* 几点需要注意：
* 尽量使用缓存，包括用户缓存，信息缓存等，多花点内存来做缓存，可以大量减少与数据库的交互，提高性能。
* 用jprofiler等工具找出性能瓶颈，减少额外的开销。
* 优化数据库查询语句，减少直接使用hibernate等工具的直接生成语句（仅耗时较长的查询做优化）。
* 优化[数据库结构](https://www.baidu.com/s?wd=%E6%95%B0%E6%8D%AE%E5%BA%93%E7%BB%93%E6%9E%84&tn=44039180_cpr&fenlei=mv6quAkxTZn0IZRqIHckPjm4nH00T1Y3mWD1ujKBmHTsm1ubn1nv0ZwV5Hcvrjm3rH6sPfKWUMw85HfYnjn4nH6sgvPsT6KdThsqpZwYTjCEQLGCpyw9Uz4Bmy-bIi4WUvYETgN-TLwGUv3EnHn3nHD4rjbLnj0sn1TdnWRsn0" \t "https://zhidao.baidu.com/question/_blank)，多做索引，提高查询效率。
* 统计的功能尽量做缓存，或按每天一统计或定时统计相关报表，避免需要时进行统计的功能。
* 能使用[静态页面](https://www.baidu.com/s?wd=%E9%9D%99%E6%80%81%E9%A1%B5%E9%9D%A2&tn=44039180_cpr&fenlei=mv6quAkxTZn0IZRqIHckPjm4nH00T1Y3mWD1ujKBmHTsm1ubn1nv0ZwV5Hcvrjm3rH6sPfKWUMw85HfYnjn4nH6sgvPsT6KdThsqpZwYTjCEQLGCpyw9Uz4Bmy-bIi4WUvYETgN-TLwGUv3EnHn3nHD4rjbLnj0sn1TdnWRsn0" \t "https://zhidao.baidu.com/question/_blank)的地方尽量使用，减少容器的解析（尽量将动态内容生成静态html来显示）。
* 解决以上问题后，使用[服务器集群](https://www.baidu.com/s?wd=%E6%9C%8D%E5%8A%A1%E5%99%A8%E9%9B%86%E7%BE%A4&tn=44039180_cpr&fenlei=mv6quAkxTZn0IZRqIHckPjm4nH00T1Y3mWD1ujKBmHTsm1ubn1nv0ZwV5Hcvrjm3rH6sPfKWUMw85HfYnjn4nH6sgvPsT6KdThsqpZwYTjCEQLGCpyw9Uz4Bmy-bIi4WUvYETgN-TLwGUv3EnHn3nHD4rjbLnj0sn1TdnWRsn0" \t "https://zhidao.baidu.com/question/_blank)来解决单台的瓶颈问题。

生成代理类：

* AopProxyFactory根据构造方法的入参AdvisedSupport来决定JDK代理还是Cglib。如果目标类Class是接口，则通过JdkDynamicAopProxy使用JDK动态代理，否则使用ObjenesisCglibAopProxy来生成代理。
* 实例化JdkDynamicAopProxy(AdvisedSupport config) ，调用getProxy方法，即Proxy.newProxyInstance方法，其中接口除了target实现的接口外，还包括SpringProxy、Advised(opaque=false) ；切面为JdkDynamicAopProxy自身。

调用代理类：

* 代理类会调用切面JdkDynamicAopProxy的invoke方法，若Method为Advised接口或者其父接口中定义的方法，直接反射调用，不应用通知。
* 从AdvisedSupport 的缓存Map中取Interceptor Chain通知链，若没有，则通过DefaultAdvisorChainFactory去获取，并将结果放入缓存。
* **DefaultAdvisorChainFactory去获取：**

1. 实例化DefaultAdvisorAdapterRegistry，即实例化并注册MethodBeforeAdviceAdapter、AfterReturningAdviceAdapter、ThrowsAdviceAdapter（AdvisorAdapter接口）；
2. 遍历AdvisedSupport 的所有Advisor，若是PointcutAdvisor，则通过DefaultAdvisorAdapterRegistry的一系列AdvisorAdapter来生成MethodInterceptor[]，若不是动态方法匹配，则直接加入到interceptorList；若是动态方法匹配，则实例化InterceptorAndDynamicMethodMatcher加入到interceptorList。若是IntroductionAdvisor，则通过DefaultAdvisorAdapterRegistry的一系列AdvisorAdapter来生成MethodInterceptor[]加入到interceptorList。
3. 返回interceptorList。

* 若Interceptor Chain为空，则直接反射调用 method.invoke(target, args)
* 若不为空，则实例化ReflectiveMethodInvocation，调用其proceed方法。**详见ReflectiveMethodInvocation**

# Q&A

* 什么情况下会发生JVm栈内存溢出?
* 数组申明太大，递归太深等等都有可能
* 类加载流程？
* **1. 类加载：Bootstrap Loader -> ExtClassLoader, Bootstrap Loader -> AppClassLoader   
  2. 静态代码块初始化   
  3. 链接  
   a) 验证：是否符合java规范   
   b) 准备：默认初始值 （如boolean为false，int为0等）  
   c) 解析：符号引用转为直接引用，解析地址   
  4. 初始化   
   a) 赋值：类中属性的初始值   
   b) 构造：构造函数**
* JDK动态代理与cglib实现的区别
* 生成代理类（有反射Method组成），目标类必须实现接口，ASM框架修改字节码，生成目标类的子类，方法不能final。
* Spring如何管理事务的
* 通过“声明式事务”的方式（注解或xml配置）对事务进行管理，通过AOP将事务切面织入程序，大大减少了代码量，比如：

|  |  |
| --- | --- |
| 1  2  3  4 | Transaction tx=session.getTransaction();  session.beginTransaction();  tx.commit();  tx.rollback(); |

* spring的controller是单例还是多例，怎么保证并发的安全
* 单例，

1. 在controller中不使用实例变量  
    2、将controller上加@Scope("prototype")，每次都创建新的controller  
    3、在Controller中使用ThreadLocal变量

* tomcat结构

<Server>          //顶层类元素，可以包括多个Service

    <Service>  //可包含一个Engine，多个Connecter

        <Connector />          //连接器类元素，代表通信接口

        <Engine>                 //容器类元素，为特定的Service组件处理客户请求，要包含Host

<Host>                 //容器类元素，为特定的虚拟主机组件处理客户请求，处理web.xml

</Host>

        </Engine>

    </Service>

</Server>

* 当出现了内存溢出，你怎么排错
* MAT工具