spring初识

1、框架

框架就是一些类和接口的集合，通过这些类和接口协调来完成一系列的程序实现。JAVA框架可以分为三层：表示层，业务层和物理层。框架又叫做开发中的半成品，它不能提供整个WEB应用程序的所有东西，但是有了框架，我们就可以集中精力进行业务逻辑的开发而不用去关心它的技术实现以及一些辅助的业务逻辑。大家熟知的Structs和Spring就是表示层和业务层框架的强力代表。（说的太官方了）

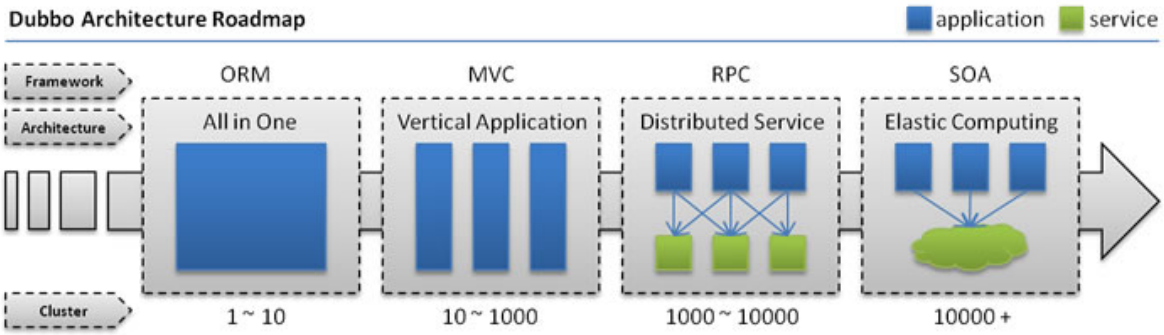
人话：

框架就是某些个人或者组织定义了一系列的类或者接口，提前定义好了一些实现，用户可以在这些类和接口的基础之上，使用这些类来迅速的形成某个领域或者某个行业的解决方案，简化开发的过程，提高开发的效率。就好比：你要盖一座房子，先把柱子，房梁等先建设好，然后只需要向房子中填充就可以了，可以按照自己的需求进行设计，其实我们做的项目、系统都是类似的方式，如果所有的代码全部都需要自己实现，那么这个工程就太庞大了，所以可以先创建出一些基础的模板框架，开发人员只需要按照自己的需求向架子中填充内容，完成自己独特需求即可，这就是框架存在的意义。其实我们之前定义的简单的工具类这些东西也是类似的原理，只不过比较单一简单而已，因此，在现在的很多项目系统开发的过程中都是利用框架进行开发。

2、spring（春天）

架构设计

随着互联网的发展，网站应用的规模不断扩大，常规的垂直应用架构已无法应对，分布式服务架构以及流动计算架构势在必行，亟需一个治理系统确保架构有条不紊的演进。



单一应用架构

当网站流量很小时，只需一个应用，将所有功能都部署在一起，以减少部署节点和成本。此时，用于简化增删改查工作量的数据访问框架(ORM)是关键。

垂直应用架构

当访问量逐渐增大，单一应用增加机器带来的加速度越来越小，提升效率的方法之一是将应用拆成互不相干的几个应用，以提升效率。此时，用于加速前端页面开发的Web框架(MVC)是关键。

分布式服务架构

当垂直应用越来越多，应用之间交互不可避免，将核心业务抽取出来，作为独立的服务，逐渐形成稳定的服务中心，使前端应用能更快速的响应多变的市场需求。此时，用于提高业务复用及整合的分布式服务框架(RPC)是关键。

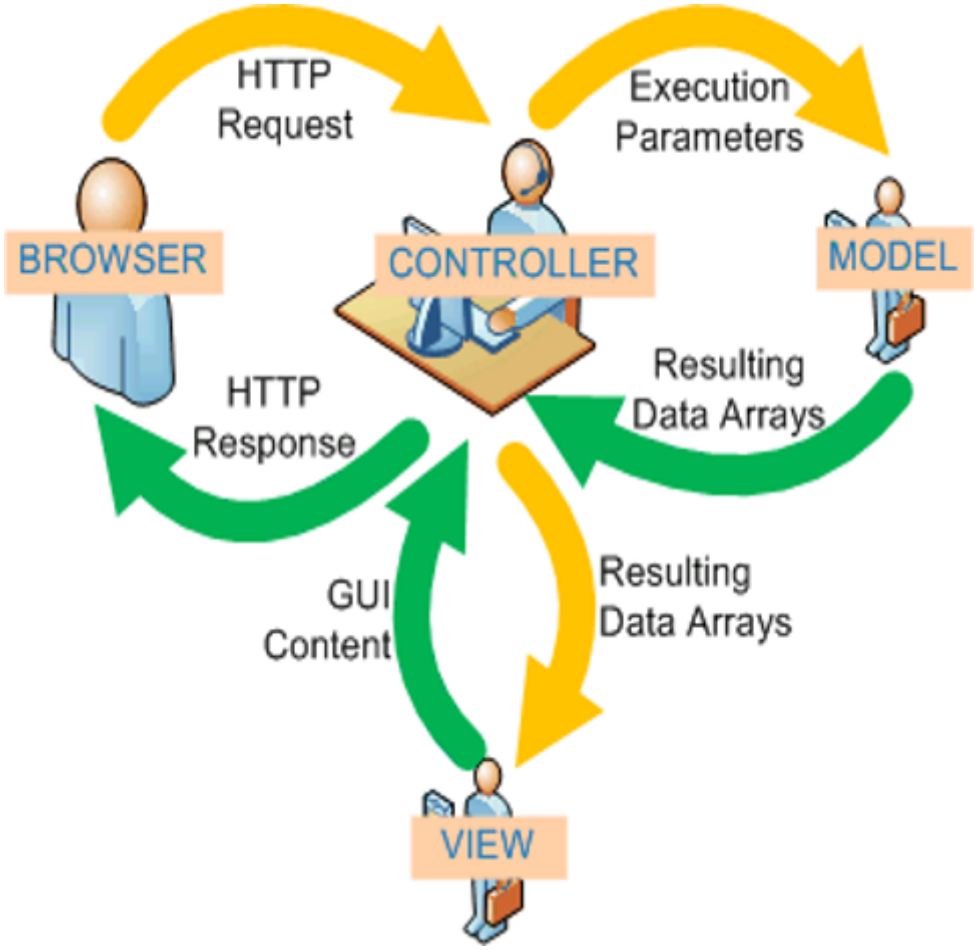
流动计算架构

当服务越来越多，容量的评估，小服务资源的浪费等问题逐渐显现，此时需增加一个调度中心基于访问压力实时管理集群容量，提高集群利用率。此时，用于提高机器利用率的资源调度和治理中心(SOA)是关键。

Java主流框架演变之路

1、JSP+Servlet+JavaBean

2、MVC三层架构



3、使用EJB进行应用的开发，但是EJB是重量级框架（在使用的时候，过多的接口和依赖，侵入性强），在使用上比较麻烦

4、Struts1/Struts2+Hibernate+Spring

5、SpringMVC+Mybatis+Spring

6、SpringBoot开发，约定大于配置

Spring官网

官网地址：https://spring.io/projects/spring-framework#overview

压缩包下载地址：https://repo.spring.io/release/org/springframework/spring/

源码地址：https://github.com/spring-projects/spring-framework

```
Spring makes it easy to create Java enterprise applications. It provides
everything you need to embrace the Java language in an enterprise environment,
with support for Groovy and Kotlin as alternative languages on the JVM, and with
the flexibility to create many kinds of architectures depending on an
application’s needs. As of Spring Framework 5.1, Spring requires JDK 8+ (Java SE
8+) and provides out-of-the-box support for JDK 11 LTS. Java SE 8 update 60 is
suggested as the minimum patch release for Java 8, but it is generally
recommended to use a recent patch release.

Spring supports a wide range of application scenarios. In a large enterprise,
applications often exist for a long time and have to run on a JDK and
application server whose upgrade cycle is beyond developer control. Others may
run as a single jar with the server embedded, possibly in a cloud environment.
Yet others may be standalone applications (such as batch or integration
workloads) that do not need a server.

Spring is open source. It has a large and active community that provides
continuous feedback based on a diverse range of real-world use cases. This has
helped Spring to successfully evolve over a very long time.

Spring 使创建 Java 企业应用程序变得更加容易。它提供了在企业环境中接受 Java 语言所需的一切,,
并支持 Groovy 和 Kotlin 作为 JVM 上的替代语言，并可根据应用程序的需要灵活地创建多种体系结
构。 从 Spring Framework 5.0 开始，Spring 需要 JDK 8(Java SE 8+)，并且已经为 JDK 9 提
供了现成的支持。

Spring支持各种应用场景， 在大型企业中，应用程序通常需要运行很长时间，而且必须运行在 jdk 和应用
服务器上，这种场景开发人员无法控制其升级周期。 其他可能作为一个单独的jar嵌入到服务器去运行，也有
可能在云环境中。还有一些可能是不需要服务器的独立应用程序(如批处理或集成的工作任务)。

Spring 是开源的。它拥有一个庞大而且活跃的社区，提供不同范围的，真实用户的持续反馈。这也帮助
Spring不断地改进,不断发展。
```

核心解释

spring是一个开源框架。

spring是为了简化企业开发而生的，使得开发变得更加优雅和简洁。

spring是一个**IOC**和**AOP**的容器框架。

IOC：控制反转

AOP：面向切面编程

容器：包含并管理应用对象的生命周期，就好比用桶装水一样，spring就是桶，而对象就是水

使用spring的优点

1、Spring通过DI、AOP和消除样板式代码来简化企业级Java开发

2、Spring框架之外还存在一个构建在核心框架之上的庞大生态圈，它将Spring扩展到不同的领域，如Web服务、REST、移动开发以及NoSQL

3、低侵入式设计，代码的污染极低

4、独立于各种应用服务器，基于Spring框架的应用，可以真正实现Write Once,Run Anywhere的承诺

5、Spring的IoC容器降低了业务对象替换的复杂性，提高了组件之间的解耦

6、Spring的AOP支持允许将一些通用任务如安全、事务、日志等进行集中式处理，从而提供了更好的复用

7、Spring的ORM和DAO提供了与第三方持久层框架的的良好整合，并简化了底层的数据库访问

8、Spring的高度开放性，并不强制应用完全依赖于Spring，开发者可自由选用Spring框架的部分或全部

如何简化开发

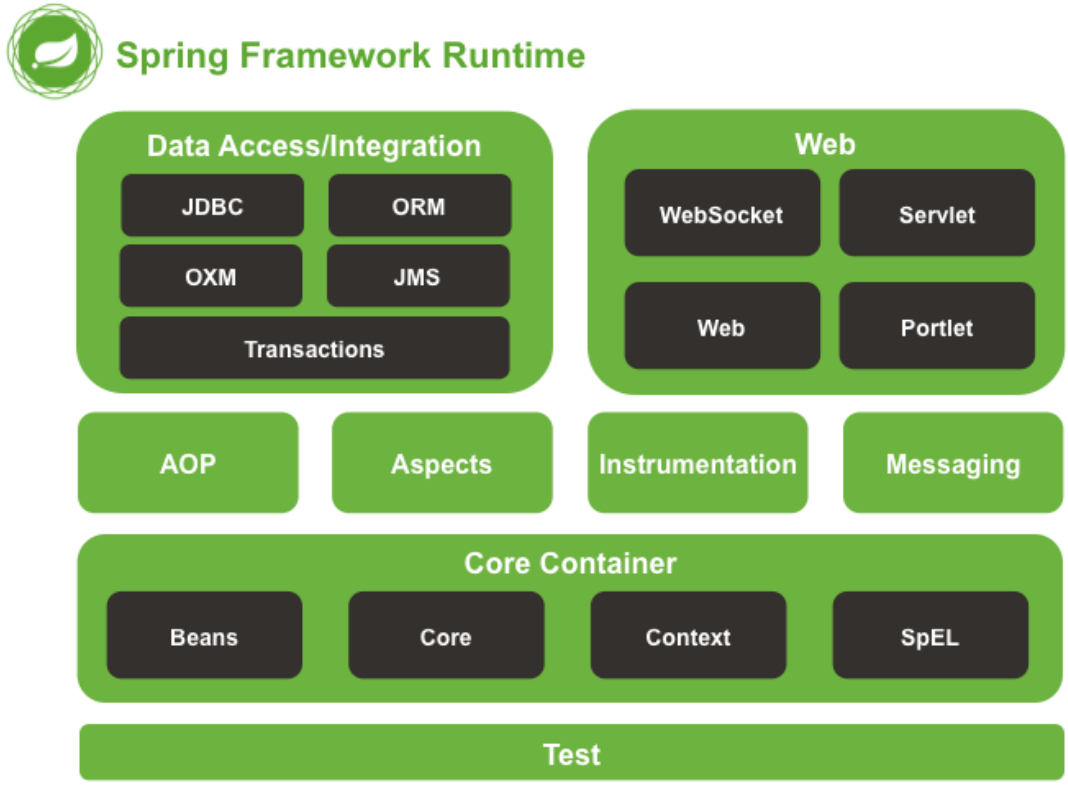
基于POJO的轻量级和最小侵入性编程

通过依赖注入和面向接口实现松耦合

基于切面和惯例进行声明式编程

通过切面和模板减少样板式代码

spring的模块划分图



模块解释：
Test:Spring的单元测试模块
Core Container:核心容器模块
AOP+Aspects:面向切面编程模块
Instrumentation:提供了class instrumentation支持和类加载器的实现来在特定的应用服务器上使用,几乎不用
Messaging:包括一系列的用来映射消息到方法的注解,几乎不用
Data Access/Integration:数据的获取/整合模块，包括了JDBC,ORM,OXM,JMS和事务模块
Web:提供面向web整合特性

3、IOC（Inversion of Control）:控制反转

为什么要引入IOC

创建一个普通的java项目，完成下述功能

UserDao.java

```
package com.mashibing.dao;

public interface UserDao {
    public void getUser();
}
```

UserDaoImpl.java

```
package com.mashibing.dao.impl;

import com.mashibing.dao.UserDao;

public class UserDaoImpl  implements UserDao {
    @Override
    public void getUser() {
        System.out.println("获取用户数据");
    }
}
```

UserService.java

```
package com.mashibing.service;

public interface UserService {
    public void getUser();
}
```

UserServiceImpl.java

```
package com.mashibing.service.impl;

import com.mashibing.dao.UserDao;
import com.mashibing.dao.impl.UserDaoImpl;
import com.mashibing.dao.impl.UserDaoMysqlImpl;
import com.mashibing.service.UserService;

public class UserServiceImpl implements UserService {

    private UserDao userDao = new UserDaoImpl();

    @Override
    public void getUser() {
        userDao.getUser();
    }
}
```

SpringDemoTest.java

```
package com.mashibing.test;

import com.mashibing.service.UserService;
import com.mashibing.service.impl.UserServiceImpl;

public class SpringDemoTest {
    public static void main(String[] args) {
       UserService service = new UserServiceImpl();
       service.getUser();
    }
}
```

在之前的代码编写过程中，我们都是这么完成我们的功能的，但是如果增加一个UserDao的实现类呢?

UserDaoMysqlImpl.java

```
package com.mashibing.dao.impl;

import com.mashibing.dao.UserDao;

public class UserDaoMysqlImpl implements UserDao {
    @Override
    public void getUser() {
        System.out.println("mysql");
    }
}
```

如果我们想要使用mysql的话，那么就必须要修改UserServiceImpl.java的代码：

```
package com.mashibing.service.impl;

import com.mashibing.dao.UserDao;
import com.mashibing.dao.impl.UserDaoImpl;
import com.mashibing.dao.impl.UserDaoMysqlImpl;
import com.mashibing.service.UserService;

public class UserServiceImpl implements UserService {

    private UserDao userDao = new UserDaoImpl();

    @Override
    public void getUser() {
        userDao.getUser();
    }
}
```

但是如果我们再增加一个oracle的类呢?

UserDaoOracleImpl.java

```
package com.mashibing.dao.impl;

import com.mashibing.dao.UserDao;

public class UserDaoOracleImpl implements UserDao {
    @Override
    public void getUser() {
        System.out.println("oracle");
    }
}
```

此时UserService还是要继续修改，很显然这样的方式已经不适用于我们的需求了，那么怎么解决呢，可以使用如下的方式

UserServiceImpl.java

```
package com.mashibing.service.impl;

import com.mashibing.dao.UserDao;
import com.mashibing.dao.impl.UserDaoImpl;
import com.mashibing.service.UserService;

public class UserServiceImpl implements UserService {
    private UserDao userDao;

    public void setUserDao(UserDao userDao){
        this.userDao = userDao;
    }
    @Override
    public void getUser() {
        userDao.getUser();
    }
}
```

测试类SpringDemoTest.java

```
package com.mashibing.test;

import com.mashibing.dao.impl.UserDaoMysqlImpl;
import com.mashibing.dao.impl.UserDaoOracleImpl;
import com.mashibing.service.UserService;
import com.mashibing.service.impl.UserServiceImpl;

public class SpringDemoTest {
    public static void main(String[] args) {
        UserServiceImpl userService = new UserServiceImpl();
        userService.setUserDao(new UserDaoMysqlImpl());
        userService.getUser();

        userService.setUserDao(new UserDaoOracleImpl());
        userService.getUser();
    }
}
```

其实从刚刚的代码中，大家应该能体会解耦的重要性了，下面我们就开始学习Spring的IOC。

IOC初始

想要搞明白IOC，那么需要搞清楚如下几个问题：

```
1、谁控制谁
2、控制什么
3、什么是反转
4、哪些方面被反转
```

基本概念

```
    IoC is also known as dependency injection (DI). It is a process whereby objects define their dependencies (that is, the other objects they work with) only through constructor arguments, arguments to a factory method, or properties that are set on the object instance after it is constructed or returned from a factory method. The container then injects those dependencies when it creates the bean. This process is fundamentally the inverse (hence the name, Inversion of Control) of the bean itself controlling the instantiation or location of its dependencies by using direct construction of classes or a mechanism such as the Service Locator pattern.
    IOC与大家熟知的依赖注入同理，. 这是一个通过依赖注入对象的过程 也就是说，它们所使用的对象，是通过构造函数参数，工厂方法的参数或这是从工厂方法的构造函数或返回值的对象实例设置的属性，然后容器在创建bean时注入这些需要的依赖。 这个过程相对普通创建对象的过程是反向的（因此称之为IoC），bean本身通过直接构造类来控制依赖关系的实例化或位置，或提供诸如服务定位器模式之类的机制。
```

如果这个过程比较难理解的话，那么可以想象自己找女朋友和婚介公司找女朋友的过程。如果这个过程能够想明白的话，那么我们现在回答上面的问题：

```
1、谁控制谁：在之前的编码过程中，都是需要什么对象自己去创建什么对象，有程序员自己来控制对象，而有了IOC容器之后，就会变成由IOC容器来控制对象，
2、控制什么：在实现过程中所需要的对象及需要依赖的对象
3、什么是反转：在没有IOC容器之前我们都是在对象中主动去创建依赖的对象，这是正转的，而有了IOC之后，依赖的对象直接由IOC容器创建后注入到对象中，由主动创建变成了被动接受，这是反转
4、哪些方面被反转：依赖的对象
```

DI与IOC

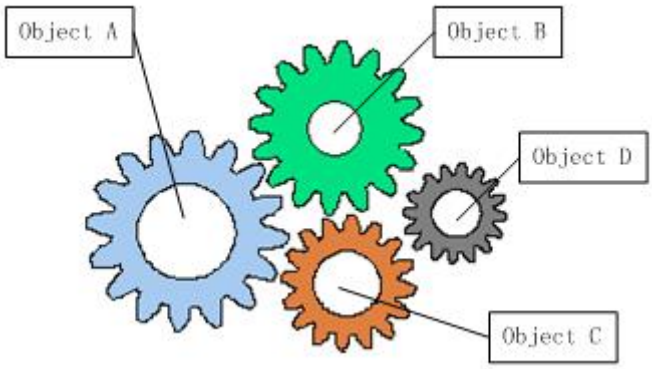
```
很多人把IOC和DI说成一个东西，笼统来说的话是没有问题的，但是本质上还是有所区别的,希望大家能够严谨一点，IOC和DI是从不同的角度描述的同一件事，IOC是从容器的角度描述，而DI是从应用程序的角度来描述，也可以这样说，IOC是设计思想，而DI是具体的实现方式
```

4、总结

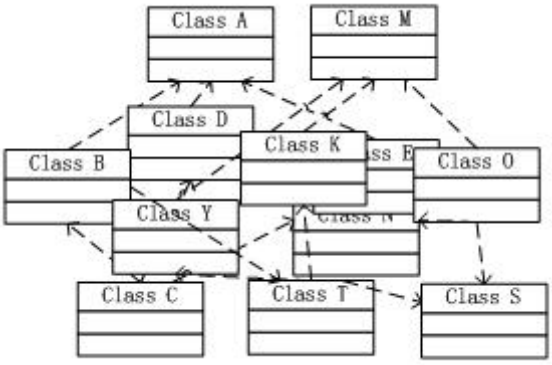
在此处总结中，希望大家能够能够明白两件事：

1、解耦

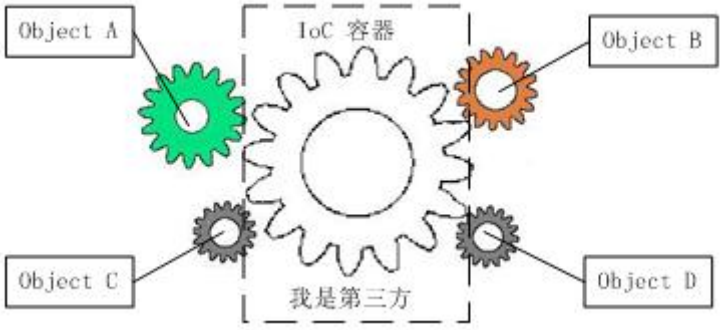
在面向对象设计的软件系统中，底层的实现都是由N个对象组成的，所有的对象通过彼此的合作，最终实现系统的业务逻辑。



需要注意的是，在这样的组合关系中，一旦某一个对象出现了问题，那么其他对象肯定回有所影响，这就是耦合性太高的缘故，但是对象的耦合关系是无法避免的，也是必要的。随着应用程序越来越庞大，对象的耦合关系可能越来越复杂，经常需要多重依赖关系，因此，无论是架构师还是程序员，在面临这样的场景的时候，都需要减少这些对象的耦合性。



耦合的关系不仅仅是对象与对象之间，也会出现在软件系统的各个模块之间，是我们需要重点解决的问题。而为了解决对象之间的耦合度过高的问题，我们就可以通过IOC来实现对象之间的解耦，spring框架就是IOC理论最最广泛的应用。



从上图中可以看到，当引入了第三方的容器之后，几个对象之间就没有了耦合关系，全部对象都交由容器来控制，这个容器就相当于粘合剂，将系统的对象粘合在一起发挥作用。

2、生态

任何一个语言或者任何一个框架想要立于不败之地，那么很重要的就是它的生态。