Real-world Java 4 beginners

Dima, Superfish
github.com/dimafrid

Real world

- Hundreds of computations per second
  - Each computation of sub-second latency
- Massive IO
- Lots of data
- In Web/Trading/Monitoring/Cellular/etc./etc.
- Squeezing every bit of juice out of HW

Real world – case study

- Google does 10B searches a day keeping entire Web in the belly
- Twitter does 3000 tweets a minute (fanning out to 600K users)
- Netflix is responsible for 30% of US traffic in certain hours

Real world – case study

- Superfish
  - 150M requests a day
  - 150K requests a minute @ peak time
  - Must have sub-500ms response time
  - ~0.5B data records
  - 300K KPIs a minute
  - Hundreds of machines

“Whatever you desire”:

- Core stuff (IO, concurrency, GC)
- Application development (frameworks/open source recommendations/best practices)
- Monitoring (what, tools)

Core Java stuff - concurrency

- All about parallel utilization of HW resources
- Basic multi-threading: wait() & notify()
- java.util.concurrent – why?
  - Introduced by Doug Leah in Java 5
  - Thread pools
  - Atomic counters
  - Lock-less data structures
  - Smart synchronizers
  - Futures

Core Java stuff - concurrency

- Thread pools
  - Creating thread is expensive (memory allocation/”forking”/GC/book-keeping)
  - Solution: pre-create (pooling)
- Atomic counters
  - ++cnt is not thread-safe operation
- Lock-less data structures
  - Locking is expensive
- Smart synchronizers

Core Java stuff - IO

- Readers/writers for EVERYTHING: file/socket/string/object (serialization)
- Comprehensive javadoc
- Buffering
  - Read in advance
  - Making IO effective in terms of system calls

Core Java stuff - IO

- BIO (blocking IO)
  - Wait until data is available
- NIO (non-blocking IO)
  - Introduced relatively late in Java, somehow still lagging
  - Old & good idea of notifying whenever data is available
  - Single reading loop calling back upon data availability
- BIO vs. NIO – real life example:
  - BIO: hundreds of threads, machine dead
  - NIO: 6 data processing threads
  - BIO straightforward, NIO harder to implement

Core Java stuff - GC

- There is no explicit memory deallocation in Java
  - Garbage collector frees allocated memory
- **Poorly tuned GC in heavy load env = major contributor to high latency**
- Definitely an expertise

Core Java stuff - GC

- Common model
  - Reach-ability from roots (static & threads)
  - Based on assumption that some objects are more durable than others
  - New and old gens, survivals, different collectors
- G1
  - New
  - No personal experience, so won't b**s you

Core Java stuff - GC

- Tuning
  - Dozens of parameters
  - **Understand your memory patterns!**
  - High-throughput/low-pause oriented collectors
  - Benchmarks unavoidable
  - Diagnostics: GC log

Core Java stuff – Memory

- Couldn't resist this one:
- “No memory leaks in Java” - good reason to terminate
  - Even w/o esoteric scenarios, creating a memory leak in Java is trivial
- “More memory is better” - like saying more butter is better
  - For better taste – yes
  - For avoiding a coronary - no

Core Java stuff – others

- JDBC
- Generics
- Data structures (java.util) – know your data structure under-the-hood
  - really heavy-stuff of tuning hash maps, for example: need to understand the implementation
- Etc. etc.

Application development

- A set of frameworks/toolkits that essentially
  - Provide integration with other SW
  - Make life easier – no need to write everything yourself
  - Speed of development
- Let's talk about the most-wanted of Java app development

Application development - Spring

- Essentially an integration framework
- Origin lays in **IoC/dependency-injection** model within a **container** of **Spring beans**
  - Spring bean is an instance of Java class declared in container definition
  - Deriving population/initialization from declared dependencies
- On top of the container, there are integrations:
  - Remoting, DB, unit testing, scheduling
  - And messaging, AOP, etc. etc. etc.

Application development - Spring

```
<context:component-scan base-package="com.superfish.fbeng"/>

<context:annotation-config/>

<bean id="placeholderConfig" class="org.springframework.beans.factory.config.PropertyPlaceholderConfigurer">
        <property name="locations">
            <list>
                <value>classpath*:config/*.properties</value>
            </list>
        </property>
</bean>

<bean id="statsPersister" class="com.superfish.realtime.services.search.StatsPersister">
        <property name="sessionFactory" ref="statsSessionFactory"/>
        <property name="persistOnceEveryX" value="1"/>
        <property name="peristThreadCount" value="10"/>
</bean>
```

Application development - Spring

```
@Repository
public class PhotoDaoImpl extends BaseJpaDaoImpl<String, PhotoEntity>
    implements PhotoDao {
…..
   @Resource(name = "dataSource")
   private DataSource ds;


   @Autowired
   private PhotoDao imageDao;
…..
}
```

Application development - Spring

- Well-written (at least whatever I hacked)
- Rich
- Convenient
- Spend some time learning the internals (especially DB-related stuff)
- Every recruiter recognizes a “Spring” word ...

Application development - REST

- How do you make 2 machines talk to each other (HTTP implied)?
- SOAP
  - Like calling a method
  - Attempts to cover everything
  - Like everything that wants to be perfect - dead

Application development - REST

- REST emerged
  - Apart from being a Ph.D. material, it's essentially like a page exchange over HTTP
  - Simple because it's modeled after HTTP
  - Implementations (REST is merely an idea with standardization):
    - Wink (personal experience)
    - Jersey
    - RestEasy

Application development - JPA

- Probably most important counter-part of any application today is DB
- JPA bridges between OO world and relational DB

Application development - Servers

- The basic component of Java web server is **servlet container**
- Servlet container is a place to put **web applications**
- Web application is a collection of **servlets** (and everything needed to run their code) and mapping of URLs to those servlets:

Application development - Servers

- tau.me:9090/rwj4b/search?student=Mark%20Zuckerberg
- **rwj4b** – web app name
- **search** – path of reqest
- **student=Mark%20Zuckerberg** - query params

Application development - Servers

- <servlet>
- <servlet-name>dummy</servlet-name>
- <servlet-class>DummyServlet</servlet-class>
- </servlet>
- <servlet-mapping>
- <servlet-name>dummy</servlet-name>
- <url-pattern>**search**/*</url-pattern>
- </servlet-mapping>

Application development - Servers

```
public class DummyServlet extends HttpServlet {

    protected void doGet(HttpServletRequest req,
    HttpServletResponse resp) throws
    ServletException, IOException {

        String student = req.getParameter("student");

        resp.getWriter().write(student + " digs real life
    java");

    }

}
```

Application development – VM (dynamic) languages

- Scala
- Groovy
- Jython
- JRuby

Application development - management

- Not really directly related to Java
- Version management
  - ClearCase (rolls royce, but expensive and requires management)
  - Subversion (oldie, mediocre, choice of many)
  - Git (version management is about branching, and that's what it does best; complicated as hell for non-vanilla use-cases)
- Project management
  - Don't really have any experience with anything but Maven
  - Transparent dependency management

Application development - practice

- If you think you're missing a very important infrastructure:
  - Don't write
  - Find an open-source
  - Understand how it works and then use/Throw away and write yourself
- Apache.org: richness, quality – your first address
- Google open-sources state-of-the-art SW
  - Collections (academic stuff)
  - Gson
  - …
- Unit-test (Junit, NG something) – not compulsively

Monitoring

- When you deal with hundreds of millions of applicative operations, you have to understand what's going on
- Local monitoring
- System profiling
- Visualization

Monitoring – Local (Java-level)

- Thread dumps
- GC logs
- Memory distribution
- Applicative logging
  - Good logging requires thorough thinking as it's a valid basis for further analysis
  - Bad logging kills performance
- JMX
  - Built-in ability to plug-in and access your custom code
  - Widely used for diag