Устранение утечки памяти с помощью снятия дампа кучи

1. Разработать сервис, позволяющий регистрировать пользователей и выполнять их аутентификацию. Аутентификация выполняется с помощью Spring Security, для получения информации о пользователе используется кастомная реализация UserDetailsService с утечкой памяти.
   В репозитории в файле README.md указаны параметры для запуска приложения и команды, которые необходимо выполнить, чтобы воспроизвести утечку памяти.
   В конфигурационном файле application.properties задается количество пользователей, которые необходимо создать при старте. При тестировании было указано users.count=100000

2. В первом коммите реализовать в UserDetailsService получение данных о пользователе с помощью загрузки всей таблицы пользователей в оперативную память
   Реализация:
```
@Service
@RequiredArgsConstructor
public class UserDetailsServiceImpl implements UserDetailsService {
   private final UserInfoRepository userInfoRepository;

   @Override
   public UserDetails loadUserByUsername(String username) throws
UsernameNotFoundException {
       return userInfoRepository.findAll().stream()
               .filter(userInfo -> userInfo.getUsername().equals(username))
               .findAny()
               .map(CustomUserDetails::new)
               .orElseThrow(() -> new UsernameNotFoundException(username));
   }
}
```

3. Запустить приложение с параметрами
   `-Xms128m -Xmx128m -XX:+HeapDumpOnOutOfMemoryError -XX:HeapDumpPath="/Users/alex/dump_find_all.hprof"`

4. После сообщения в консоли “Users created” выполнить скрипт из раздела “Authenticate users” в README.md и ждать ошибки OutOfMemory

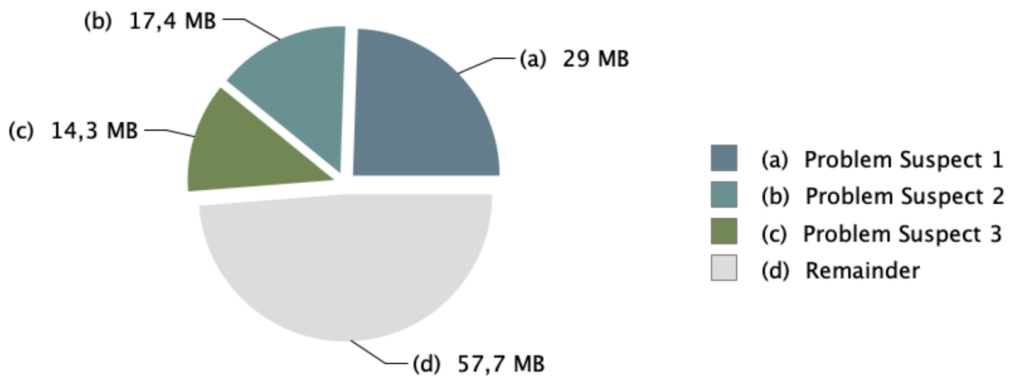
5. После получения ошибки открыть дамп в Eclipse Memory Analyzer Tool и построить отчет об утечках памяти. В нем видно, что создаются 4 объекта org.h2.result.LocalResult, принадлежащие потокам, обрабатывающим HTTP-запросы. Остальные проблемы относятся к H2

Leak Suspects

System Overview

Leaks

Overview



Total: 118,4 MB

Problem Suspect 1

300 003 instances of **"org.h2.value.ValueVarchar"**, loaded by **"jdk.internal.loader.ClassLoaders$AppClassLoader @ 0x7fb37c2e8"** occupy **30 361 784 (24,46 %)** bytes.

Keywords

org.h2.value.ValueVarchar
jdk.internal.loader.ClassLoaders$AppClassLoader

Details »

Problem Suspect 2

4 instances of **"org.h2.result.LocalResult"**, loaded by **"jdk.internal.loader.ClassLoaders$AppClassLoader @ 0x7fb37c2e8"** occupy **18 212 672 (14,67 %)** bytes.

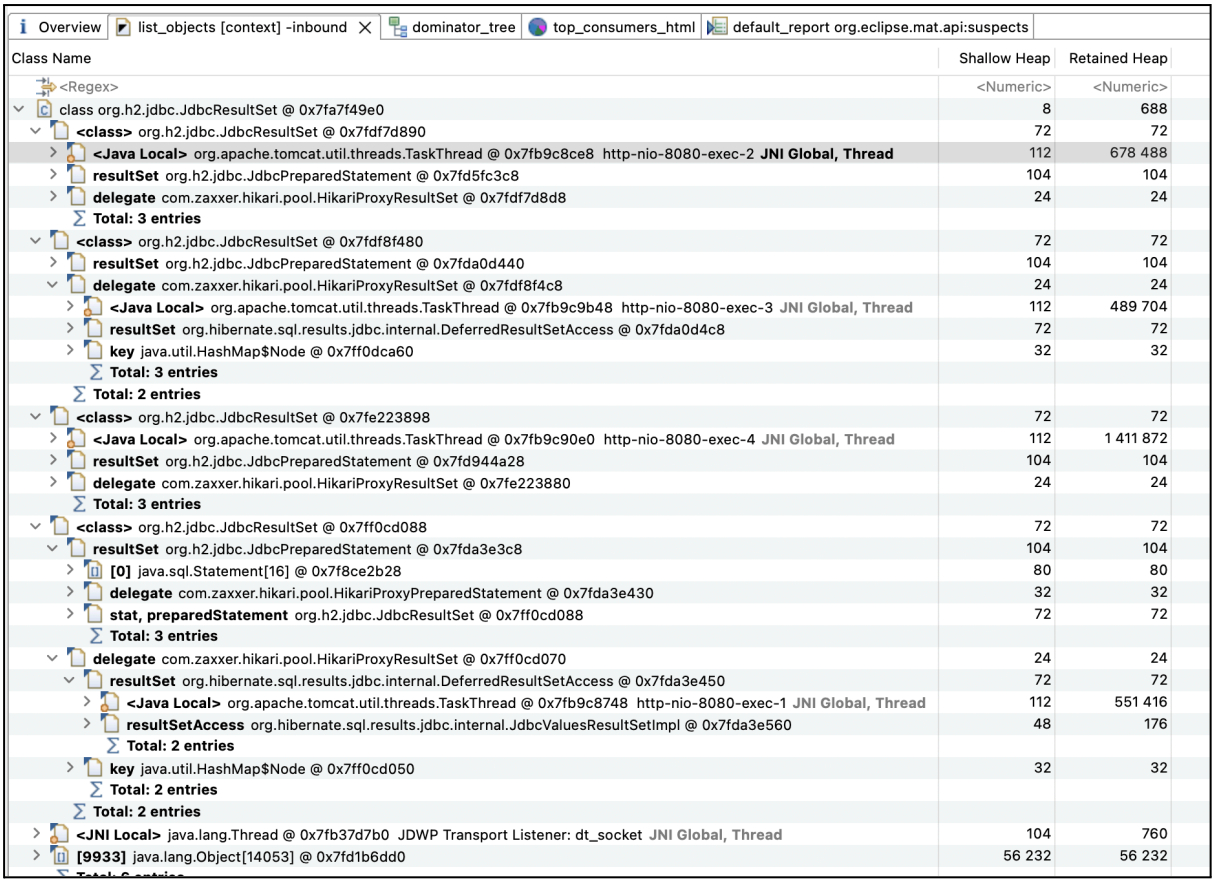
Biggest instances:

- org.h2.result.LocalResult @ 0x7fd5fc248 - 4 553 168 (3,67 %) bytes.
- org.h2.result.LocalResult @ 0x7fd9448a8 - 4 553 168 (3,67 %) bytes.
- org.h2.result.LocalResult @ 0x7fda0d2c0 - 4 553 168 (3,67 %) bytes.
- org.h2.result.LocalResult @ 0x7fda3e248 - 4 553 168 (3,67 %) bytes.

Keywords

org.h2.result.LocalResult
jdk.internal.loader.ClassLoaders$AppClassLoader

Details »



Class Name	Shallow Heap	Retained Heap
<Regex>	<Numeric>	<Numeric>
class org.h2.jdbc.JdbcResultSet @ 0x7fa7f49e0	8	688
<class> org.h2.jdbc.JdbcResultSet @ 0x7fdf7d890	72	72
<Java Local> org.apache.tomcat.util.threads.TaskThread @ 0x7fb9c8ce8 http-nio-8080-exec-2 **JNI Global, Thread**	112	678 488
resultSet org.h2.jdbc.JdbcPreparedStatement @ 0x7fd5fc3c8	104	104
delegate com.zaxxer.hikari.pool.HikariProxyResultSet @ 0x7fdf7d8d8	24	24
Total: 3 entries		
<class> org.h2.jdbc.JdbcResultSet @ 0x7fdf8f480	72	72
resultSet org.h2.jdbc.JdbcPreparedStatement @ 0x7fda0d440	104	104
delegate com.zaxxer.hikari.pool.HikariProxyResultSet @ 0x7fdf8f4c8	24	24
<Java Local> org.apache.tomcat.util.threads.TaskThread @ 0x7fb9c9b48 http-nio-8080-exec-3 JNI Global, Thread	112	489 704
resultSet org.hibernate.sql.results.jdbc.internal.DeferredResultSetAccess @ 0x7fda0d4c8	72	72
key java.util.HashMap$Node @ 0x7ff0dca60	32	32
Total: 3 entries		
Total: 2 entries		
<class> org.h2.jdbc.JdbcResultSet @ 0x7fe223898	72	72
<Java Local> org.apache.tomcat.util.threads.TaskThread @ 0x7fb9c90e0 http-nio-8080-exec-4 JNI Global, Thread	112	1 411 872
resultSet org.h2.jdbc.JdbcPreparedStatement @ 0x7fd944a28	104	104
delegate com.zaxxer.hikari.pool.HikariProxyResultSet @ 0x7fe223880	24	24
Total: 3 entries		
<class> org.h2.jdbc.JdbcResultSet @ 0x7ff0cd088	72	72
resultSet org.h2.jdbc.JdbcPreparedStatement @ 0x7fda3e3c8	104	104
[0] java.sql.Statement[16] @ 0x7f8ce2b28	80	80
delegate com.zaxxer.hikari.pool.HikariProxyPreparedStatement @ 0x7fda3e430	32	32
stat, preparedStatement org.h2.jdbc.JdbcResultSet @ 0x7ff0cd088	72	72
Total: 3 entries		
delegate com.zaxxer.hikari.pool.HikariProxyResultSet @ 0x7ff0cd070	24	24
resultSet org.hibernate.sql.results.jdbc.internal.DeferredResultSetAccess @ 0x7fda3e450	72	72
<Java Local> org.apache.tomcat.util.threads.TaskThread @ 0x7fb9c8748 http-nio-8080-exec-1 JNI Global, Thread	112	551 416
resultSetAccess org.hibernate.sql.results.jdbc.internal.JdbcValuesResultSetImpl @ 0x7fda3e560	48	176
Total: 2 entries		
key java.util.HashMap$Node @ 0x7ff0cd050	32	32
Total: 2 entries		
Total: 2 entries		
<JNI Local> java.lang.Thread @ 0x7fb37d7b0 JDWP Transport Listener: dt_socket JNI Global, Thread	104	760
[9933] java.lang.Object[14053] @ 0x7fd1b6dd0	56 232	56 232

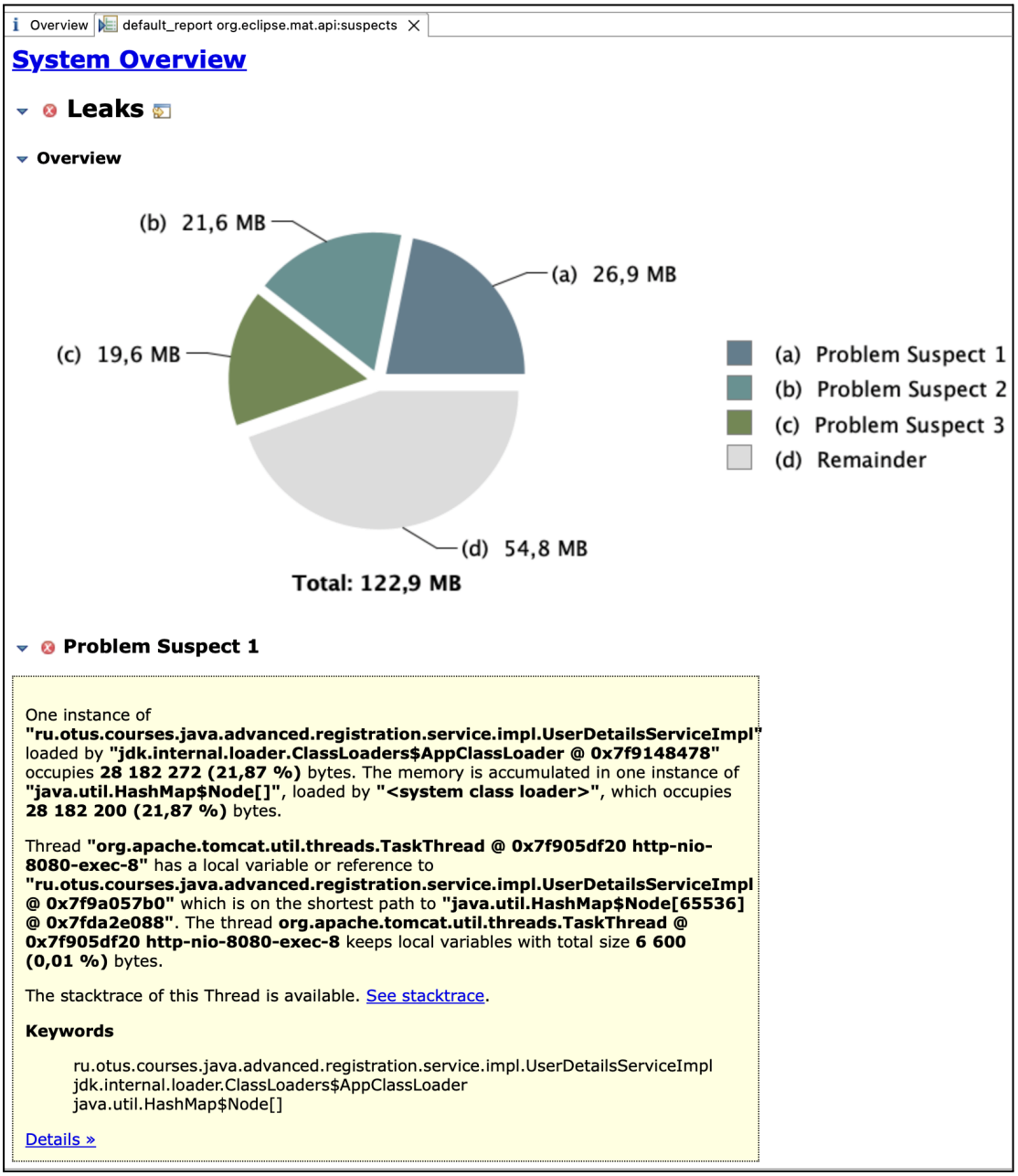
6. Исправить ошибку во 2м коммите - читать только 1 пользователя и сохранять его в кэше.
   Реализация:

```
@Service
@RequiredArgsConstructor
public class UserDetailsServiceImpl implements UserDetailsService {
   private final UserInfoRepository userInfoRepository;

   private final Map<String, UserDetails> userDetailsCache = new HashMap<>();

   @Override
   public UserDetails loadUserByUsername(String username) throws
UsernameNotFoundException {
       if (userDetailsCache.containsKey(username)) {
           return userDetailsCache.get(username);
       }

       UserDetails userDetails = userInfoRepository.findByUsername(username)
               .map(CustomUserDetails::new)
               .orElseThrow(() -> new UsernameNotFoundException(username));

       userDetailsCache.put(username, userDetails);
       System.out.printf("Cache size %d%n", userDetailsCache.size());

       return userDetails;
   }
}
```

7. Запустить приложение с параметрами
   `-Xms128m -Xmx128m -XX:+HeapDumpOnOutOfMemoryError -XX:HeapDumpPath="/Users/alex/dump_cache.hprof"`

8. После сообщения в консоли “Users created” выполнить скрипт из раздела “Authenticate users” в README.md и ждать ошибки OutOfMemory

9. После получения ошибки открыть дамп в Eclipse Memory Analyzer Tool и построить отчет об утечках памяти. В нем видно, что достаточно большой объем памяти занимает HashMap, принадлежащая UserDetailsServiceImpl. Остальные проблемы относятся к H2



10. Исправить ошибку в 3 коммите - заменить HashMap на WeapHashMap
    Реализация:

```java
@Service
@RequiredArgsConstructor
public class UserDetailsServiceImpl implements UserDetailsService {
   private final UserInfoRepository userInfoRepository;

   private final Map<String, UserDetails> userDetailsCache = new WeakHashMap<>();

   @Override
   public UserDetails loadUserByUsername(String username) throws
UsernameNotFoundException {
       if (userDetailsCache.containsKey(username)) {
           return userDetailsCache.get(username);
       }

       UserDetails userDetails = userInfoRepository.findByUsername(username)
               .map(CustomUserDetails::new)
               .orElseThrow(() -> new UsernameNotFoundException(username));

       userDetailsCache.put(username, userDetails);
       System.out.printf("Cache size %d%n", userDetailsCache.size());

       return userDetails;
   }
}
```

11. Запустить приложение с теми же параметрами, выполнить те же команды и убедиться, что ошибка устранена