【强制】异常信息应该包括两类信息：案发现场信息和异常堆栈信息。如果不处理，那么往上抛。 正例：logger.error(各类参数或者对象toString + "\_" + e.getMessage(), e);

7. 输出的POJO类必须重写toString方法，否则只输出此对象的hashCode值（地址值），没啥参考意义

【推荐】在代码中使用“抛异常”还是“返回错误码”，对于公司外的http/api开放接口必须使用“错误码”；而应用内部推荐异常抛出；跨应用间RPC调用优先考虑使用Result方式，封装isSuccess、“错误码”、“错误简短信息”。 说明：关于RPC方法返回方式使用Result方式的理由： 1）使用抛异常返回方式，调用方如果没有捕获到就会产生运行时错误。 2）如果不加栈信息，只是new自定义异常，加入自己的理解的error message，对于调用端解决问题的帮助不会太多。如果加了栈信息，在频繁调用出错的情况下，数据序列化和传输的性能损耗也是问题。

【推荐】定义时区分unchecked / checked 异常，避免直接使用RuntimeException抛出，更不允许抛出Exception或者Throwable，应使用有业务含义的自定义异常。推荐业界已定义过的自定义异常，如：DaoException / ServiceException等。

异常不要用来做流程控制，条件控制，因为异常的处理效率比条件分支低。

SimpleDateFormat 是线程不安全的类，一般不要定义为static变量，如果定义为static，必须加锁，或者使用DateUtils工具类。 正例：注意线程安全，使用DateUtils。亦推荐如下处理： private static final ThreadLocal<DateFormat> df = new ThreadLocal<DateFormat>() { @Override protected DateFormat initialValue() { return new SimpleDateFormat("yyyy-MM-dd"); } }; 说明：如果是JDK8的应用，可以使用instant代替Date，Localdatetime代替Calendar，Datetimeformatter代替Simpledateformatter，官方给出的解释：simple beautiful strong immutable thread-safe。

【推荐】类成员与方法访问控制从严：

 1） 如果不允许外部直接通过new来创建对象，那么构造方法必须是private。  2） 工具类不允许有public或default构造方法。

 3） 类非static成员变量并且与子类共享，必须是protected。   4） 类非static成员变量并且仅在本类使用，必须是private。  5） 类static成员变量如果仅在本类使用，必须是private。  6） 若是static成员变量，必须考虑是否为final。  7） 类成员方法只供类内部调用，必须是private。   8） 类成员方法只对继承类公开，那么限制为protected。

说明：任何类、方法、参数、变量，严控访问范围。过宽泛的访问范围，不利于模块解耦。思考：如果是一个private的方法，想删除就删除，可是一个public的Service方法，或者一个public的成员变量，删除一下，不得手心冒点汗吗？变量像自己的小孩，尽量在自己的视线内，变量作用域太大，如果无限制的到处跑，那么你会担心的。

所有的相同类型的包装类对象之间值的比较，全部使用equals方法比较。

说明：对于Integer var=?在-128至127之间的赋值，Integer对象是在IntegerCache.cache产生，会复用已有对象，这个区间内的Integer值可以直接使用==进行判断，但是这个区间之外的所有数据，都会在堆上产生，并不会复用已有对象，这是一个大坑，推荐使用equals方法进行判断

@Inherited 如果自定义的解释组合了此注解，如以下自定义的注解：

@Target(ElementType.TYPE)

@Retention(RetentionPolicy.RUNTIME)

@Inherited

public @interface ABC{}

自定义了一个名为ABC的注解，如果放在放在一个名为A的类上，那么如果A有B,C两个子类，那么B,C这2个子类同样被自定加上了ABC这个注解。

可以在一个类上重复多个注解，以前是用{}嵌套的。

以前

@Schedules({@Schedule(...),@Schedule(...)})

现在

@Schedule(dayOfMonth="last")
@Schedule(dayOfWeek="Fri", hour="23")
public void doPeriodicCleanup() { ... }

步骤：

1.声明可重复的注解

import java.lang.annotation.Repeatable;
**@Repeatable(Schedules.class)**
public @interface Schedule {
String dayOfMonth() default "first";
String dayOfWeek() default "Mon";
int hour() default 12;
}

2.声明包装

public @interface Schedules {
Schedule[] value();
}

java8为ElementType枚举增加了TYPE\_PARAMETER,TYPE\_USE两个枚举类，这样就允许 定义枚举时使用@Target(ElementType.TYPE\_USE)修饰，这种注解被称为Type Annotation，Type Annotaion可用在任何用到类型的地方。可用在：创建对象（用new关键字创建）,类型转换，使用implement实现接口，使用throws声明抛出异常。

<http://www.mscharhag.com/java/java-8-type-annotations>

编译时处理Annotation (APT工具Annotaion Processing Tool)

Hibernate 里将包含注解的对象转换成hbm.xml就是通过这个工具.java提供的javac.exe 工具有一个-processor选项，该选项可指定一个Annotaion 处理器，如果在编译java源文件时通过该选项指定了Annotaion处理器，那么这个Annotaion处理器将会在编译时提取并处理java源文件中的Annotaion.

每个Annotaion处理器都需要实现javax.annotaion.processing包下的Processor接口。不过该实现接口必须实现它里面所有的方法，因此通常会采用继承AbstractProcessor的方式来实现Annotaion处理器，一个Annotaion处理器可以处理一种或多种Annotation类型。

@SupportedSourceVersion(SourceVersion.RELEASE\_8)

@SupportedAnnotaionTypes({"id",...."注解类型"});

public class HibernatAnnotationProcessor extends AbstractProcessor{...}

java8中增加了Arrays工具类

内部类：<http://www.cnblogs.com/chenssy/p/3388487.html>

内部类可以解决多重继承问题，外类继承一个类，内部类再继承另一个类来解决多重继承的问题。

Java NIO.2 提供了Files,Paths两个工具类。Paths提供了get(String first,string ...more)方法来获取Path对象，Paths会将给定的从个字符串连缀成路径。

FileVisitor遍历文件和目录。

Files.walkFileTree(Path start, Filevisitor<? super Path> visitor)遍历start路径下的所有文件和子目录 。

WatchService监控文件变化。NIO２的Path类提供了如下一个方法来监听文件系统的变化。

register(WatchService watcher, Watch?Event.Kind<?> ... events);用watcher监听该path代表的目录下的文件变化。events参数指定要监听哪些类型的事件。在这个方法中WatchService代表一个文件系统监听服务，它负责监听path代表的目录下的文件变化。一旦使用register()方法完成注册之后，接下来就可调用watchservice的如下三个方法来获取被监听目录的文件变化事件。

WatchKey poll()获取下一个WatchKey,如果没有WatchKey发生就立即返回null.

WatchKey poll(long timeout, TimeUnit unit)尝试等待timeout时间去获取下一个WatchKey.

WatchKey take()获取下一个WatchKey，如果没有WatchKey发生就一直等待。

访问文件属性的包都在java.nio.file.attribute包下面。

TypeSafe 可以读取多个配置文件，也可以用spring中的 PropertiesLoader去加载

【强制】所有的相同类型的包装类对象之间值的比较，全部使用equals方法比较。 说明：对于Integer var = ? 在-128至127范围内的赋值，Integer对象是在IntegerCache.cache产生，会复用已有对象，这个区间内的Integer值可以直接使用==进行判断，但是这个区间之外的所有数据，都会在堆上产生，并不会复用已有对象，这是一个大坑，推荐使用equals方法进行判断

关于基本数据类型与包装数据类型的使用标准如下： 1） 【强制】所有的POJO类属性必须使用包装数据类型。 2） 【强制】RPC方法的返回值和参数必须使用包装数据类型。 3） 【推荐】所有的局部变量使用基本数据类型。 说明：POJO类属性没有初值是提醒使用者在需要使用时，必须自己显式地进行赋值，任何NPE问题，或者入库检查，都由使用者来保证。

【推荐】 类内方法定义顺序依次是：公有方法或保护方法 > 私有方法 > getter/setter方法。 说明：公有方法是类的调用者和维护者最关心的方法，首屏展示最好；保护方法虽然只是子类关心，也可能是“模板设计模式”下的核心方法；而私有方法外部一般不需要特别关心，是一个黑盒实现；因为方法信息价值较低，所有Service和DAO的getter/setter方法放在类体最后。

【推荐】final可以声明类、成员变量、方法、以及本地变量，下列情况使用final关键字： 1） 不允许被继承的类，如：String类。 2） 不允许修改引用的域对象，如：POJO类的域变量。 3） 不允许被重写的方法，如：POJO类的setter方法。 4） 不允许运行过程中重新赋值的局部变量。 5） 避免上下文重复使用一个变量，使用final描述可以强制重新定义一个变量，方便更好地进行重

【推荐】类成员与方法访问控制从严： 1） 如果不允许外部直接通过new来创建对象，那么构造方法必须是private。 2） 工具类不允许有public或default构造方法。 3） 类非static成员变量并且与子类共享，必须是protected。 4） 类非static成员变量并且仅在本类使用，必须是private。 5） 类static成员变量如果仅在本类使用，必须是private。 6） 若是static成员变量，必须考虑是否为final。 7） 类成员方法只供类内部调用，必须是private。 8） 类成员方法只对继承类公开，那么限制为protected。 说明：任何类、方法、参数、变量，严控访问范围。过于宽泛的访问范围，不利于模块解耦。思考：如果是一个private的方法，想删除就删除，可是一个public的service方法，或者一个public的成员变量，删除一下，不得手心冒点汗吗？变量像自己的小孩，尽量在自己的视线内，变量作用域太大，如果无限制的到处跑，那么你会担心的。

不要在foreach循环里进行元素的remove/add操作。remove元素请使用Iterator方式，如果并发操作，需要对Iterator对象加锁。

【推荐】集合初始化时，指定集合初始值大小。 说明：HashMap使用HashMap(int initialCapacity) 初始化， 正例：initialCapacity = (需要存储的元素个数 / 负载因子) + 1。注意 负载因子（即loader factor）默认 为 0.75，如果 暂时 无法确定 初始 值大小，请设置 为 16。 反例： HashMap需要 放置 1024个元素 ，由于 没有设置容量 初始大小，随着元素不断增加容 量 7次被迫扩大， resize需要重建 hash表，严重影响性能。

【参考】合理利用好集合的有序性(sort)和稳定性(order)，避免集合的无序性(unsort)和不稳定性(unorder)带来的负面影响。 说明：有序性是指遍历的结果是按某种比较规则依次排列的。稳定性指集合每次遍历的元素次序是一定的。如：ArrayList是order/unsort；HashMap是unorder/unsort；TreeSet是order/sort。



【强制】SimpleDateFormat 是线程不安全的类，一般不要定义为static变量，如果定义为static，必须加锁，或者使用DateUtils工具类。 正例：注意线程安全，使用DateUtils。亦推荐如下处理：

private static final ThreadLocal<DateFormat> df = new ThreadLocal<DateFormat>() {

@Override

protected DateFormat initialValue() {

return new SimpleDateFormat("yyyy-MM-dd");

}

};

说明：如果是JDK8的应用，可以使用Instant代替Date，LocalDateTime代替Calendar，DateTimeFormatter代替Simpledateformatter，官方给出的解释：simple beautiful strong immutable thread-safe。

【强制】高并发时，同步调用应该去考量锁的性能损耗。能用无锁数据结构，就不要用锁；能锁区块，就不要锁整个方法体；能用对象锁，就不要用类锁。 说明：尽可能使加锁的代码块工作量尽可能的小，避免在锁代码块中调用RPC方法。

【强制】并发修改同一记录时，避免更新丢失，需要加锁。要么在应用层加锁，要么在缓存加锁，要么在数据库层使用乐观锁，使用version作为更新依据。 说明：如果每次访问冲突概率小于20%，推荐使用乐观锁，否则使用悲观锁。乐观锁的重试次数不得小于3次。

【参考】volatile解决多线程内存不可见问题。对于一写多读，是可以解决变量同步问题，但是如果多写，同样无法解决线程安全问题。如果是count++操作，使用如下类实现：AtomicInteger count = new AtomicInteger(); count.addAndGet(1); 如果是JDK8，推荐使用LongAdder对象，比AtomicLong性能更好（减少乐观锁的重试次数）。

【参考】ThreadLocal无法解决共享对象的更新问题，ThreadLocal对象建议使用static修饰。这个变量是针对一个线程内所有操作共有的，所以设置为静态变量，所有此类实例共享此静态变量 ，也就是说在类第一次被使用时装载，只分配一块存储空间，所有此类的对象(只要是这个线程内定义的)都可以操控这个变量。

【参考】下列情形，需要进行参数校验： 1） 调用频次低的方法。 2） 执行时间开销很大的方法。此情形中，参数校验时间几乎可以忽略不计，但如果因为参数错误导致中间执行回退，或者错误，那得不偿失。 3） 需要极高稳定性和可用性的方法。 4） 对外提供的开放接口，不管是RPC/API/HTTP接口。

5） 敏感权限入口。

【强制】后台输送给页面的变量必须加$!{var}——中间的感叹号。 说明：如果var=null或者不存在，那么${var}会直接显示在页面上。

【推荐】任何数据结构的构造或初始化，都应指定大小，避免数据结构无限增长吃光内存。

MYSQL:

【强制】表达是与否概念的字段，必须使用is\_xxx的方式命名，数据类型是unsigned tinyint（ 1表示是，0表示否）。 说明：任何字段如果为非负数，必须是unsigned。 正例： 表达逻辑删除的字段名 is\_deleted，1表示删除， 0表示未删除。 表

【强制】表必备三字段：id, gmt\_create, gmt\_modified。 说明：其中id必为主键，类型为unsigned bigint、单表时自增、步长为1。gmt\_create, gmt\_modified的类型均为date\_time类型。

【推荐】表的命名最好是加上“业务名称\_表的作用”。 正例：tiger\_task / tiger\_reader / mpp\_config

【推荐】字段允许适当冗余，以提高查询性能，但必须考虑数据一致。冗余字段应遵循： 1）不是频繁修改的字段。 2）不是varchar超长字段，更不能是text字段。 正例：商品类目名称使用频率高，字段长度短，名称基本一成不变，可在相关联的表中冗余存储类目名称，避免关联查询。

【推荐】单表行数超过500万行或者单表容量超过2GB，才推荐进行分库分表。 说明：如果预计三年后的数据量根本达不到这个级别，请不要在创建表时就分库分表。

