# 性能优化专题

## 程序性能优化

主讲人：ROBERT: 2831742582

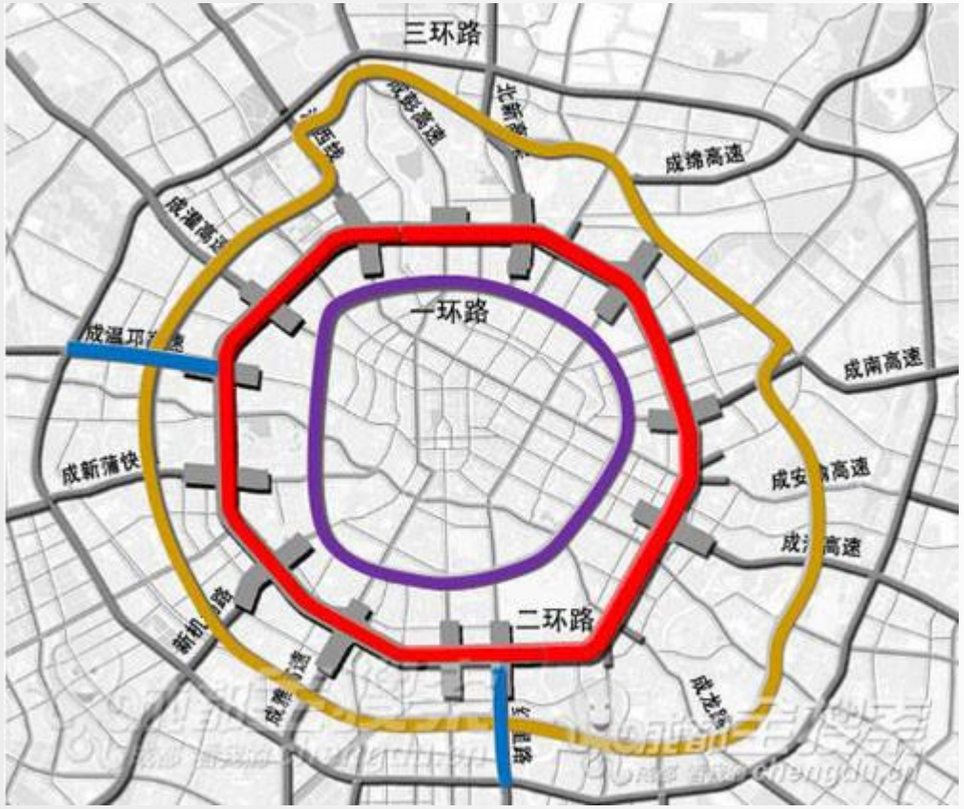
# 目录 /CONTENTS

# 01

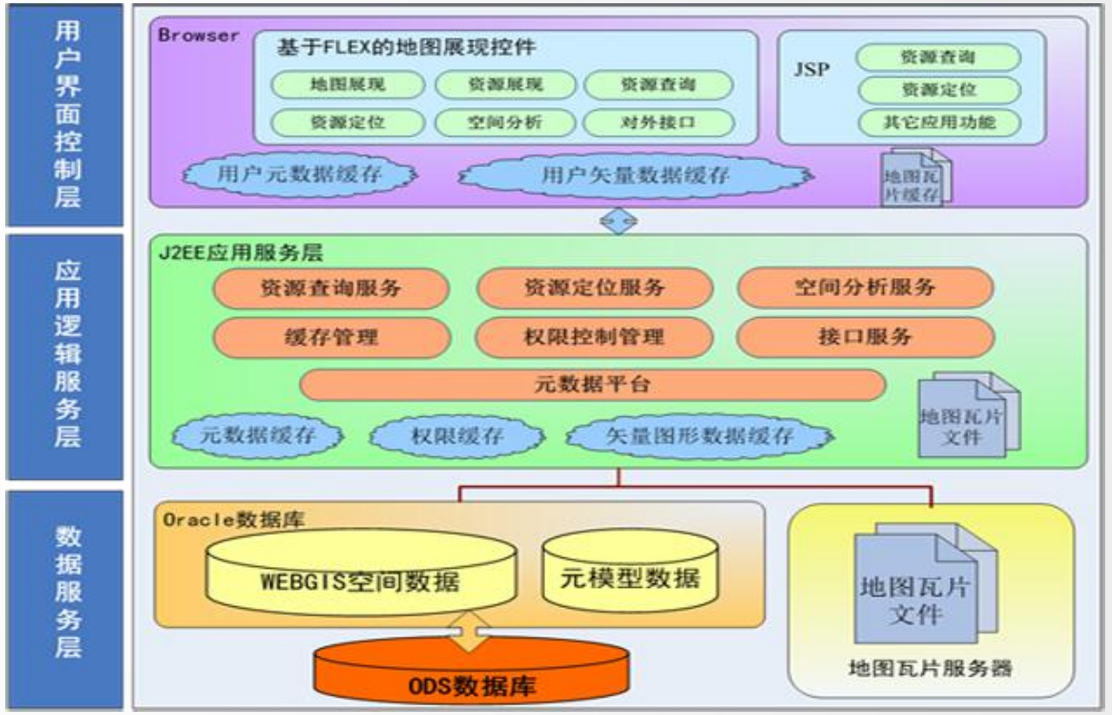
## 优化策略

OPTIMIZATION STRATEGY

# 不要过早进行优化？？？战略级的优化思路应在设计之初就建立




成都交通规划设计图




软件初始设计，模块，架构，很难深远的思考性能瓶颈，模块规模等因素。但应以适合优化的角度实施开发。

# 战略优化思路1


低调

假设我们是最不会编程

随时准备使用最先进的模块功能接入，由后续优化的模块或第三方更优的模块提供高效的访问与处理。

优雅

面向过程语言是更快的

但是面向对象，面向领域，面向切面的编程会更容易优化和扩容。

# 战略优化思路2

全局战略优化措施

开发框架的选择
数据存储的设计
模块层级的衔接与拆分

局部战略优化措施

1、单一职责，单例模式
2、开闭原则，对扩展开放，对修改关闭
3、里氏替换原则，使用基类的地方可以使用子类
4、依赖倒置原则，依赖于抽象，依赖于接口
5、接口隔离原则，类间的依赖关系应该建立在最小的接口上
6、迪米特原则，最少知识，信息隐藏

# 02

## 编码原则

**CODING PRINCIPLE**

# 代码优化的总目标

## 战略目标

1、高级可扩展的框架（可任意切换相关功能组成部分）
2、逻辑模块划分，分层、清晰、功能及算法可替换

## 依托战略

1、一切与战略冲突的代码优化，都是错误的
2、编程方式将向战略计划靠拢

## 优化目标

1、更少的代码（减少体积）
2、更高的效率（提高效率）

# 35条军规

**1、尽量指定类、方法的final修饰符，** 虚拟机会想办法内联所有的final方法
**2、尽量重用对象，** 生存还是灭亡，这是个问题(哈姆雷特)
**3、尽可能使用局部变量，** 一些局部变量是在栈中，免了垃圾回收过程
**4、及时关闭流，** 不说了，资源浪费问题
**5、尽量减少对变量的重复计算，** list.size()在循环中的处理
**6、尽量采用懒加载的策略，即在需要的时候才创建，** 还是创建对象问题，比如在if块外创建了对象
**7、慎用异常，** 异常只能用于错误处理，不应该用来控制程序流程。
**8、不要在循环中使用try…catch…，应该把其放在最外层**
**9、如果能估计到待添加的内容长度，为底层以数组方式实现的集合、工具类指定初始长度，**
new HashMap(256)，StringBuilder(int size)
**10、当复制大量数据时，使用System.arraycopy()命令，** 采用native
**11、乘法和除法使用移位操作，** 乘法：<<，除法：>>
**12、循环内不要不断创建对象引用，** Object obj = null; for (int i = 0; i <= count; i++) { obj = new Object(); }
**13、基于效率和类型检查的考虑，应该尽可能使用array，无法确定数组大小时才使用ArrayList**
**14、**尽量使用HashMap、ArrayList、StringBuilder，除非线程安全需要，否则不推荐使用Hashtable、Vector、StringBuffer，
后三者由于使用同步机制而导致了性能开销
**15、不要将数组声明为public static final，** public不安全，而final没有用

# 35条军规

**16、尽量在合适的场合使用单例，** 节省加载开销
**17、尽量避免随意使用静态变量，** gc通常是不会回收， public class A{ private static B b = new B(); }，B会在A清除后才清除
**18、及时清除不再需要的会话，** HttpSession的invalidate()方法清除会话。
**19、实现RandomAccess接口的集合比如ArrayList，应当使用最普通的for循环而不是foreach循环来遍历**
假如是随机访问的，使用普通for循环效率将高于使用foreach循环；反过来，如果是顺序访问的，则使用Iterator会效率更高
**20、使用同步代码块替代同步方法**
**21、将常量声明为static final，并以大写命名，** 常量池
**22、不要创建一些不使用的对象，不要导入一些不使用的类，** not use的警告，可以清除
**23、程序运行过程中避免使用反射，** 尽量在启动时就反射完成，大家不关心启动花了多长时间
**24、使用数据库连接池和线程池，** 这个都有了，可以使用的连接池：c3p0,dbcp...
**25、使用带缓冲的输入输出流进行IO操作,** 即BufferedReader、BufferedWriter、BufferedInputStream、BufferedOutputStream，提升IO效率
**26、顺序插入和随机访问比较多的场景使用ArrayList，元素删除和中间插入比较多的场景使用LinkedList**
**27、不要让public方法中有太多的形参，** 能用类就用类，不要直接使用属性
**28、字符串变量和字符串常量equals的时候将字符串常量写在前面**
**29、请知道，在java中if (i == 1)和if (1 == i)是没有区别的，但从阅读习惯上讲，建议使用前者**
**30、不要对数组使用toString()方法，** 输出没有意义
**31、不要对超出范围的基本数据类型做向下强制转型，** 长整型不适合强转为整型数
**32、公用的集合类中不使用的数据一定要及时remove掉，** 因为内存泄漏问题
**33、**把一个基本数据类型转为字符串，基本数据类型.toString()是最快的方式、String.valueOf(数据)次之、数据+""最慢
**34、使用最有效率的方式去遍历Map，** iterator接口
**35、对资源的close()建议分开操作，** 分开关闭不同的资源，避免未释放

# 附加的

JDK7中新引入的Objects工具类，对象比较

避免使用正则表达式,使用Apache Commons Lang作为代替

远离递归.递归会占用大量资源!

避免Random实例被多线程使用，虽然共享该实例是线程安全的，但会因竞争同一seed 导致的性能下降，JDK7之后，可以使用ThreadLocalRandom来获取随机数

静态类、单例类、工厂类将它们的构造函数置为private

提前编译正则表达式

尽可能地缓存

# 03

## 善用工具

USE TOOLS

**什么是静态代码分析**

静态代码分析是指无需运行被测代码，仅通过分析或检查源程序的语法、结构、过程、接口等来检查程序的正确性，找出代码隐藏的错误和缺陷，如参数不匹配，有歧义的嵌套语句，错误的递归，非法计算，可能出现的空指针引用等等。

在软件开发过程中，静态代码分析往往先于动态测试之前进行，同时也可以作为制定动态测试用例的参考。统计证明，在整个软件开发生命周期中，30% 至 70% 的代码逻辑设计和编码缺陷是可以通过静态代码分析来发现和修复的。

但是，由于静态代码分析往往要求大量的时间消耗和相关知识的积累，因此对于软件开发团队来说，使用静态代码分析工具自动化执行代码检查和分析，能够极大地提高软件可靠性并节省软件开发和测试成本。

# 善用工具

1、下载 pmd-bin-6.2.0.zip
(https://github.com/pmd/pmd/releases/download/pmd_releases%2F6.2.0/pmd-bin-6.2.0.zip)

2.解压到：类似于：C:\pmd-bin-6.2.0
3.添加到环境变量Path 临时处理：SET PATH=C:\pmd-bin-6.2.0;%PATH%

4.命令行处理:pmd.bat -d c:\src -R java-basic -f text

它的能力：
**可能的bug——try/catch/finally/switch语句中返回空值。**
**死代码——未使用的局部变量、参数、私有方法。**
**不理想的代码——使用String/StringBuffer。**
**过于复杂的表达式——没有必要使用if语句、while循环可以代替for循环。**
**重复代码——复制/粘贴的代码引发的bug。**
**PMD集成了JDeveloper, Eclipse, JEdit, JBuilder, BlueJ, CodeGuide, NetBeans, IntelliJ IDEA, TextPad, Maven, Ant, Gel, JCreator, 以及 Emacs。**

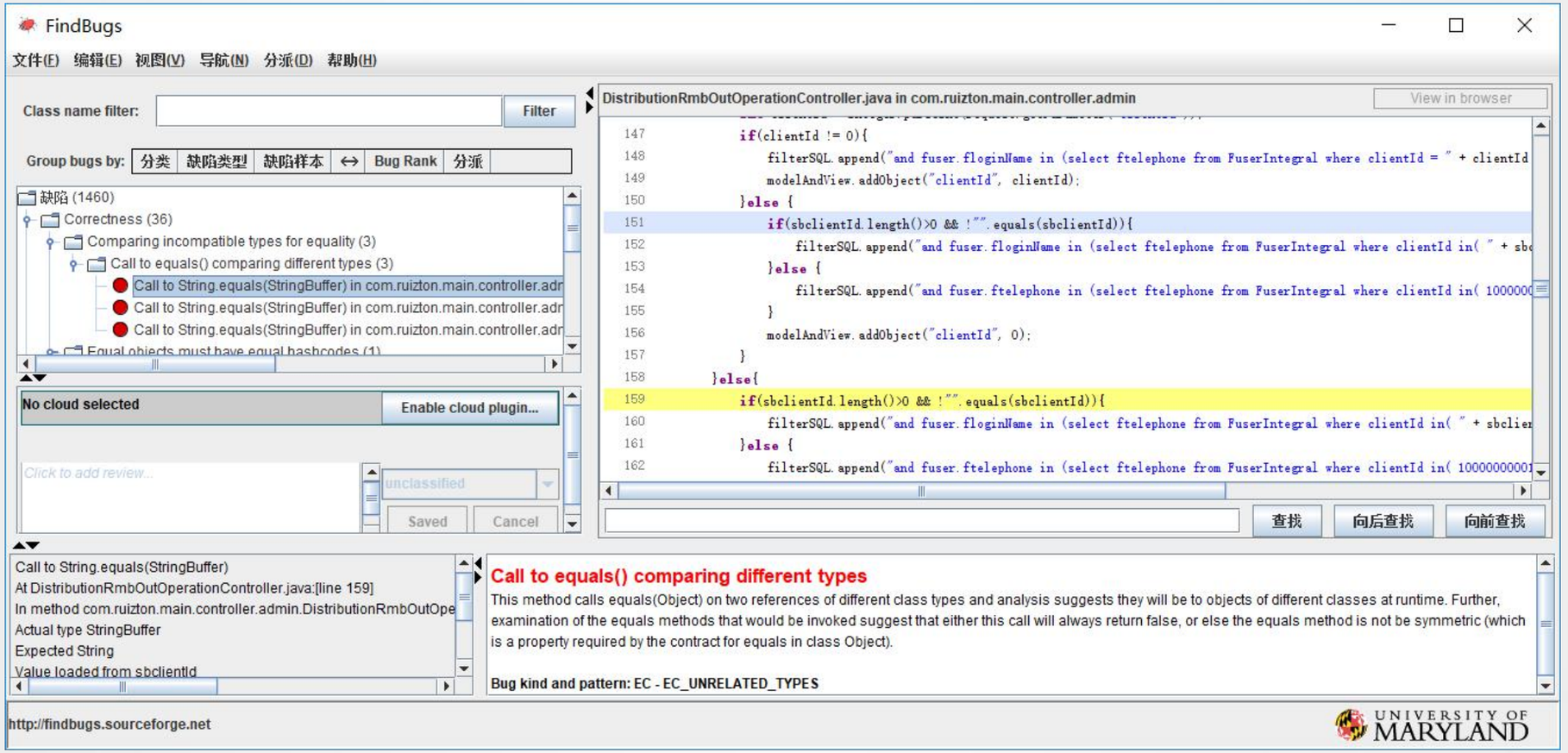
# 善用工具

| 工具 | 目的 | 检查项 |
| --- | --- | --- |
| **FindBugs**<br>**检查.class** | 基于Bug Patterns概念，查找javabytecode（.class文件）中的潜在bug | 主要检查bytecode中的bug patterns，如code 性能、NullPoint空指针检查、没有合理关闭资源、字符串相同判断错（==，而不是equals）等 |
| **PMD**<br>**检查源文件** | 检查**Java**源文件中的潜在问题 | 主要包括：<br>空try/catch/finally/switch语句块<br>未使用的局部变量、参数和private方法<br>空if/while语句<br>过于复杂的表达式，如不必要的if语句等<br>复杂类 |
| **CheckStyle**<br>**检查源文件**<br>**主要关注格式** | 检查Java源文件是否与代码规范相符 | 主要包括：<br>Javadoc注释<br>命名规范<br>多余没用的Imports<br>Size度量，如过长的方法<br>缺少必要的空格Whitespace<br>重复代码 |

# 善用工具

**2. FindBug from http://findbugs.sourceforge.net**

**FindBug是一个使用静态方法来查找Java代码漏洞的程序。**



# 善用工具

http://checkstyle.sourceforge.net/

Checkstyle是一款检查Java程序源代码样式的工具，它可以有效的帮助我们检视代码以便更好的遵循代码编写标准，特别适用于小组开发时彼此间的 样式规范和统一。Checkstyle提供了高可配置性，以便适用于各种代码规范，所以除了使用它提供的几种常见标准之外，你也可以定制自己的标准。 Checkstyle提供了支持大多数常见IDE的插件，大部分插件中就含有最新的Checkstyle。

Windows->prefences->checkstyle:

# 04

## 异步、缓存

ASYNCHRONOUS, CACHING

# 异步开发

定时器，使用间隔的查询，合理的脱耦

外部请求（短信等），使用线程池确保请求能够快速的响应

•1）newCachedThreadPool 是一个可根据需要创建新线程的线程池，但是在以前构造的线程可用时将重用它们。对于执行很多短期异步任务的程序而言，这些线程池通常可提高程序性能。调用 execute() 将重用以前构造的线程（如果线程可用）。如果现有线程没有可用的，则创建一个新线程并添加到池中。终止并从缓存中移除那些已有 60 秒钟未被使用的线程。因此，长时间保持空闲的线程池不会使用任何资源。注意，可以使用 ThreadPoolExecutor 构造方法创建具有类似属性但细节不同（例如超时参数）的线程池。

•2）newSingleThreadExecutor 创建是一个单线程池，也就是该线程池只有一个线程在工作，所有的任务是串行执行的，如果这个唯一的线程因为异常结束，那么会有一个新的线程来替代它，此线程池保证所有任务的执行顺序按照任务的提交顺序执行。

•3）newFixedThreadPool 创建固定大小的线程池，每次提交一个任务就创建一个线程，直到线程达到线程池的最大大小，线程池的大小一旦达到最大值就会保持不变，如果某个线程因为执行异常而结束，那么线程池会补充一个新线程。

•4）newScheduledThreadPool 创建一个大小无限的线程池，此线程池支持定时以及周期性执行任务的需求。

# 缓存及工具

堆缓存，**Guava Cache、EhCache、MapDB**

堆外缓存，**Guava Cache、EhCache、MapDB**

磁盘缓存，**EhCache、MapDB**

分布式缓存，**Redis**

# 缓存及工具

Guava示例

```
Cache<String,String>myCache=
CacheBuilder.newBuilder().concurrencyLevel(4).expireAfterWrite(10,TimeUnit.SECONDS)        .maximumSize(10000).build();
```

# 缓存及工具

EhCache示例

```
CacheManager cacheManager = CacheManagerBuilder.newCacheManagerBuilder().
build(true);CacheConfigurationBuilder<String,String>cacheConfig=
CacheConfigurationBuilder.newCacheConfigurationBuilder(String.class,String.class,
ResourcePoolsBuilder.newResourcePoolsBuilder()
.heap(100,EntryUnit.ENTRIES))
.withDispatcherConcurrency(4)
.withExpiry(Expirations.timeToLiveExpiration(Duration.of(10,TimeUnit.SECONDS)));
Cache<String, String> myCache = cacheManager.createCache("myCache",cacheConfig);
```

# 缓存及工具

EhCache示例

```
HTreeMap myCache =
DBMaker.memoryDirectDB().concurrencyScale(16).make().hashMap("myCache")
.expireStoreSize(64 * 1024 * 1024) //指定堆外缓存大小64MB
.expireMaxSize(10000)
.expireAfterCreate(10, TimeUnit.SECONDS)
.expireAfterUpdate(10, TimeUnit.SECONDS)
.expireAfterGet(10, TimeUnit.SECONDS)
.create();
```