项目名称：_____ 模块名称：_____ 版本号：_____

检查时间：_____ 检查员：_____

#	检查项	是/否	注释
1.	命名、注释及风格		
	1.1 文件、类 /接口、静态变量、成员变量、方法及关键代码是否都有格式良好、 简明扼要、 的注释？注释是否是对设计思路的说明而不仅仅是代码行为的描述？是否存在过时的注释或废代码注释？		
	1.2 文件中各种段落布局是否合理、 是否用恰当的空行分隔？代码的断行、对齐、缩进、空行是否恰当？		
	1.3 代码中的变量、 属性、 参数、 方法、 类是否恰当命名？有无易相互混淆的命名？		
	1.4 其他编码风格是否符合规范要求？如 if、else 永远加{}；for 、while 的格式； switch 永远有 default ；case 有 break 等等		
2.	设计原则		
	2.1 成员变量和方法 public/private/protected 关键字是否恰当？内部类、 非公共类及其内部成员、 方法的访问权限是否合适？		

2.2 嵌套内部类是否超过 2 层？		
2.3 所有方法是抽象的且所有成员变量是静态常量的抽象类是否声明成了接口？		
2.4 当类所有的方法和属性都是静态的时，是否定义了缺省的私有构造方法？		
2.5 没有使用任何实例类成员（包括方法和成员变量）的方法是否被声明为静态的？		
2.6 异常发生时是否均恰当的记录了错误日志？是否存在使用 System.out.println 而不是日志模块记录日志的情况发生？		
2.7 是否有应定义为常量的数字、字符、字符串等存在？		
2.8 参数、变量等的类型是否定义的合适？精度是否足够？声明为超类的子类实例是否恰当？方法的返回值是否定义恰当？		
2.9 使用已有设计模式时，该模式要求的技术细节是否实现正确、完整？		
2.9.1 单态模式类的构造函数是否声明为了私有？Lazy 类型的单态模式是否使用了同步？		
2.9.2 观察者模式 attach 与 detach 是否匹配？		
2.10 是否存在废代码？		
2.10.1 是否存在没有使用的参数、变量、对象实例？		

	2.10.2 是否存在重复、 无效的方法、 语句或子条件表达式？			
	2.10.3 子类或数据处理下游代码中是否重复设计了父类或数据处理上游代码中已有的功能？			
	2.10.4 调试用的代码是否恰当的封装在了 if (log.isDebugEnable) 中？			
	2.10.5 是否存在不必要的 import 句？			
3.	缺陷检查			
	3.1 程序逻辑是否符合详细设计？			
	3.2 输入参数、 调用其他函数的返回值是否有必要校验合法性、完整性、依赖关系？如有必要，是否做了恰当的校验？如无必要，是否有多余的代码？			
	3.3 是否避免了在抽象类构造方法中调用抽象方法？			
	3.4 是否避免了直接抛出 Exception 类异常， 而没有抛出恰当的由 Exception 派生的异常类？			
	3.5 try catch 的结构是否合理？ catch 语句处理是否恰当？异常转抛时是否携带了嵌套异常？			
	3.6 equals() 方法处理中是否使用了 getClass() 方法检查类型相同？			

3.7 打开的流、 连接等资源是否在 finally 语句块或恰当的地方关闭或释放了？临时资源使用完后是否及时释放了？如临时文件要及时删除。 各种资源释放的顺序是否正确？		
3.8 克隆方法中是否调用了父克隆方法？克隆方法中是否避免了调用构造函数？		
3.9 使用 ObjectStream 后是否调用了 reset() 方法以避免内存泄漏？		
3.10 条件、循环中的判断边界值是否恰当？		
3.11 程序块的 break 、 return 、 throw 是否恰当？		
3.12 charAt() 、数组下标、 parseInt 之类可能抛运行时异常的方法是否需要事先判断或事后 catch ？如需要，处理是否恰当？		
3.13 相似的代码块是否是拷贝过来的？如果是，则需检查拷贝的代码中每处需要修改的地方是否都修改了。		
3.14 是否存在其他可能造成缺陷的代码？		
3.14.1 在 if 条件中赋值		
3.14.2 精度丢失		
3.14.3 浮点数判等		
3.14.4 循环体中修改循环变量		
3.14.5 case 语句缺失 break		

	3.14.6 字符串比较没有使用 equals 或 compareTo		
	3.14.7 不恰当的 static 变量		
	代码优化		
	4.1 同步方法的使用是否必要？同步代码块是否已粒度最小化？		
	4.2 在不影响可读性和易维护性的前提下，对象是否可重复利用？如 StringBuffer 可以通过 setLength(0) 重复利用，无需每次重复创建新实例。		
	4.3 是否有这样的代码： new String(”)？		
	4.4 是否有循环拷贝数组而没有使用 System.arrayCopy() 的代码？		
4.	4.5 基于性能考虑，在可能的情况下，StringBuffer 、ArrayList 、HashMap 、HashSet 、Hashtable 、Vector 、WeakHashMap 等结构实例化时是否指定了初始大小？		
	4.6 嵌套的条件判断、循环是否可优化？		
	4.6.1 嵌套的条件判断、循环是否可替换为较简单的结构？		
	4.6.2 是否存在可以合并的循环或条件判断？		
	4.6.3 循环体内是否存在可以提到循环体外的代码？如实例化对象或声明变量等		

	4.7 switch 中 case 的常量是否可以安排为连续的值？这样做可以使得代码被编译为 tableswitch 类型的跳转表提高性能。连续的 else if 结构是否按情况出现的频率进行了排序？		
	4.8 循环条件中或重复出现的如.length() .size() .getClass().getName() 等取值方法的返回值是否使用了临时变量代替？		
	4.9 频繁拼装字符串时，是否使用了 StringBuffer ，并事先分配恰当空间？		
	目录结构		
	5.1 Jsp,Action ， service ， dao , 逻辑分层是否明确		
	5.2 配置文件（ struts ， spring ， sql）是否条理清晰，目录规范		
5.	5.3 页面资源文件（ JavaScript,CSS,images ）目录是否规范		