



Spring基础 - Spring简单例子引入Spring要点

上文中我们简单介绍了Spring和Spring Framework的组件，那么这些Spring Framework组件是如何配合工作的呢？本文主要承接上文，向你展示Spring Framework组件的典型应用场景和基于这个场景设计出的简单案例，并以此引出Spring的核心要点，比如IOC和AOP等；在此基础上还引入了不同的配置方式，如XML，Java配置和注解方式的差异。@pdai

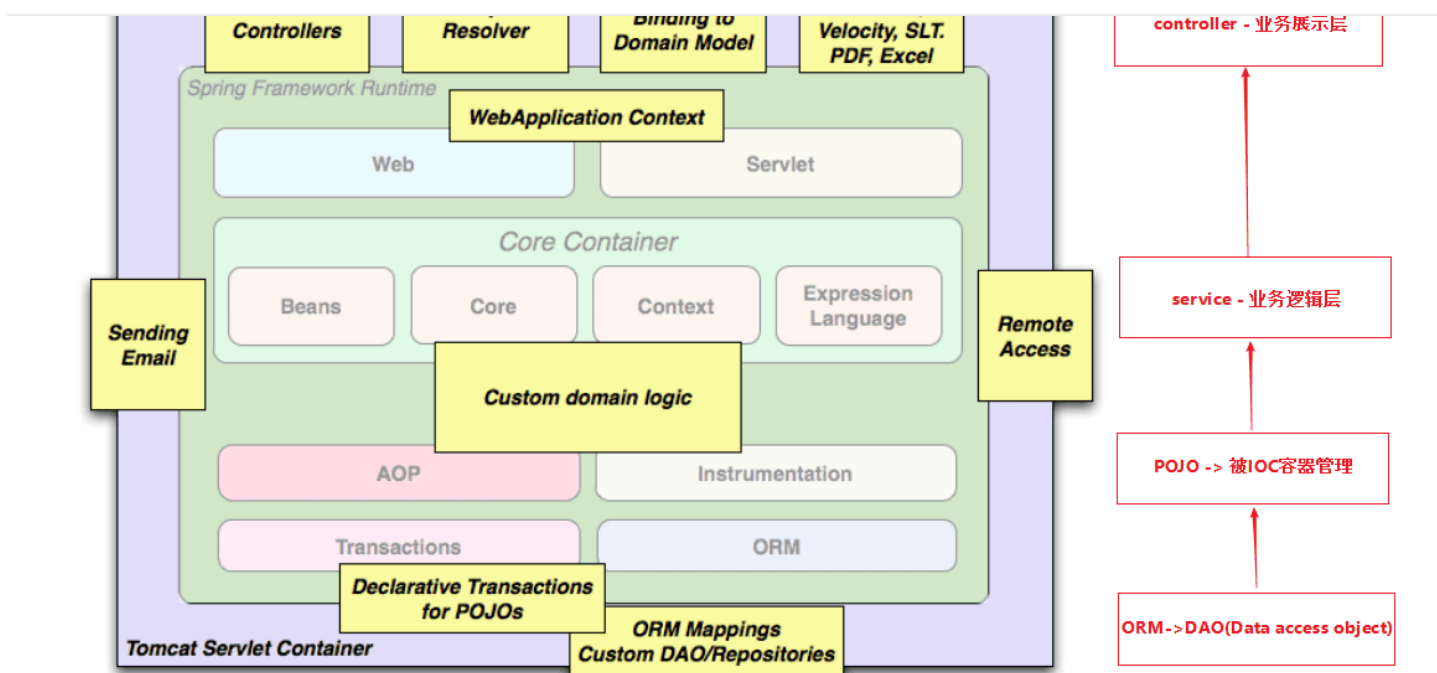
- Spring基础 - Spring简单例子引入Spring要点
 - Spring框架如何应用
 - 设计一个Spring的Hello World
 - 这个例子体现了Spring的哪些核心要点
 - 控制反转 - IOC
 - 面向切面 - AOP
 - Spring框架设计如何逐步简化开发的
 - Java 配置方式改造
 - 注解配置方式改造
 - SpringBoot托管配置
 - 结合Spring历史版本和SpringBoot看发展

Spring框架如何应用

上文中，我们展示了Spring和Spring Framework的组件，这里对于开发者来说有几个问题：

1. 首先，对于Spring进阶，直接去看IOC和AOP，存在一个断层，所以需要整体上构建对Spring框架认知上进一步深入，这样才能构建知识体系。
2. 其次，很多开发者入门都是从Spring Boot开始的，他对Spring整体框架底层，以及发展历史不是很了解；特别是对于一些老旧项目维护和底层bug分析没有全局观。
3. 再者，Spring代表的是一种框架设计理念，需要全局上理解Spring Framework组件是如何配合工作的，需要理解它设计的初衷和未来趋势。

如下是官方在解释Spring框架的常用场景的图



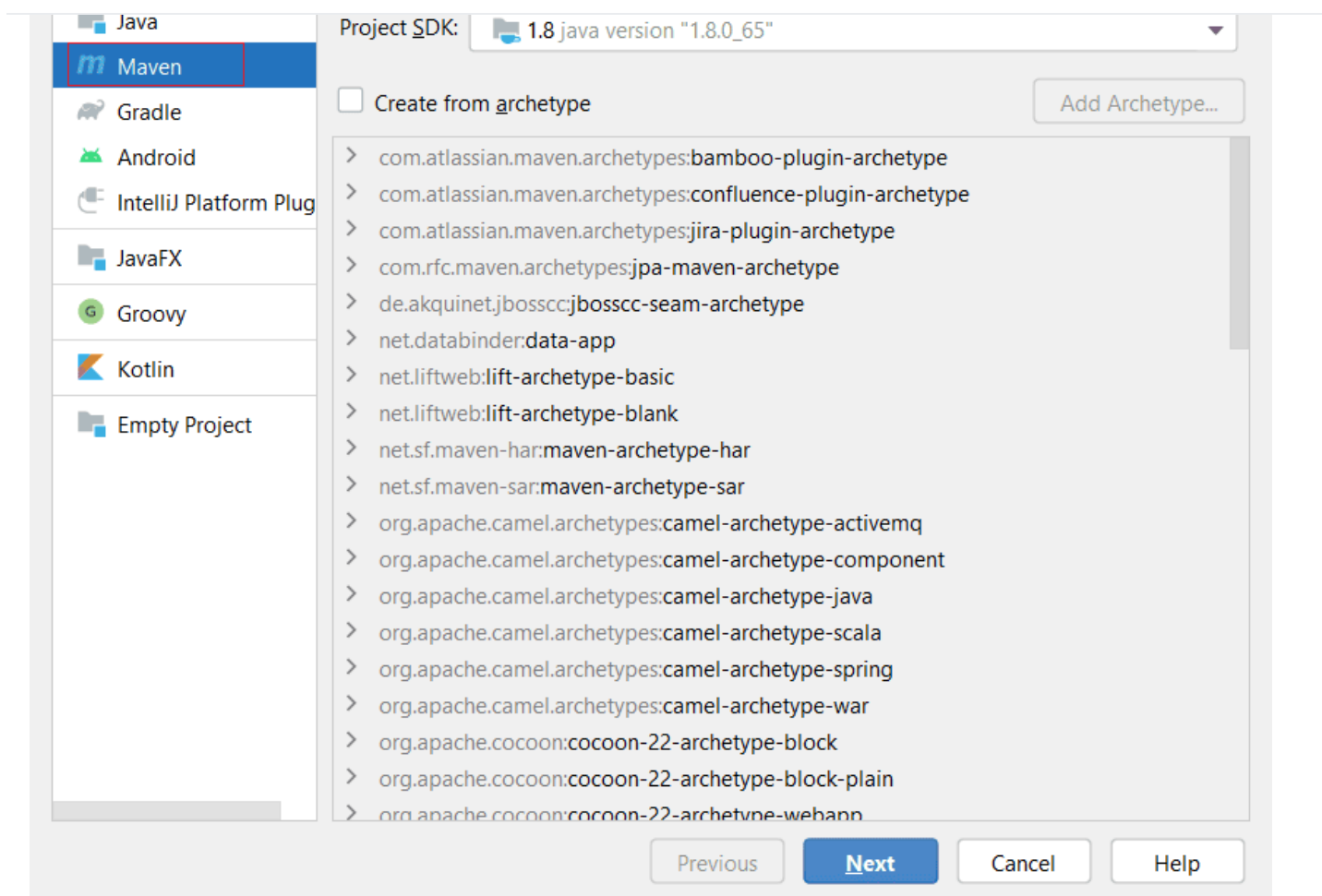
我加上一些注释后，是比较好理解的；引入这个图，重要的原因是为后面设计一个案例帮助你构建认知。

设计一个Spring的Hello World

结合上面的使用场景，设计一个查询用户的案例的两个需求，来看Spring框架帮我们简化了什么开发工作：

1. 查询用户数据 - 来看DAO+POJO-> Service 的初始化和装载。
2. 给所有Service的查询方法记录日志

- 创建一个Maven的Java项目



• 引入Spring框架的POM依赖，以及查看这些依赖之间的关系

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <project xmlns="http://maven.apache.org/POM/4.0.0"
3     xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4     xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">
5     <modelVersion>4.0.0</modelVersion>
6
7     <groupId>tech.pdai</groupId>
8     <artifactId>001-spring-framework-demo-helloworld-xml</artifactId>
9     <version>1.0-SNAPSHOT</version>
10
11     <properties>
12         <maven.compiler.source>8</maven.compiler.source>
13         <maven.compiler.target>8</maven.compiler.target>
14         <spring.version>5.3.9</spring.version>
15         <aspectjweaver.version>1.9.6</aspectjweaver.version>
16     </properties>
17
18     <dependencies>
19         <dependency>
20             <groupId>org.springframework</groupId>
21             <artifactId>spring-context</artifactId>
22             <version>${spring.version}</version>
23         </dependency>
24         <dependency>
25             <groupId>org.springframework</groupId>
26             <artifactId>spring-core</artifactId>
27             <version>${spring.version}</version>
28         </dependency>
29     </dependencies>
```

xml

```

32         <version>${spring.version}</version>
33     </dependency>
34     <dependency>
35         <groupId>org.aspectj</groupId>
36         <artifactId>aspectjweaver</artifactId>
37         <version>${aspectjweaver.version}</version>
38     </dependency>
39 </dependencies>
40
41 </project>

```

```

aspectjweaver : 1.9.6 [compile]
▼ spring-beans : 5.3.9 [compile]
    spring-core : 5.3.9 [compile]
▼ spring-context : 5.3.9 [compile]
    ▼ spring-aop : 5.3.9 [compile]
        spring-beans : 5.3.9 [compile]
        spring-core : 5.3.9 [compile]
        spring-beans : 5.3.9 [compile]
        spring-core : 5.3.9 [compile]
    ▼ spring-expression : 5.3.9 [compile]
        spring-core : 5.3.9 [compile]
▼ spring-core : 5.3.9 [compile]
    spring-jcl : 5.3.9 [compile]

```

• POJO - User

java

```

1  package tech.pdai.springframework.entity;
2
3  /**
4   * @author pdai
5   */
6  public class User {
7
8      /**
9       * user's name.
10      */
11     private String name;
12
13     /**
14      * user's age.
15      */
16     private int age;
17
18     /**
19      * init.
20      *
21      * @param name name
22      * @param age age
23      */
24     public User(String name, int age) {
25         this.name = name;
26         this.age = age;
27     }
28

```

```
32     }
33
34     public void setName(String name) {
35         this.name = name;
36     }
37
38     public int getAge() {
39         return age;
40     }
41
42     public void setAge(int age) {
43         this.age = age;
44     }
45 }
```

- DAO 获取 POJO, UserDaoServiceImpl (mock 数据)

```
1 package tech.pdai.springframework.dao;
2
3 import java.util.Collections;
4 import java.util.List;
5
6 import tech.pdai.springframework.entity.User;
7
8 /**
9  * @author pdai
10  */
11 public class UserDaoImpl {
12
13     /**
14      * init.
15      */
16     public UserDaoImpl() {
17     }
18
19     /**
20      * mocked to find user list.
21      *
22      * @return user list
23      */
24     public List<User> findUserList() {
25         return Collections.singletonList(new User("pdai", 18));
26     }
27 }
```

java

并增加daos.xml

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <beans xmlns="http://www.springframework.org/schema/beans"
3       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4       xsi:schemaLocation="http://www.springframework.org/schema/beans
5 http://www.springframework.org/schema/beans/spring-beans.xsd">
6     <bean id="userDao" class="tech.pdai.springframework.dao.UserDaoImpl">
7         <!-- additional collaborators and configuration for this bean go here -->
8     </bean>
9     <!-- more bean definitions for data access objects go here -->
10 </beans>
```

xml

java

```
1 package tech.pdai.springframework.service;
2
3 import java.util.List;
4
5 import tech.pdai.springframework.dao.UserDaoImpl;
6 import tech.pdai.springframework.entity.User;
7
8 /**
9  * @author pdai
10  */
11 public class UserServiceImpl {
12
13     /**
14      * user dao impl.
15      */
16     private UserDaoImpl userDao;
17
18     /**
19      * init.
20      */
21     public UserServiceImpl() {
22     }
23
24     /**
25      * find user list.
26      *
27      * @return user list
28      */
29     public List<User> findUserList() {
30         return this.userDao.findUserList();
31     }
32
33     /**
34      * set dao.
35      *
36      * @param userDao user dao
37      */
38     public void setUserDao(UserDaoImpl userDao) {
39         this.userDao = userDao;
40     }
41 }
```

并增加services.xml

xml

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <beans xmlns="http://www.springframework.org/schema/beans"
3       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4       xsi:schemaLocation="http://www.springframework.org/schema/beans
5 http://www.springframework.org/schema/beans/spring-beans.xsd">
6     <!-- services -->
7     <bean id="userService" class="tech.pdai.springframework.service.UserServiceImpl">
8         <property name="userDao" ref="userDao"/>
9         <!-- additional collaborators and configuration for this bean go here -->
10    </bean>
11    <!-- more bean definitions for services go here -->
12 </beans>
```

- 拦截所有service中的方法，并输出记录

```
3 import java.lang.reflect.Method;
4
5 import org.aspectj.lang.ProceedingJoinPoint;
6 import org.aspectj.lang.annotation.Around;
7 import org.aspectj.lang.annotation.Aspect;
8 import org.aspectj.lang.reflect.MethodSignature;
9 import org.springframework.context.annotation.EnableAspectJAutoProxy;
10
11 /**
12  * @author pdai
13  */
14 @Aspect
15 public class LogAspect {
16
17     /**
18      * aspect for every methods under service package.
19      */
20     @Around("execution(* tech.pdai.springframework.service.*.*(..))")
21     public Object businessService(ProceedingJoinPoint pjp) throws Throwable {
22         // get attribute through annotation
23         Method method = ((MethodSignature) pjp.getSignature()).getMethod();
24         System.out.println("execute method: " + method.getName());
25
26         // continue to process
27         return pjp.proceed();
28     }
29
30 }
```

并增加aspects.xml

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <beans xmlns="http://www.springframework.org/schema/beans"
3       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4       xmlns:aop="http://www.springframework.org/schema/aop"
5       xmlns:context="http://www.springframework.org/schema/context"
6       xsi:schemaLocation="http://www.springframework.org/schema/beans
7 http://www.springframework.org/schema/beans/spring-beans.xsd
8 http://www.springframework.org/schema/aop
9 http://www.springframework.org/schema/aop/spring-aop.xsd
10 http://www.springframework.org/schema/context
11 http://www.springframework.org/schema/context/spring-context.xsd
12 ">
13
14     <context:component-scan base-package="tech.pdai.springframework" />
15
16     <aop:aspectj-autoproxy/>
17
18     <bean id="logAspect" class="tech.pdai.springframework.aspect.LogAspect">
19         <!-- configure properties of aspect here as normal -->
20     </bean>
21     <!-- more bean definitions for data access objects go here -->
22 </beans>
```

xml

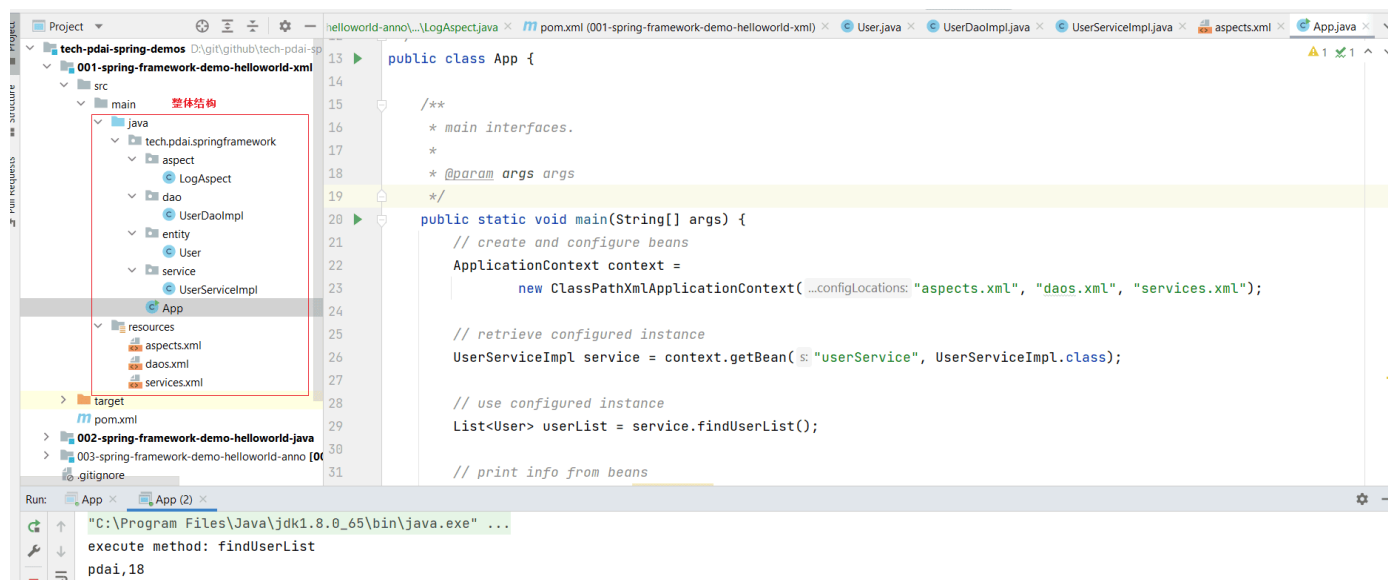
• 组装App

```
1 package tech.pdai.springframework;
2
3 import java.util.List;
```

java

```
7 import tech.pdai.springframework.entity.User;
8 import tech.pdai.springframework.service.UserServiceImpl;
9
10 /**
11  * @author pdai
12  */
13 public class App {
14
15     /**
16      * main interfaces.
17      *
18      * @param args args
19      */
20     public static void main(String[] args) {
21         // create and configure beans
22         ApplicationContext context =
23             new ClassPathXmlApplicationContext("aspects.xml", "daos.xml", "services.xml");
24
25         // retrieve configured instance
26         UserServiceImpl service = context.getBean("userService", UserServiceImpl.class);
27
28         // use configured instance
29         List<User> userList = service.findUserList();
30
31         // print info from beans
32         userList.forEach(a -> System.out.println(a.getName() + "," + a.getAge()));
33     }
34 }
```

• 整体结构和运行app



这个例子体现了Spring的哪些核心要点

那么Spring框架帮助我们做什么，它体现了什么哪些要点呢？

控制反转 - IOC

- 如果没有Spring框架，我们需要自己创建User/Dao/Service等，比如：

```
1 UserDaoImpl userDao = new UserDaoImpl();
2 UserServiceImpl userService = new UserServiceImpl();
3 userService.setUserDao(userDao);
4 List<User> userList = userService.findUserList();
```

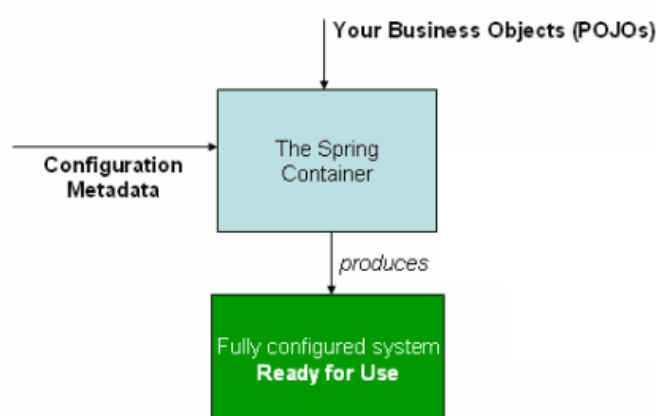
java

- 有了Spring框架，可以将原有Bean的创建工作转给框架，需要用时从Bean的容器中获取即可，这样便简化了开发工作

Bean的创建和使用分离了。

```
1 // create and configure beans
2 ApplicationContext context =
3     new ClassPathXmlApplicationContext("aspects.xml", "daos.xml", "services.xml");
4
5 // retrieve configured instance
6 UserServiceImpl service = context.getBean("userService", UserServiceImpl.class);
7
8 // use configured instance
9 List<User> userList = service.findUserList();
```

java



更进一步，你便能理解为何会有如下的知识点了：

1. Spring框架管理这些Bean的创建工作，即由用户管理Bean转变为框架管理Bean，这个就叫**控制反转** - Inversion of Control (IoC)
2. Spring 框架托管创建的Bean放在哪里呢？这便是IoC Container;
3. Spring 框架为了更好让用户配置Bean，必然会引入**不同方式来配置Bean？** 这便是xml配置，Java配置，注解配置等支持
4. Spring 框架既然接管了Bean的生成，必然需要**管理整个Bean的生命周期**等；
5. 应用程序代码从IoC Container中获取依赖的Bean，注入到应用程序中，这个过程叫 **依赖注入**(Dependency Injection, DI)；所以说控制反转是通过依赖注入实现的，其实它们是同一个概念的不同角度描述。通俗来说就是IoC是设计思想，DI是实现方式
6. 在依赖注入时，有哪些方式呢？这就是构造器方式，@Autowired, @Resource, @Qualifier... 同时Bean之间存在依赖（可能存在先后顺序问题，以及**循环依赖问题**等）

这边引入我们后续的相关文章：[Spring基础 - Spring之控制反转\(IoC\)](#)

面向切面 - AOP

来看第二个需求：**给Service所有方法调用添加日志**（调用方法时），本质上是解耦问题；

```
1  /**
2   * find user list.
3   *
4   * @return user list
5   */
6   public List<User> findUserList() {
7       System.out.println("execute method findUserList");
8       return this.userDao.findUserList();
9   }
```

java

- 有了Spring框架，通过@Aspect注解 定义了切面，这个切面中定义了拦截所有service中的方法，并记录日志；可以明显看到，框架将日志记录和业务需求的代码解耦了，不再是侵入式的了

```
1  /**
2   * aspect for every methods under service package.
3   */
4   @Around("execution(* tech.pdai.springframework.service.*(..))")
5   public Object businessService(ProceedingJoinPoint pjp) throws Throwable {
6       // get attribute through annotation
7       Method method = ((MethodSignature) pjp.getSignature()).getMethod();
8       System.out.println("execute method: " + method.getName());
9
10      // continue to process
11      return pjp.proceed();
12  }
```

java

更进一步，你便能理解为何会有如下的知识点了：

- Spring 框架通过定义切面，通过拦截切点实现了不同业务模块的解耦，这个就叫**面向切面编程 - Aspect Oriented Programming (AOP)**
- 为什么@Aspect注解使用的是aspectj的jar包呢？这就引出了**Aspect4J和Spring AOP的历史渊源**，只有理解了Aspect4J和Spring的渊源才能理解有些注解上的兼容设计
- 如何支持**更多拦截方式**来实现解耦，以满足更多场景需求呢？这就是@Around, @Pointcut... 等的设计
- 那么Spring框架又是如何实现AOP的呢？这就引入**代理技术**，分**静态代理和动态代理**，动态代理又包含JDK代理和CGLIB代理等

这边引入我们后续的相关文章：[Spring基础 - Spring之面向切面编程\(AOP\)](#)

Spring框架设计如何逐步简化开发的

通过上述的框架介绍和例子，已经初步知道了Spring设计的两个大的要点：IOC和AOP；从框架的设计角度而言，更为重要的是简化开发，比如提供更为便捷的配置Bean的方式，直至0配置（即约定大于配置）。这里我将通过Spring历史版本的发展，和SpringBoot的推出等，来帮你理解Spring框架是如何逐步简化开发的。

Java 配置方式改造

在前文的例子中，通过xml配置方式实现的，这种方式实际上比较麻烦；我通过Java配置进行改造：

- User, UserDaoImpl, UserServiceImpl, LogAspect不用改
- 将原通过.xml配置转换为Java配置

```
1  package tech.pdai.springframework.config;
2
3  import org.springframework.context.annotation.Bean;
4  import org.springframework.context.annotation.Configuration;
```

java

```
8      import tech.pdai.springframework.service.UserServiceImpl;
9
10     /**
11      * @author pdai
12      */
13     @EnableAspectJAutoProxy
14     @Configuration
15     public class BeansConfig {
16
17         /**
18          * @return user dao
19          */
20         @Bean("userDao")
21         public UserDaoImpl userDao() {
22             return new UserDaoImpl();
23         }
24
25         /**
26          * @return user service
27          */
28         @Bean("userService")
29         public UserServiceImpl userService() {
30             UserServiceImpl userService = new UserServiceImpl();
31             userService.setUserDao(userDao());
32             return userService;
33         }
34
35         /**
36          * @return log aspect
37          */
38         @Bean("logAspect")
39         public LogAspect logAspect() {
40             return new LogAspect();
41         }
42     }
```

- 在App中加载BeansConfig的配置

```
1      package tech.pdai.springframework;
2
3      import java.util.List;
4
5      import org.springframework.context.annotation.AnnotationConfigApplicationContext;
6      import tech.pdai.springframework.config.BeansConfig;
7      import tech.pdai.springframework.entity.User;
8      import tech.pdai.springframework.service.UserServiceImpl;
9
10     /**
11      * @author pdai
12      */
13     public class App {
14
15         /**
16          * main interfaces.
17          *
18          * @param args args
19          */
20         public static void main(String[] args) {
21             // create and configure beans
22             AnnotationConfigApplicationContext context = new AnnotationConfigApplicationContext(BeansConfig.class);
23
24         }
```

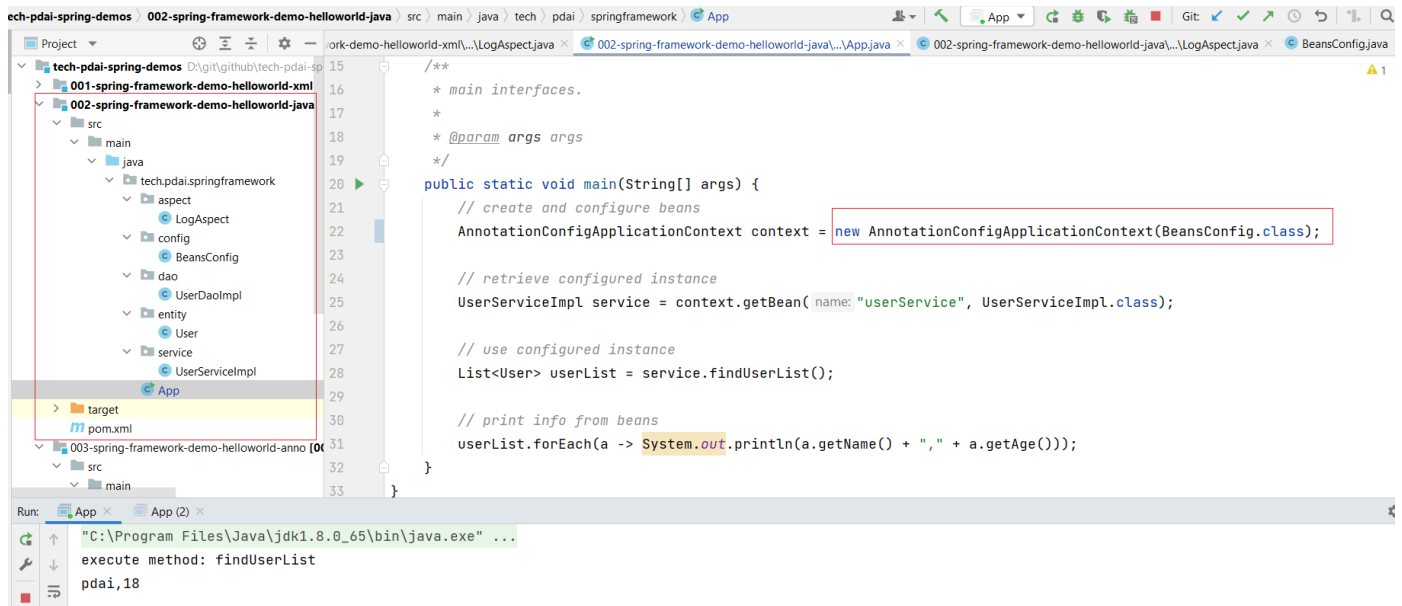
java

```

27
28     // use configured instance
29     List<User> userList = service.findUserList();
30
31     // print info from beans
32     userList.forEach(a -> System.out.println(a.getName() + "," + a.getAge()));
33 }

```

• 整体结构和运行app



注解配置方式改造

更进一步, Java 5开始提供注解支持, Spring 2.5 开始完全支持基于注解的配置并且也支持JSR250 注解。在Spring后续的版本发展倾向于通过注解和Java配置结合使用。

• BeanConfig 不再需要Java配置

```

1 package tech.pdai.springframework.config;
2
3 import org.springframework.context.annotation.ComponentScan;
4 import org.springframework.context.annotation.ComponentScans;
5 import org.springframework.context.annotation.Configuration;
6 import org.springframework.context.annotation.EnableAspectJAutoProxy;
7
8 /**
9  * @author pdai
10  */
11 @Configuration
12 @EnableAspectJAutoProxy
13 public class BeansConfig {
14
15 }

```

java

• UserDaoImpl 增加了 @Repository注解

```
3    */
4    @Repository
5    public class UserDaoImpl {
6
7        /**
8         * mocked to find user list.
9         *
10         * @return user list
11         */
12        public List<User> findUserList() {
13            return Collections.singletonList(new User("pdai", 18));
14        }
15    }
```

- UserServiceImpl 增加了@Service 注解，并通过@Autowired注入userDao.

```
1    /**
2     * @author pdai
3     */
4    @Service
5    public class UserServiceImpl {
6
7        /**
8         * user dao impl.
9         */
10        @Autowired
11        private UserDaoImpl userDao;
12
13        /**
14         * find user list.
15         *
16         * @return user list
17         */
18        public List<User> findUserList() {
19            return userDao.findUserList();
20        }
21    }
22 }
```

java

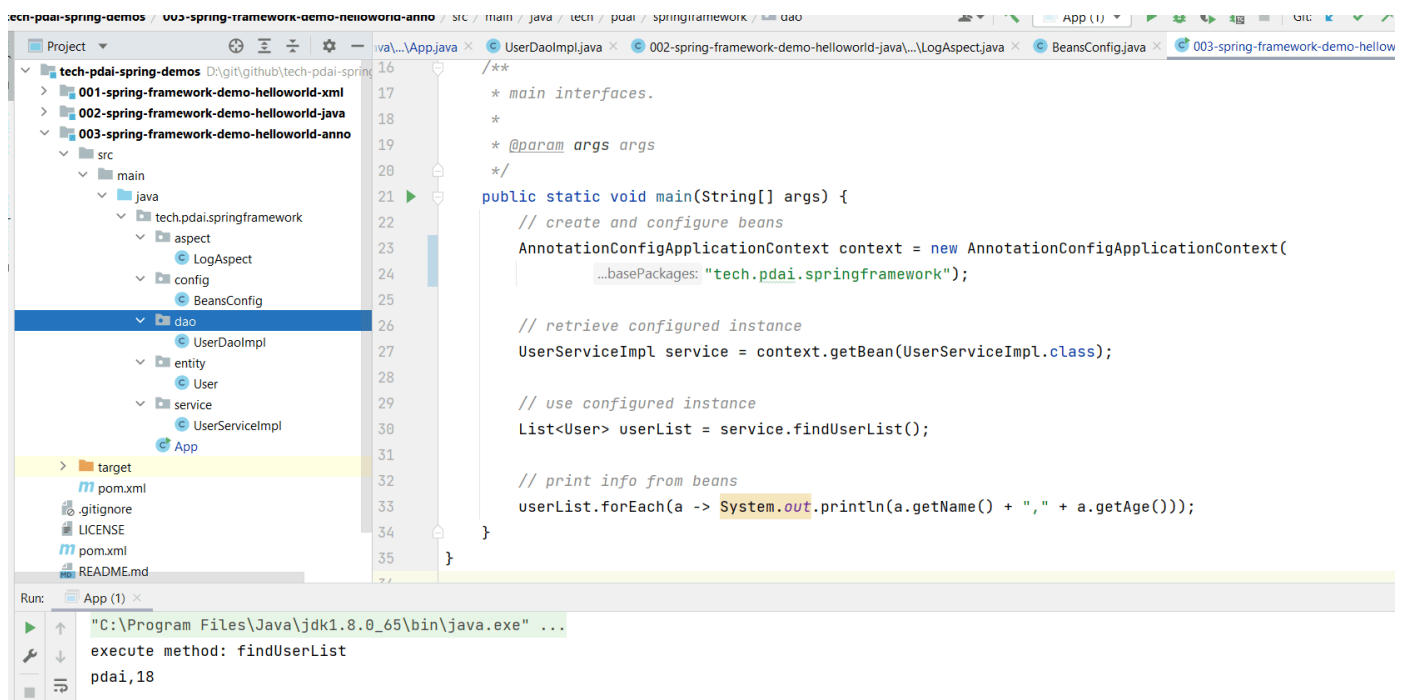
- 在App中扫描tech.pdai.springframework包

```
1    package tech.pdai.springframework;
2
3    import java.util.List;
4
5    import org.springframework.context.annotation.AnnotationConfigApplicationContext;
6    import tech.pdai.springframework.entity.User;
7    import tech.pdai.springframework.service.UserServiceImpl;
8
9    /**
10     * @author pdai
11     */
12    public class App {
13
14        /**
15         * main interfaces.
16         *
17         * @param args args
18         */
```

java

```
22         "tech.pdai.springframework");
23
24         // retrieve configured instance
25         UserServiceImpl service = context.getBean(UserServiceImpl.class);
26
27         // use configured instance
28         List<User> userList = service.findUserList();
29
30         // print info from beans
31         userList.forEach(a -> System.out.println(a.getName() + "," + a.getAge()));
32     }
33 }
```

• 整体结构和运行app



SpringBoot托管配置

Springboot实际上通过约定大于配置的方式，使用xx-starter统一的对Bean进行默认初始化，用户只需要很少的配置就可以进行开发了。

这个因为很多开发者都是从SpringBoot开始着手开发的，所以这个比较好理解。我们需要的是将知识点都串联起来，构筑认知体系。

结合Spring历史版本和SpringBoot看发展

最后结合Spring历史版本总结下它的发展：

(这样是不是能够帮助你在整体上构建了知识体系的认知了呢？)



PS：相关代码，可以通过[这里](#)直接查看

```
001-spring-framework-demo-helloworld.xml > src > main > java > tech > pdai > springframework > App.java > ...
1 package tech.pdai.springframework;
2
3 import java.util.List;
4
5 import org.springframework.context.ApplicationContext;
6 import org.springframework.context.support.ClassPathXmlApplicationContext;
7 import tech.pdai.springframework.entity.User;
8 import tech.pdai.springframework.service.UserServiceImpl;
9
10 /**
11  * @author pdai
12  */
13 public class App {
14
15     /**
16      * main interfaces.
17      *
18      * @param args args
19      */
20     public static void main(String[] args) {
21         // create and configure beans
22         ApplicationContext context =
23             new ClassPathXmlApplicationContext("aspects.xml", "daos.xml", "services.xml");
24
25         // retrieve configured instance
26         UserServiceImpl service = context.getBean("userService", UserServiceImpl.class);
27
28         // use configured instance
29         List<User> userList = service.findUserList();
30
31         // print info from beans
32         userList.forEach(a -> System.out.println(a.getName() + ", " + a.getAge()));
33     }
34
35 }
```

我要纠错

← **Spring基础 - Spring和Spring框架组成**

Spring基础 - Spring核心之控制反转(IOC) →

苏ICP备19053722号 | pdai | copyright © 2017-present