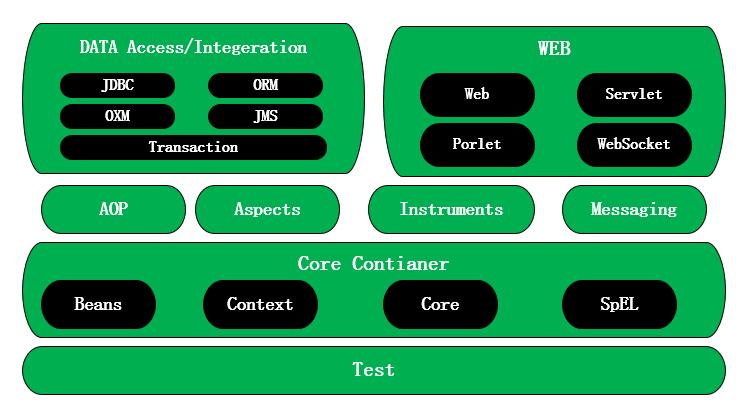
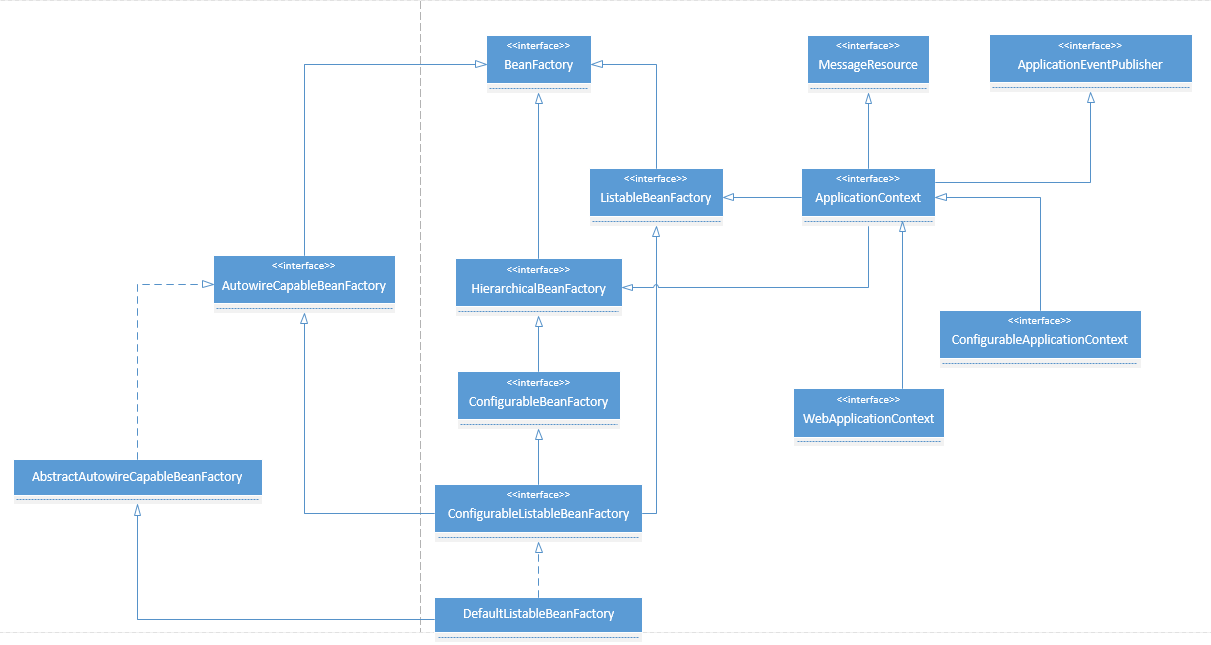
**SpringFramework**

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。

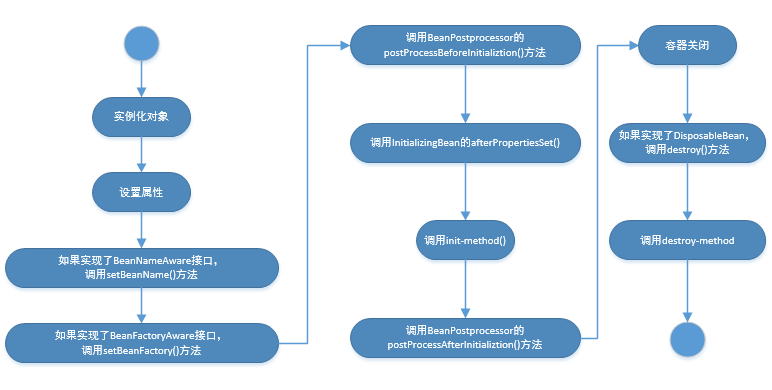
第二条设计主线是：以ApplicationContext应用上下文为核心的设计。涉及的主要接口类有BeanFactory🡪ListableBeanFactory🡪ApplicationContext🡪 ConfigurableApplicationContext(WebApplicationContext)，ApplicationContext继承MessageSource(支持信息源，实现国际化)，ResourcePatternResolver(访问资源)，ApplicationEventPublisher(支持响应时间)

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**

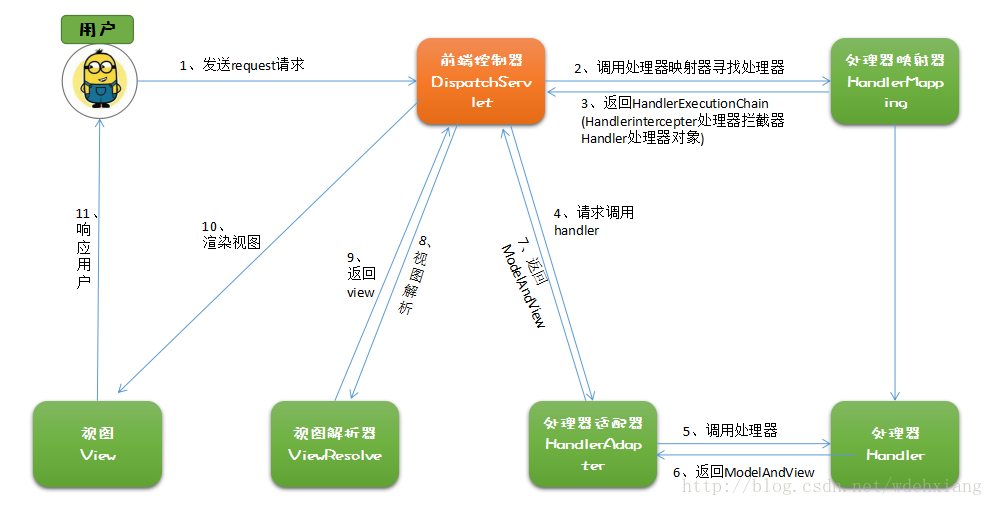
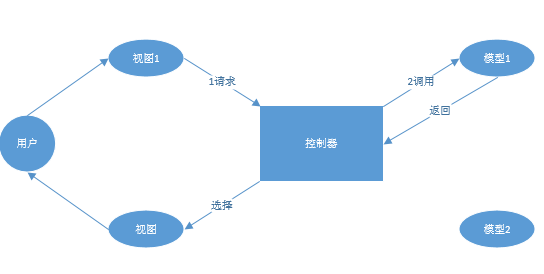
MVC(Model-View-Controller)：即把一个应用的输入、处理和输出流程按照Model、Controller、View的方式进行分离，将应用分为模型层，控制层和视图层的三层结构。

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。

Mybatis缓存：

Mybatis提供了两级缓存，缓存机制减轻数据局压力，提高数据库性能。

一级缓存为SQLSession级别的缓存，mybatis默认开启一级缓存，不需要配置。sqlsession执行了DML(insert,update,delete)并commit，mybatis会清除当前是sqlsession所有的一级缓存，sqlsession结束后，当前的sqlsession也就不存在；

二级缓存为mapper级别的缓存，同一个namespace公用的一个缓存，即同一个namespace的mapper.xml，当多个sqlsesion使用同一个mapper操作数据库时，得到的数据会缓存在同一个二级缓存区域，二级缓存默认false,，可以在setting中配置二级缓存并设置相关属性配置。一个sqlsession执行了select之后commit并关闭session后，会将查询结果缓存在二级区域，如果另一个sqlsession执行select时，会现在当前的sqlsession的一级缓存中查找，如果没有，则去二级缓存查找。Sqlsession执行了DML操作并commit后，mybatis会清除当前mapper的所有缓存。Mybatis的二级缓存在多表查询时，极大有可能会出现脏数据，可以配置cache-ref解决。

**7、Hibernate框架**

Hibernate是一个对象关系映射框架(ORM)，对jdbc进行了轻量的封装，将POJO(Plain Old Java Object)与数据库表建立映射关系，可以自动生成sql语句，自动执行。

核心API：Session、SessionFactory、Transaction、Query、Criteria和Configuration。

1、Session接口

Session接口负责执行被持久化对象的CRUD操作

2、SessionFactory接口

SessionFactory接口负责初始化hibernate，充当数据源的代理，并创建session对象

3、Transaction接口

可选API，hibernate的设计者自己写的底层事务处理代码可以取代这个接口

4、Query接口

Query接口对数据库及持久化对象进行查询，表达方式有HQL语言或者本地数据库的SQL语句

5、Criteria接口

Criteria接口和Query接口相似，允许创建并执行面向对象的标准化查询

6、Configuration

Configuration类的作用是对hibernate进行配置以及如何进行启动

Hibernate缓存：

一级缓存：session级别的缓存，即一级缓存的声明周期和session的声明周期一致，session关闭，一级缓存失效。一级缓存不可配置，可以使用evict(将某个对象从session的一级缓存中清除)和clear(将一级缓存中的所有对象全部清除)两个方法管理。TIPS：query.list()每次查询都从数据库中取出所有数据，不使用一级缓存；query.iterator()先通过一条语句取出所有数据，然后通过id在hibernate的一级缓存中查找该对象，如果存在则取出，如果没有则从数据去取出，并存入一级缓存（如果使用iterator，若数据都不存在一级缓存，则可能发生n+1条SQL语句的问题）

二级缓存：二级缓存也称SessionFactory缓存，可以被所有的session共享，在hibernate中可配置，支持第三方缓存。

Hibernate和Mybatis的比较

1. 开发速度的对比：hibernate中SQL语句被封装，可以直接使用，加快开发速度；Mybatis属于半自动化，SQL语句需要手动完成，复杂语句较多
2. Sql语句优化：hibernate自动生成SQL，繁琐语句会多消耗性能；Mybatis手动编写，可以避免不必要的查询，提高系统性能
3. 对象的管理：hibernate是完整的对象/映射解决方案，开发中不需要过多的关注底层实现，更多的管理对象；mybatis需要自己管理映射关系
4. 缓存：相同点：两者的二级缓存除了采用系统默认的缓存机制外，都可以通过实现自己的缓存或者其他第三方缓存方案；不同点：hibernate的二级缓存在SessionFactory生成的配置文件中进行详细配置，然后在具体的表-对象映射中配置那种缓存；mybatis的二级缓存在每个具体的表-对象映射中具体配置，这样可以针对不同的表配置不同的缓存方式。

**8、java多线程与并发**

线程：是操作系统能够进行运算调度的最小单位，它被包含在进程之中，是进程中实际运行的单位。

实现线程的方法：

1. 继承java.lang.Thread类
2. 实现java.lang.Runable接口重写run()方法

创建多线程时，由于java不支持多继承，但是允许调用多个接口，如果实现类要继承其他接口，可以通过调用Runnable接口实现多线程效果更好

Thread中的start()和run()方法的区别：

start()方法用来创建新的线程，java虚拟机调用run()方法，这样调用start()方法后，无需等待run()方法可以继续向下执行，当run()方法获取到cpu时间片开始执行run()方法；当直接调用run()方法时，run()方法会被看做thread类的一个普通方法，只会在原来的线程中调用，并没有创建新的线程。

sleep()、wait()、yeild()方法的区别：

sleep()会让线程由运行状态🡪等待(阻塞)状态，但是不会释放对象锁，sleep(long mills)一段时间后，该线程重新执行

wait()让线程由运行状态🡪等待(阻塞)状态，线程释放对象锁，其他线程执行，如果要唤起该想成，调用notify()或者notrifyAll()方法对线程进行唤起

yield()让线程由运行状态🡪就绪状态，调用yeild()方法后，线程会让出对象锁给相同级别的线程，其他线程获得对象锁，也有可能还是该线程获取对象锁