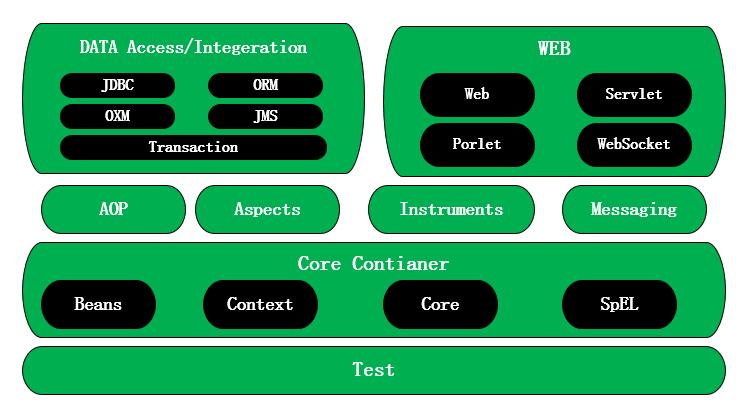
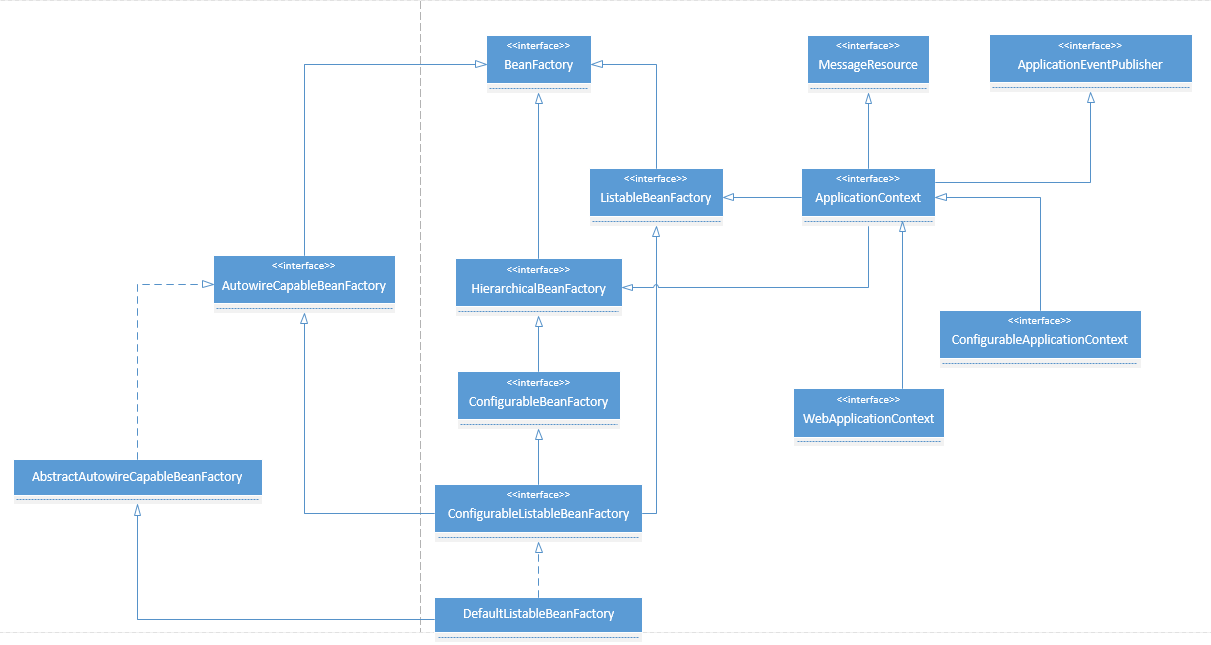
**SpringFramework**

1. **Spring体系结构**

Spring核心容器(IOC)：提供Spring框架的基本功能，管理着Spring应用中bean的创建、配置和管理。核心容器的主要组件BeanFactory和ApplicationContext，都可以看成容器的具体表现形式。

Spring提供的基本IOC容器的接口和定义的基础之上，spring通过定义BeanDefinition来管理各种spring对象之间的依赖关系。BeanDefinition抽象bean的定义，对于IOC容器来说，BeanDefinition就是对依赖反转模式管理依赖关系的的数据抽象，也是容器实现依赖反转功能的核心数据结构。

IOC容器设计的主要源码在Spring-beans和Spring-context这两个包中，主要接口设计如下：



第一条设计主线是：BeanFactory作为顶层接口类，它的三个子接口为ListableBeanFactory(说明这些bean是可列表的)，HierarchicalBeanFactory(表示这些bean是有继承关系的，每个bean有可能有父类bean，增加getParentBeanFactory()功能后，使得BeanFactory具备了双亲IOC容器的管理功能)，AutowireCapableBeanFactory(定义bean的自动装配规则)，默认实现类是DefaultListableBeanFactory。

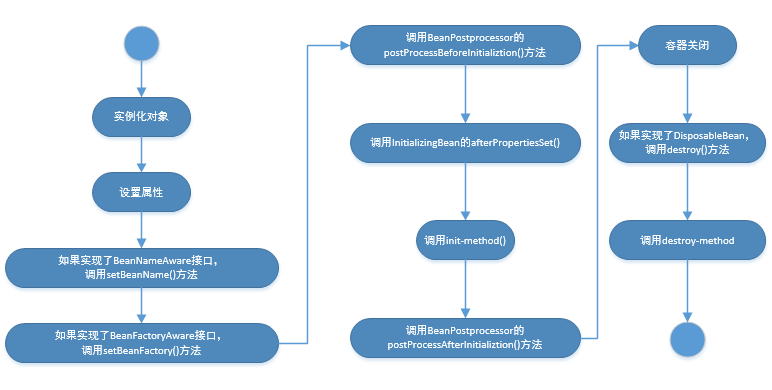
第二条设计主线是：以ApplicationContext应用上下文为核心的设计。涉及的主要接口类有BeanFactory🡪ListableBeanFactory🡪ApplicationContext🡪 ConfigurableApplicationContext(WebApplicationContext)，ApplicationContext继承MessageSource(支持信息源，实现国际化)，ResourcePatternResolver(访问资源)，ApplicationEventPublisher(支持响应时间)

BeanFactory和FactoryBean的区别：

BeanFactory是IOC容器的编程抽象，主要是管理对象的创建、配置和建立对象之间的依赖

FactoryBean是工厂bean，可以看成是一个抽象工厂，对FactoryBean的调用返回的是工厂产生的bean(使用该容器时不会返回factory bean本身，而是返回其生成的对象)，所有的Factory bean都实现了FactoryBean接口，spring包括了大部分通用资源和服务的抽象(JNDI查询的处理，对代理对象的处理，对事务性的处理，对RMI代理的处理)

1. **Spring Bean的生命周期**



**3.Spring aop原理**

实现aop的技术主要是：JDK代理和CGLIB代理

相关概念：

1).方面(Aspect)：关注点的模块化

2).连接点(JoinPoint)：程序执行过程中明确的点，如方法的调用

3).通知(Advice)：特点的连接点，AOP框架执行的动作，spring定义了四个Advice（BeforeAdvice、AfterAdvice、ThrowAdvice、DynamicIntroductionAdvice）

4).切入点(Pointcut)：指定一个通知将被引发的一系列连接点的集合，如：使用正则表达式

5).引入(Instroduction)：添加方法或字段到被通知的类。

6).目标对象(TargetObject)：包含连接点的对象（也称作被通知或代理对象）

7).AOP代理(AOP Proxy)：AOP框架创建的对象（JDK或者CGLIB代理）

8).织入(Weaving)：组装方面来创建一个被通知对象（Aspectj编译器可在编译期完成，Spring和其他java aop框架在运行时完成织入）

**4.MVC设计思想**

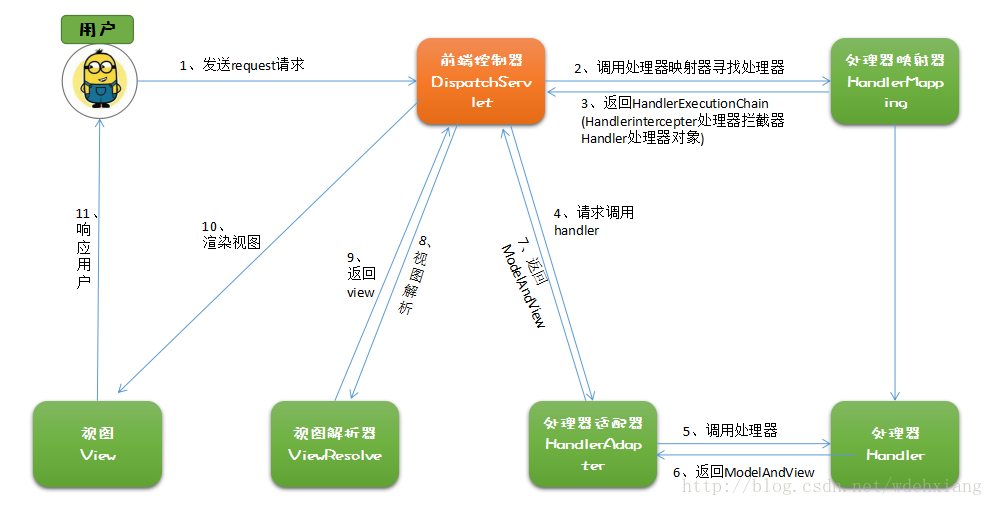
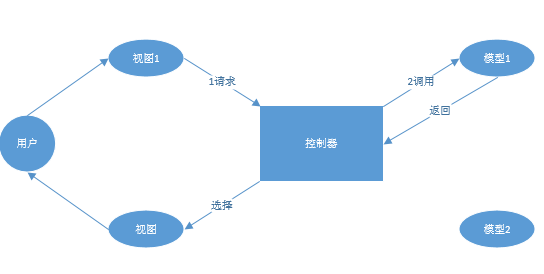
MVC(Model-View-Controller)：即把一个应用的输入、处理和输出流程按照Model、Controller、View的方式进行分离，将应用分为模型层，控制层和视图层的三层结构。

Model：模型层—应用程序中用于处理应用数据逻辑的部分

View：视图层—应用中用于展示处理数据显示的部分

Controller：控制层—应用中处理用户交互的部分

MVC视图：

****Spring MVC工作原理：

1).用户发起请求到前端控制器DispatchServlet

2).前端控制器调用HandlerMapping处理器映射器

3). 处理器映射找到具体的处理器，生成处理器对象及处理器拦截器对象（HandlerInterceptor）

4).前端控制器调用处理器适配器HandlerAdapter

5).HandlerAdapter调用具体的处理器(controller,后端处理器)，返回ModelAndView

6).前端控制器处理将ModelAndView返回给视图解析器ViewResolver

7).ViewResolver解析后返回具体的View

8).前端控制器根据view进行视图渲染

9).DispatchServlet响应用户，返回视图

**5、Tomcat容器原理**

Tomcat服务器的启动

入口是Bootstrap的main方法，main调用Bootstrap的init方法利用反射调用Catalina中的load方法进行tomcat的server(StandardServer)、service(StandardService)、container(Engine、Host、Context、Wrapper)、connector等初始化

**Tomcat处理一个http请求的过程**：

假设请求来自<http://localhost:8080/bbb/ccc.jsp>

1).请求发送到端口8080，被侦听coyote HTTP/1.1的connector获得

2).connector将请求交给它所在的service的Container的Engine处理，并等待Engine的回应

3).Engine获取请求中的host，并匹配Engine所拥有的host

4).Engine匹配到名localhost所在的host(如果匹配不到也把请求交给该host，因为localhost为该Engine默认的主机)

5).localhost获得请求“/bbb/ccc.jsp”，匹配该localhost所拥有的context

6).host匹配到路径为“/bbb/ccc.jsp”的context(未匹配到则将请求交给路径为””的context处理)

7).path=”/bbb”的context请求获得“/ccc.jsp”，在它的mapping table中寻找对应的servlet

8).context匹配到URL

9).构造HttpServletRequest和HttpServletResponse对象，作为参数调用tomcat内置JspServlet的service

10).context将执行完之后的HttpServletResponse对象返回给host

11).host将HttpServletResponse对象返回给engine

12).engine将HttpServletResponse返回给connector

13).connector将HttpServletResponse对象返回给客户browser

**6、Mybatis架构**

1、传统的数据库连接步骤及查询

**传统数据库连接：**

1. 加载数据库驱动
2. 创建数据库连接
3. 创建JDBC Statement对象
4. 设置SQL语句的传入参数
5. 执行SQL语句并获取查询结果
6. 对查询结果进行转换并返回
7. 释放相关资源(关闭connection、Statement、ResultSet等)

**查询步骤：**

1. 创建Connection对象：Connection con = DriverManager.getConnection(相关参数)
2. 创建Statement对象：Statement st = con.prepareStatement(sql语句)
3. 设置SQL语句中的参数：st.setXXX(“”, “”)
4. 创建ResultSet对象：ResultSet rs = st.executeXXX()
5. 对rs中的结果进行提取
6. 释放资源（rs.close(),st.close(),con.close()）,资源的释放和资源的创建顺序相反
7. 返回结果

Mybatis架构：支持普通SQL查询，存储过程和高级映射的持久层框架。

Mybatis底层依然采用原生JDBC对数据库操作，通过SqlSessionFactory、SqlSession、Executor、StatementHandler、ParameterHandler、ResultHandler、TypeHandler几个处理器对这些JDBC数据库操作过程进行了封装。