## 总体设计

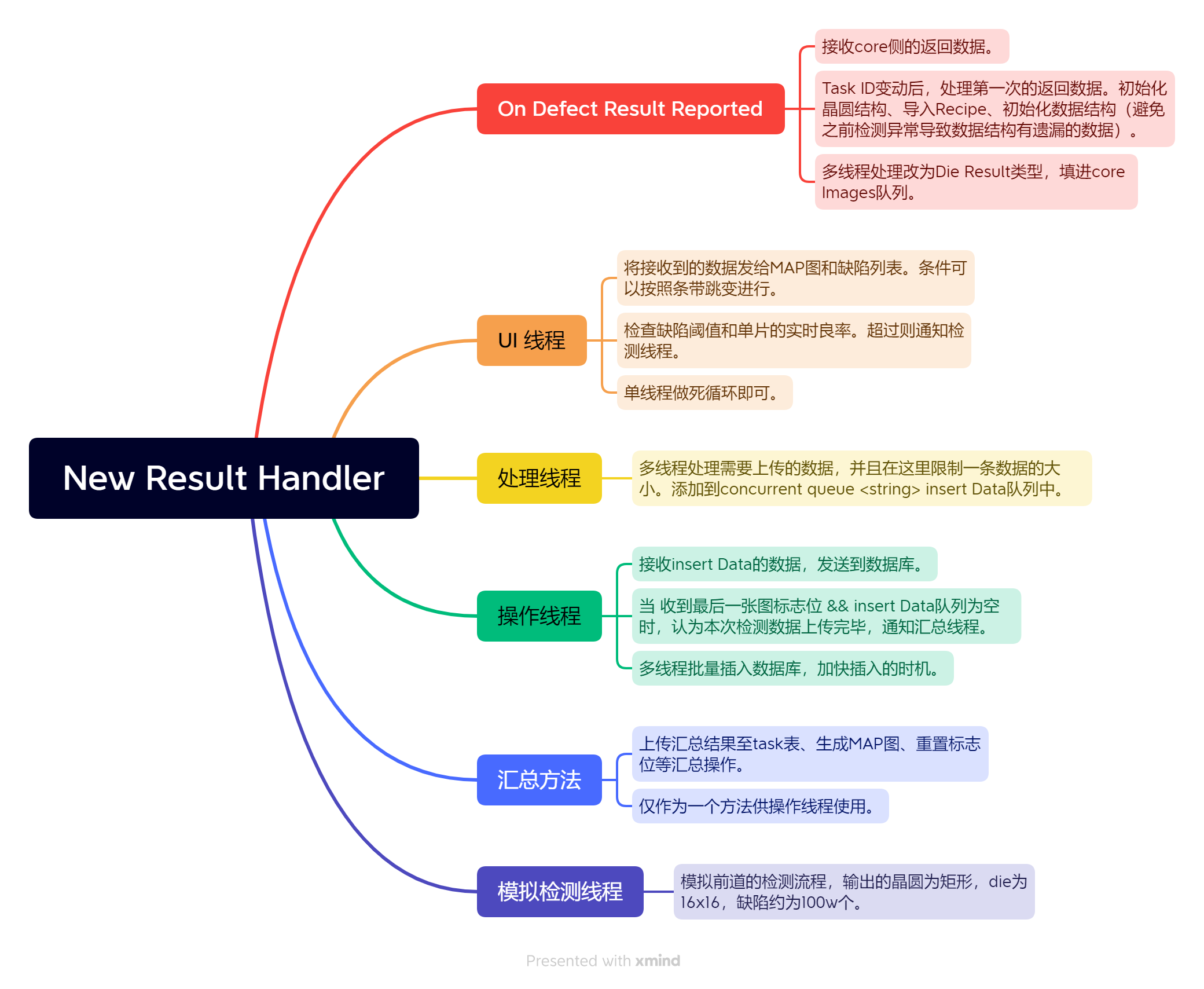
在Tornado V4结果处理设计的基础上，对代码结构进行了重构。之前的代码对于UI、数据库的更新混合在一起，没有区分开。这就会导致在处理大量数据时，如果有一边没有处理完，会耽误整个流程的时间。这个情况在处理前道、无图形检测尤其突出。

对于连续检测模式和生产模式，也没有做好兼容。之前的结果处理线程为单线程处理，其结构和之前V3的结果处理类似。当处于连续检测模式时，core返回的数据量很大，检测到下一片时可能还没有传完。此时可能会导致上一片的数据丢失。

汇总方法之前只会在完成一片的数据插入后自动汇总，不能满足因为手动停止、检测出错等情况导致中途停止时的情况，因此在新的结果线程中也提供了给外部调用的手动汇总方法。

在Tornado V4中，检测采图、core侧计算、结果处理完全独立运行，相互之间没有耦合的部分，只会因为一些设计好的事件来影响其它部分。

下图是新的结果线程重构设计图。



## **core侧返回数据方法**

沿用之前的设计，提供给core侧事件注册的形式来获取返回的数据。Core侧会以多线程的形式，List<DefectInfo>的数据返回给结果处理部分。因为结果处理和检测已经相互独立，所以需要core侧在返回数据时，附带上本次task的task ID来标明任务。

在第一次接收到新的task ID后，程序会用lock锁来锁住方法，防止多线程同步操作，进行一系列的初始化操作：

1. 启用Task等待上一片检测任务的数据全部插入完毕。
2. 重置标志位，初始化变量，防止数据遗漏。
3. 获取新的task ID到结果处理中，表明数据处理已经到了本次task中。
4. 根据新的task ID去数据库获取本次task的任务信息，包括recipe name等，获取本次task所使用的recipe。
5. 获取数据库中缺陷详细表当前的主键，作为本次task的min ID插入到task表中。
6. 初始化绘制缺陷详细开关，目前的逻辑为：无图形 或 有图形且程式中打开绘制缺陷详细开关，才会需要绘制缺陷。
7. 清空界面上的列表，包括die列表、缺陷列表和task列表。
8. 根据task ID获取task信息后，导入上一道工序的MAP文件，更新晶圆结构。
9. 获取缺陷图片的尺寸，并且转换为物理单位（mm）。
10. 从全局变量中获取当前的die尺寸和外围尺寸。
11. 获取本次task使用的镜头的X和Y方向的像元大小，并且将图片的像素转换为毫米。

在初始化完成后，就会关闭锁。允许多线程调用本方法，因为存储变量使用的是Concurrent Queue。

判断本次task是否已经调用了手动汇总，是则不会接收本次task的剩余数据。

判断晶圆结构是否已经初始化完毕，否则循环等待100ms，直到晶圆结构初始化完毕。

下面就是接收数据并且进行初步处理。

判断数据是否为最后一张图标志位，是则在界面上显示core侧返回最后一张图。

将传来的缺陷图片结果处理成适应新的数据库的实体类型，并且分别入队UI和DB的队列中。并且将缺陷图片结果加入到当前晶圆结构中。

至此就完成了对core侧返回数据的初步处理，后续就会交给UI线程和处理线程操作。

## UI线程

UI线程负责提供数据来绘制位于界面左下角的MAP图，以及界面右侧的缺陷列表。更新逻辑为条带发生跳变或UI队列数据已经全部取出。

因为绘制缺陷详细是根据具体的检测程式和片子类型决定的，而且两者绘制使用的数据不相同，所以将绘制MAP图上的die状态和缺陷位置函数分离。

绘制MAP图上的die状态：根据晶圆结构中的DieArray来判断。首先过滤未初始化的die和已经完成显示的die，这两种一个是不需要显示，另一个是已经完成显示不需要重复绘制。如果die的状态是被排除，添加exclude die的状态并且跳过剩下的流程。剩下的die就是需要绘制的die。判断是否发生条带跳变，die是否为检测die并且该die已经收到缺陷图片。如果是，那么就需要绘制该die的状态。对于前道die数据的显示，这里会有额外的处理：前道一个die由多个条带的小图组成，core侧一次返回的数据只会包含一个条带，并不是该die的全部。如果当前检测的条带已经比该die的列数要大，表示该die的数据已经收齐，可以置其完成标志位为真。如果还未收齐，那么对于暂时没有显示出缺陷的die，状态会有一个中间的过渡态。在其收齐后或者收到缺陷数据后置为对应的完成态。

判断当前实时数据的缺陷数量或实时良率是否超过程式设置的阈值，超过则会通知检测线程。一次task只会通知检测线程一次，不会重复进行通知。

绘制MAP图上的缺陷位置：缺陷详细数据从UI队列中直接取出，按照实际面积转换成对应的尺寸。这里有对前道数据的额外处理：前道一个die有多个小图组成，所以缺陷在一张图上的位置不代表就是该缺陷在die上的最终位置。需要先计算偏移后，加上坐标才能获取到其正确位置。

缺陷列表上的数据显示：UI线程会将UI数据队列以事件的形式，直接发送给结果列表界面，由结果界面进行处理。

结果界面的重构：将列表的数据源都改为了MVVM模式，而不是之前的直接插入数据。更新模式有两种：一是检测时更新，通过触发更新事件来更新当前的列表；二是未检测时，加载历史数据查看。

## 处理线程

这里会对DB队列中的数据进行处理，将其转换为真正插入数据库的字符串。主要是批量处理大量的数据，因此会用多线程处理数据。

MySQL对于一条插入语句的数据大小有限制，所以对于数据需要进行分割处理，保证一条插入语句的大小不会超过500k。首先判断一个die的缺陷数据是否已经超过限制，如果超过，那么需要将该die的缺陷数据进行分割后添加到数据队列中；如果未超过，那么就累积到字符串中，并且开始计算下个die的缺陷数据，累积超过限制后，添加到数据队列中。

## **操作线程**

操作线程会将处理好的数据取出后，完成插入数据库的操作。分离的原因是不混淆数据的处理和操作，两者互不干扰。

插入数据库会使用多线程进行插入，并且每次插入都会以Task的形式异步调用，Task会记录在一个list中。

当收到最后一张图标志位或手动要求汇总时，会去检查对当前的数据是否已经完成处理（手动要求汇总，会停止接收本次task的剩余数据）。然后会等待Task的list全部执行完毕，将汇总任务添加到list中，并且等待其执行完毕。最后把list初始化。

插入语句使用INSERT (COLUMN1, COLUMN2, ...) VALUES (FIELDS1) ... 的形式进行批量插入。

## **汇总方法**

该方法仅会在操作线程中被调用。当本次task的数据全部处理完毕或手动要求汇总时，会进行以下操作：

1. 获取本次task的bad die和ok die总数，缺陷总数，上传到task表中。更新task的状态，正常结束为finished，手动停止为aborted。
2. 生成MAP文件，分别是在本地路径下备份和程式指定的路径中。
3. 调用事件通知界面一次task完成，更新下发的task列表。
4. 置标志位，防止重复调用。