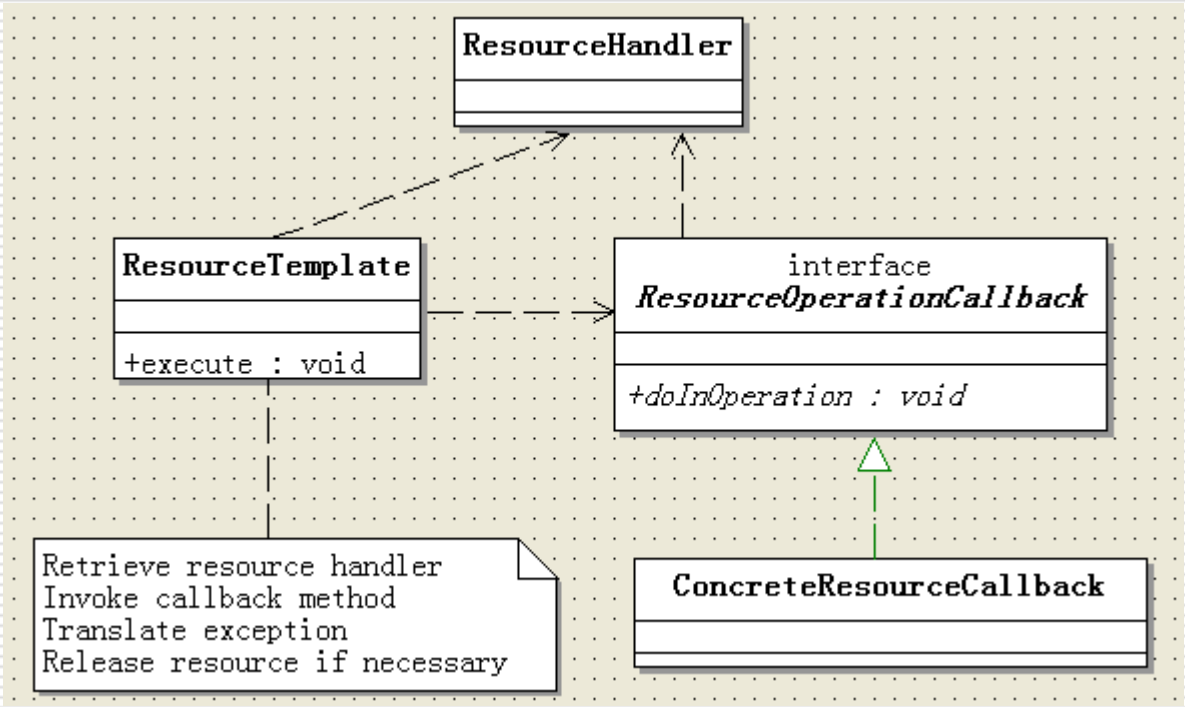
- Support/Template
  - JCA
  - JDBC
  - JMS
  - JNDI
  - ORM
    - Hibernate
    - iBatis
    - JDO
    - OJB
    - …
  - Transaction

Take JdbcTemplate for Example

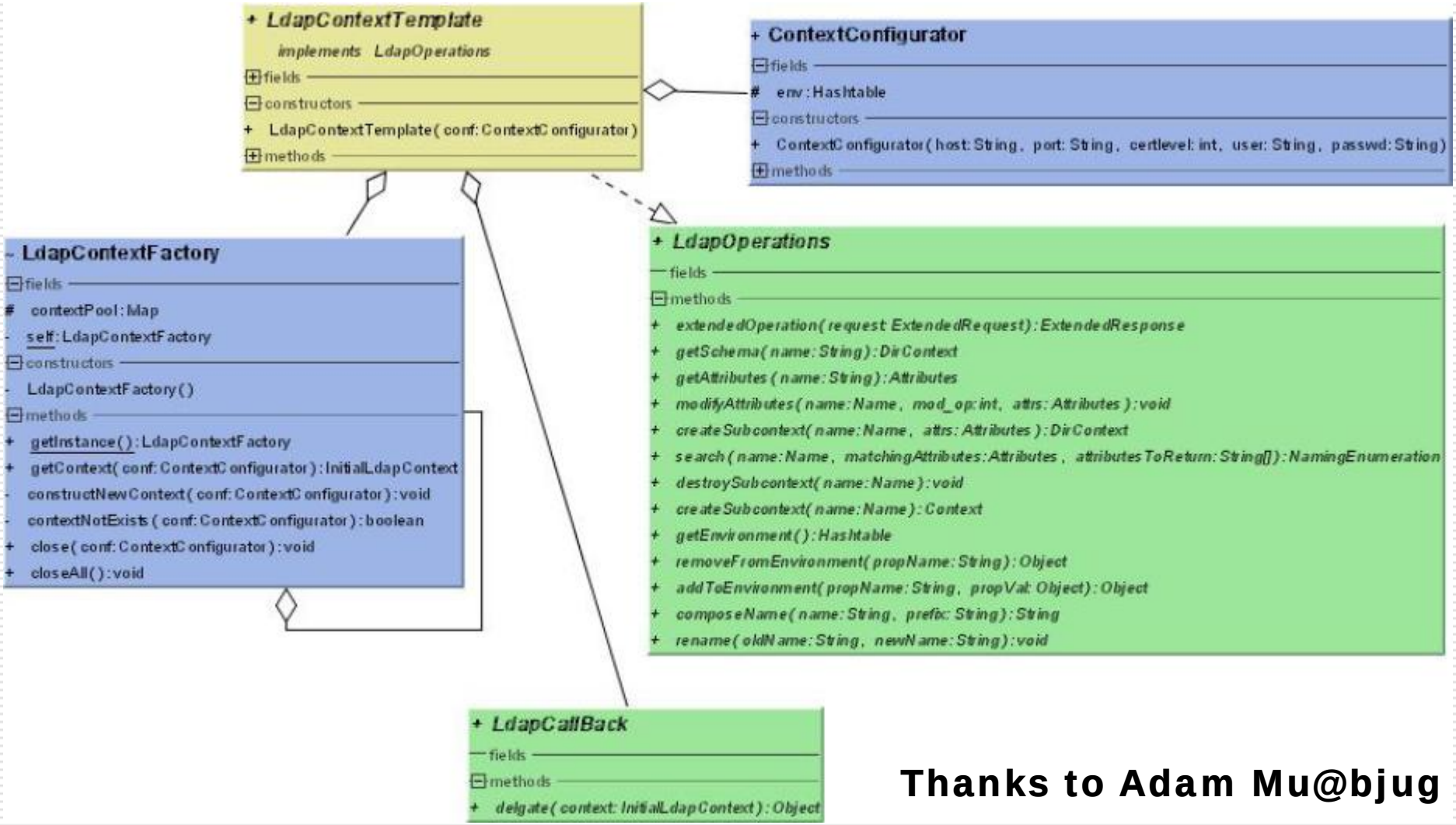
- We
  - Wrap resource operation within anonymous inner class and hand it to the framework
  - The inner class implement a callback interface given by framework
- The framework
  - Obtain resource handler from current thread or create new one
  - Invoke callback method
  - Do exception translating
  - Release resource handler if necessary

Take JdbcTemplate for Example

- Class Diagram



Case Study



Thanks to Adam Mu@bjug

A Brief Summary

- Separate the resource accessing logic from resource operation code
- No try/catch block any more
- No need to worry about resource leakage
- Reduce many resource access methods to one-lines
- Provide a common and consistent way to access resource

Reference

- Adam Mu, *Experiencing the Spring* (http://www.bjug.org/party/2005.07.16/2005.07.16.spring-dao.pdf)
- Martin Fowler, *Inversion of Control Containers and the Dependency Injection pattern* (http://www.martinfowler.com/articles/injection.html)
- Martin Fowler, *Inversion of Control* (http://martinfowler.com/bliki/InversionOfControl.html)
- Bob Lee, *Getter-based Dependency Injection* (http://weblogs.java.net/pub/wlg/1333)
- Matt Raible, *Spring Live*
- Rod Johnson, Juergen Hoeller, *J2EE development without EJB*
- Robert C. Martin, *Agile Software Development*

Thanks

北京**Java**用户组

http://www.bjug.org

满江红开放技术研究组织

http://www.redsaga.com