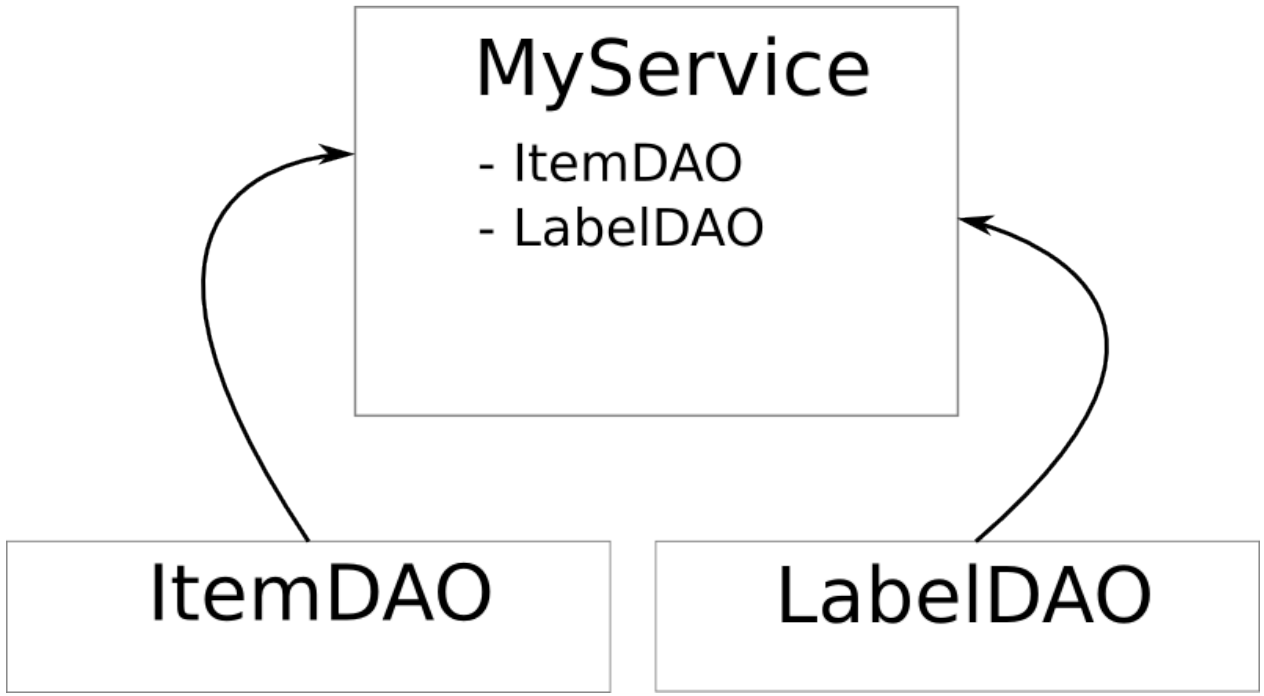
Overview of Dependency Injection in Spring

Richard Paul
Kiwiplan NZ Ltd
27 Mar 2009

What is Dependency Injection

Dependency Injection is a form of Inversion of Control.
Also known as the Hollywood principle,
"Don't call us, we'll call you!"

MyService is given instances of ItemDAO and LabelDAO. MyService isn't responsible for looking up DAOs.



Benefits of Dependency Injection

Ensures configuration and usages of services are separate.

Can switch implementations by simply changing the configuration.

Enhances testability as mock dependencies can be injected

Dependencies are easily identified

- No need to read the source code to see what dependencies your code talks to.

Constructor vs Setter Injection

```
// Constructor
public class MyService implement Service {
  private ItemDAO itemDAO;
  private LabelDAO labelDAO;

  public MyService(ItemDAO itemDAO, LabelDAO labelDAO) {
    this.itemDAO = itemDAO;
    this.labelDAO = labelDAO;
  }
}
// Setter
public class MyService implement Service {
  private ItemDAO itemDAO;
  private LabelDAO labelDAO;

  public setItemDAO(ItemDAO itemDAO) {
    this.itemDAO = itemDAO;
  }
  public setLabelDAO(LabelDAO labelDAO) {
    this.labelDAO = labelDAO;
  }
}
```

Constructor vs Setter Injection

Both constructor and setter injection provide:

- Loose coupling
- Separation of configuration and usage
- Enhanced testability

Constructor injection has a slight advantage (IMO) as it is upfront about what collaborators it requires.

More details at:
http://misko.hevery.com/2009/02/19/constructor-injection-vs-setter-injection/

Dependency Injection in Spring

There are multiple ways of setting up the dependency configuration in spring.

- XML
- Annotations
- JavaConfig
- ...

It is possible to mix and match definition styles.

XML

Give *myService* access to *labelDAO*.

```
<bean id="labelDAO" class="com.example.dao.MyLabelDAO"/>

<!-- Constructor Injection -->
<bean id="myService1" class="com.example.service.MyServiceImpl">
  <constructor-arg ref="labelDAO"/>
</bean>

<!-- Setter Injection -->
<bean id="myService2" class="com.example.service.MyServiceImpl">
  <property name="labelDAO" ref="labelDAO"/>
</bean>
```

XML - Multiple Files

The XML configuration can be split amoung multiple files. With each file containing configuration for different areas.

- applicationContext.xml
- controllers.xml
- dao.xml
- ...

Annotations - Example

You can by-pass some of the XML configuration by using Spring's annotation support.

```
@Service
public class MyServiceImpl implement MyService {
  private ItemDAO itemDAO;
  private LabelDAO labelDAO;

  @Autowired
  public MyService(ItemDAO itemDAO, LabelDAO labelDAO) {
    this.itemDAO = itemDAO;
    this.labelDAO = labelDAO;
  }
}

@Repository
public class MyItemDAO implements ItemDAO {
  // ...
}
```

Annotations - Listing

There are many available annotation for wiring:

@Autowired
@Required
@Qualifier("xxx")
@PostConstruct/@PreDestroy
@Resource - JSR-250
@Component
@Service
@Respository
@Controller

http://static.springframework.org/spring/docs/2.5.x/reference/beans.html#beans-annotation-config

Java Config

Dependency Injection with configuration done using Java. Uses annotations and replaces the XML config with Java code.

```
@Configuration
public class AppConfig {
  @Bean
  public Foo foo() {
    return new Foo(bar());
  }
  @Bean
  public Bar bar() {
    return new Bar();
  }
}
```

Alternatives

The main alternative to Spring's Dependency Injection is Guice.

More info available at:
http://www.infoq.com/news/2007/03/guice-javaconfig

Questions?