**网管四部代码走查’一页纸’规范**

**1.使用外部数据前是否进行了空判断和数组/容器的长度判断**

反例;

public NetworkInfo queryNetWorkInfo() {

DbResource dbResource = new DbResource();

String[] queryFields = {"mcc", "mnc"};

Map<String, Object> queryData =

dbResource.querySingle("mocName", "moi", "where", queryFields);

Object mcc = queryData.get("mcc"); //bad, mcc may be not in 'queryData'

Object mnc = queryData.get("mnc");

return new NetworkInfo(mcc.toString(), mnc.toString());

}

public void getMeasureParameter() {

DbResource dbResource = new DbResource();

String[] queryFields = {"measureParameterName"};

List<Map<String, Object>> queryData =

dbResource.query("EUtranCell", "moi", "where", queryFields);

//bad, 'queryData' may be null or size < 6

Map<String, Object> targetMeasure = queryData.get(6);

}

正例：

public static final int CELL\_MEASURE\_PARA\_LENGTH = 6;

public NetworkInfo queryNetWorkInfo() {

DbResource dbResource = new DbResource();

String[] queryFields = {"mcc", "mnc"};

Map<String, Object> queryData =

dbResource.querySingle("mocName", "moi", "where", queryFields);

Object mcc = queryData.get("mcc");

Object mnc = queryData.get("mnc");

if (mcc == null || mnc == null) {

//throw new RuntimeException();

return NetworkInfo.emptyNetworkInfo(); //strongly recommend

}

return new NetworkInfo(mcc.toString(), mnc.toString());

}

public void getMeasureParameter() {

DbResource dbResource = new DbResource();

String[] queryFields = {"measureParameterName"};

List<Map<String, Object>> queryData =

dbResource.query("EUtranCell", "moi", "where", queryFields);

if (queryData.size() >= CELL\_MEASURE\_PARA\_LENGTH) {

Map<String, Object> targetMeasure = queryData.get(CELL\_MEASURE\_PARA\_LENGTH);

//...other code

}

}

**2.代码中是否存在拼接SQL的问题，而没有使用PreparedStatement**

使用PreparedStatement的好处：

1.代码的可读性和可维护性  
2.PreparedStatement尽最大可能提高性能  
3.极大地提高了安全性  
**在JDBC应用中,如果你已经是稍有水平开发者,你就应该始终以PreparedStatement代替Statement.也就是说,在任何时候都不要使用Statement.**

**3.对于多线程代码，要确认访问代码是否需要做同步保护**

线程同步的“基础知识”要清楚(类锁，对象锁，锁分离)

**4.使用java.util.concurrent包中的类处理多线程问题，代替传统的wait()/notify()**

各种阻塞队列，Future接口，Lock/Condition

**5.涉及外部接口（变更）时，重点走查**

1.一定要团队走查（知识来源不一，概率性一致）  
2.有疑问一定要弄清楚，多沟通，不要猜测，外部接口有很多你不知道的“坑”,例如：平台的FTP接口  
3.一定要测试

**6.代码是否具备兼容性要求，EMS和OMMB之间，OMMB与NE之间，新旧版本之间**

1.BA确认需求时要明确  
2.启动会讲解故事卡时要强调，再故事卡中注明  
3.代码走查时要格外注意

**7.处理集合时，要根据业务场景明确是“事务”方式，还是“最大努力”方式**

(伪)事务方式：

public void transactionStyle(Collection<?> items) {

if(items == null || items.isEmpty()) {

return;

}

try {

for (Object item : items) {

processSingleItem(item);

}

} catch (Exception e) {

e.printStackTrace();

//rollback(items);

}

}

private void processSingleItem(Object item) throws Exception {

}

private void rollback(Object item) {

//rollback action

}

最大努力方式：

public void transactionStyle(Collection<?> items) {

if(items == null || items.isEmpty()) {

return;

}

for (Object item : items) {

try {

processSingleItem(item);

} catch (Exception e) {

e.printStackTrace();

}

}

}

private void processSingleItem(Object item) throws Exception {

}

private void rollback(Object item) {

//rollback action

}

**8.处理数量巨大或处理单个元素比较耗时的集合时，考虑引入多线程提高效率**

未使用多线程方式：

void process(Collection<?> items) {

if(items == null || items.isEmpty()) {

return;

}

for (Object item : items) {

try {

processSingleItem(item);

} catch (Exception e) {

e.printStackTrace();

}

}

}

private void processSingleItem(Object item) throws Exception {

Thread.sleep(1000L);

}

//failed!

@Test(timeout = 5000)

public void should\_process\_10\_items\_within\_5\_seconds() throws Exception {

CollectionProcessor processor = new CollectionProcessor();

List<String> transactionIDs = Arrays.asList(new String[10]);

Collections.fill(transactionIDs, "123456");

processor.process(transactionIDs);

}

使用多线程方式：

void processWithMultiThread(Collection<?> items) throws InterruptedException {

if(items == null || items.isEmpty()) {

return;

}

ExecutorService executorService = Executors.newCachedThreadPool();

final CountDownLatch latch = new CountDownLatch(items.size());

for (final Object item : items) {

executorService.submit(new Runnable() {

@Override

public void run() {

try {

processSingleItem(item);

} catch (Exception e) {

e.printStackTrace();

} finally {

latch.countDown();

}

}

});

}

latch.await(5, SECONDS);

executorService.shutdown();

}

private void processSingleItem(Object item) throws Exception {

Thread.sleep(1000L);

}

//Pass!

@Test(timeout = 5000)

public void should\_process\_10\_items\_within\_5\_seconds() throws Exception {

CollectionProcessor processor = new CollectionProcessor();

List<String> transactionIDs = Arrays.asList(new String[10]);

Collections.fill(transactionIDs, "123456");

processor.processWithMultiThread(transactionIDs);

}

欠下的债迟早要还，开发的时候不考虑性能，大容量测试倒逼的时候就得”痛苦“的考虑了。

**9.是否采用了规范的异常处理信息（正确抛出和捕获，未隐匿异常信息）**

private Logger logger = Logger.getLogger(this.getClass().getName());

public void hideExceptionInfo() {

try {

thatMethodThrowsException();

} catch (Exception e) {

;//terrible

logger.log(WARNING, "XXXXXXXXXXXXXXXXX"); //bad

logger.log(WARNING, "XXXXXXXXXXXXXXXXX", e); //right!

}

}

private void thatMethodThrowsException() throws Exception {

}

**10.是否在异常出现时进行了相关资源的释放**

**11.UT/RF是否对FDD/TDD测试用例覆盖完全**

**12.采用MessageFormat类格式化字符串，当传入的值是int或long类型时，需要将int或long转成String(或改用String.format)**

这段代码会打印什么？

private static final String NE\_MOI\_TEMPLATE = "SUBNETWORKID={0}, NEID={1}";

public static void main(String[] args) {

String neMoi = MessageFormat.format(NE\_MOI\_TEMPLATE, 211, 211);

System.out.println(neMoi); //output: SUBNETWORKID=211, NEID=211

}

换个NeID试试，

private static final String NE\_MOI\_TEMPLATE = "SUBNETWORKID={0}, NEID={1}";

public static void main(String[] args) {

String neMoi = MessageFormat.format(NE\_MOI\_TEMPLATE, 211, 3350);

System.out.println(neMoi); //output: SUBNETWORKID=211, NEID=3,350

}

**13.方法最好只有两层缩进**

**14.方法行数不超过30行**

**15.使用static Map/List存储数据时，是否会导致内存泄漏的发生**

案例：

1.OMMB在修改参数前，将<小区主键信息，待修改参数信息>放入一static Map的缓存  
2.将小区主键信息发送给EMS，待EMS确认后，EMS返回确认消息给OMMB  
3.OMMB收到确认消息，从缓存中获取修改参数下发  
4.删除该小区的缓存数据

解决方案：

1. 缓存用WeakHashMap
2. 缓存一定要加超时机制，考虑使用guava的cache
3. 是否一定要用Map

**16.数据库/JMS消息等耗资源操作，是否能够采用批量方式处理**

反例：

public class CellOperation {

public List<Map<String, Object>> queryCellData(List<String> cellMois,

String[] queryFields) {

if(cellMois == null || cellMois.isEmpty()) {

return Collections.emptyList();

}

List<Map<String, Object>> cellData = new ArrayList<Map<String, Object>>();

DbResource dbResource = new DbResource();

for (String singleCellMoi : cellMois) {

cellData.addAll(dbResource.query("CellMoc", singleCellMoi,

"", queryFields));

}

return cellData;

}

}

正例：

public List<Map<String, Object>> queryCellData(List<String> cellMois,

String[] queryFields) {

if(cellMois == null || cellMois.isEmpty()) {

return Collections.emptyList();

}

List<Map<String, Object>> cellData = new ArrayList<Map<String, Object>>();

DbResource dbResource = new DbResource();

cellData.addAll(

dbResource.query("CellMoc", "",buildQueryCellCondition(cellMois),queryFields));

return cellData;

}

private String buildQueryCellCondition(List<String> cellMois) {

return "MOI IN (" + Joiner.on(",").join(

Iterables.transform(cellMois, new Function<String, String>() {

@Override

public String apply(String singleCellMoi) {

return "'" + singleCellMoi + "'";

}

})) + ")";

}

generated by [haroopad](http://pad.haroopress.com)