Graduation Project
PRMA
weekly Report

20/10/2014 – 25/10/2014

Cai Bowen

25/10/2014

Progress:

Scheduled:

- Finish Basic Java Logger API
- Finish MongoDB based log storage
- Testing storage module

Progress:

- Finished MongoDB logger and schema design
- After a deeper insight into databases and some benchmark on MongoDB and MySQL 5.6, I change the storage solution to RDBMS, namely, H2database for embedded use and MySQL for server client use.
- Finished schema design, JDBC Logger API, and log store.

1. The MongoDB Logger

Existing MongoDB Java logger:

Logback-contrib.mongo: logback + mongoDB 2.10

https://github.com/qos-ch/logback-contrib

Log4mongo for Java: Log4J + mongoDB 2.10

http://log4mongo.org/display/PUB/Log4mongo+for+Java

Drawbacks of Existing MongoDB Java logger:

Inactive projects, non-mainstream practice, insufficient quality.

data storage structure is messy.

1. The MongoDB Logger

Drawbacks of MongoDB log storage solution:

Less normalized/schemaless data structure in mongoDB does not contribute to query and storage efficiency.

No, or no significant insertion & query performance gain from MongoDB, poorer aggregation speed on MongoDB.[1]

Current RDBMS are more reliable than NoSQL

2. NewRDBMS Solution

Store logs in RDBMS

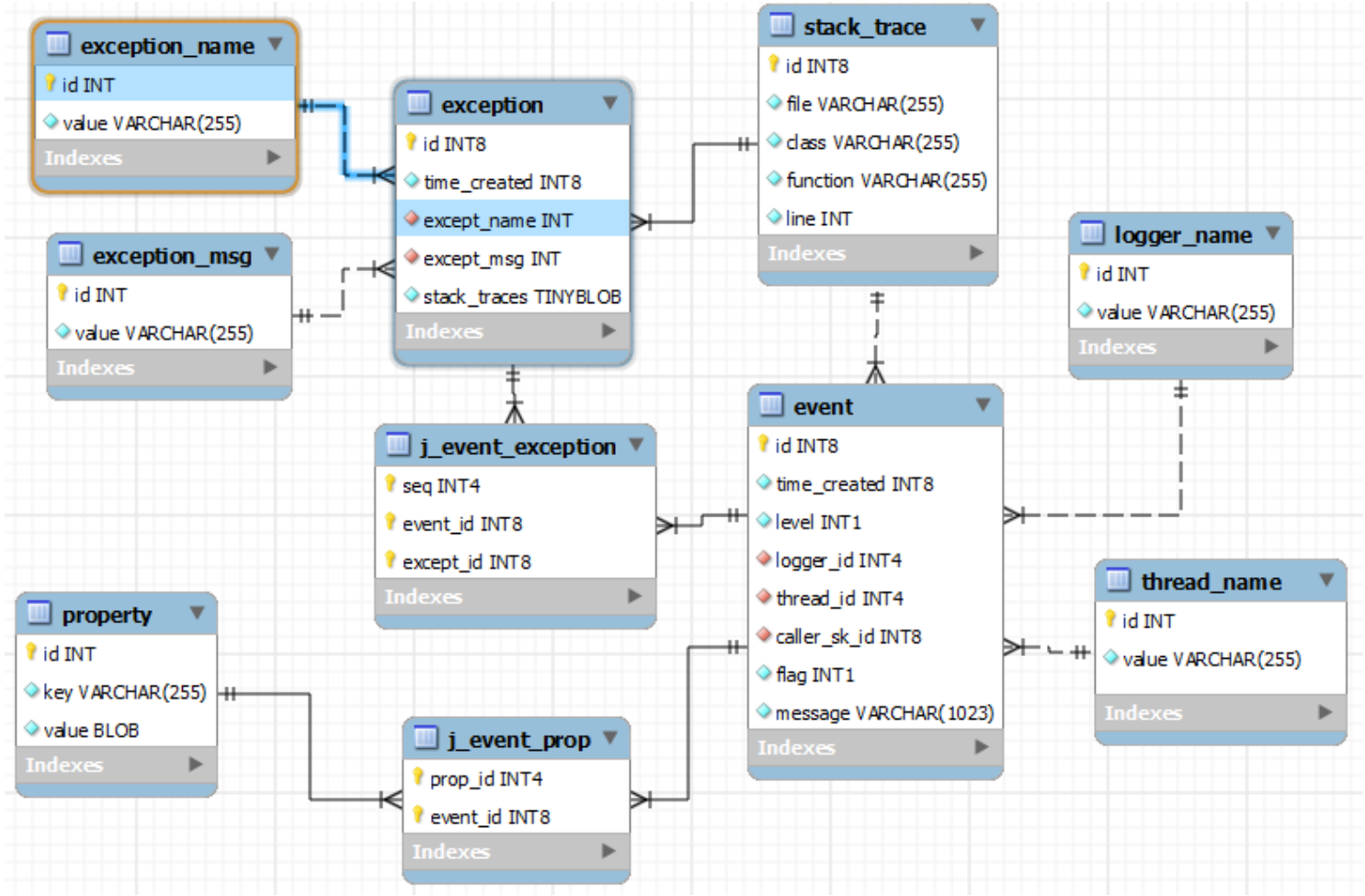
1, Embedded mode for programs on a single machine : the log storage API, monitor and H2database are embedded in the Logger, no extra process is created, provide querying API

2, Client-Server Mode for distributed logging: a centralized server, log server, receive logs from replications and store log to MySQL server farm. Monitoring and querying program is running on the log server.

First we focus on embedded mode.

3. Data Storage Design

Originally, Logback DBAppender store logs to 3 table: l*ogging_event*, *logging_event_property* and *logging_event_exception*. I store a log to 10 tables as shown below:



3. Data Storage Design

Pros:

More normalized data structure, less redundancy (test data will be provided later to proof it).

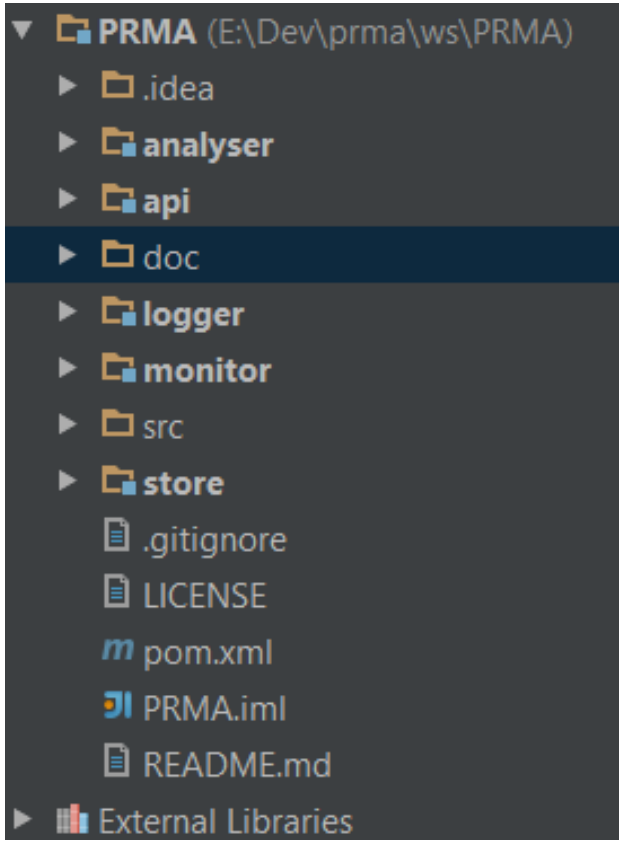
Faster insertion and query.

Cons:

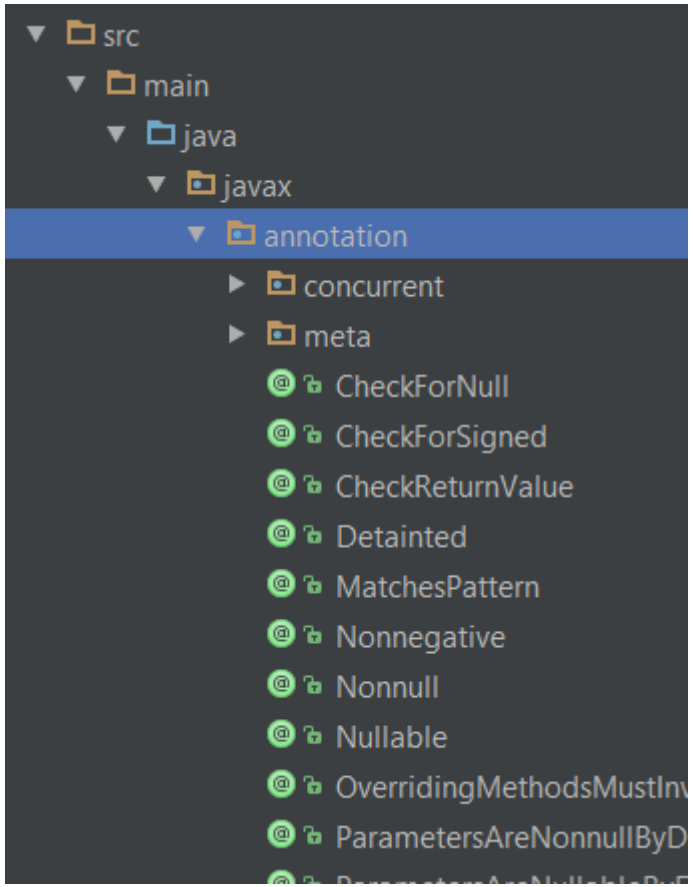
Extra complexity, difficult to manage transactions and failures.

Project information

There are 6 modules:

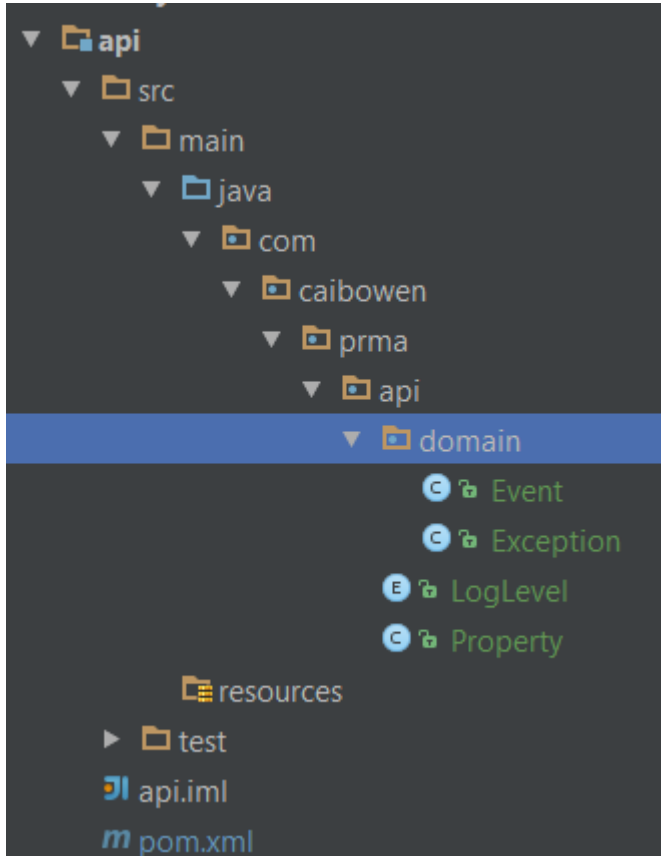


Module 1 parent: basic class and annotations

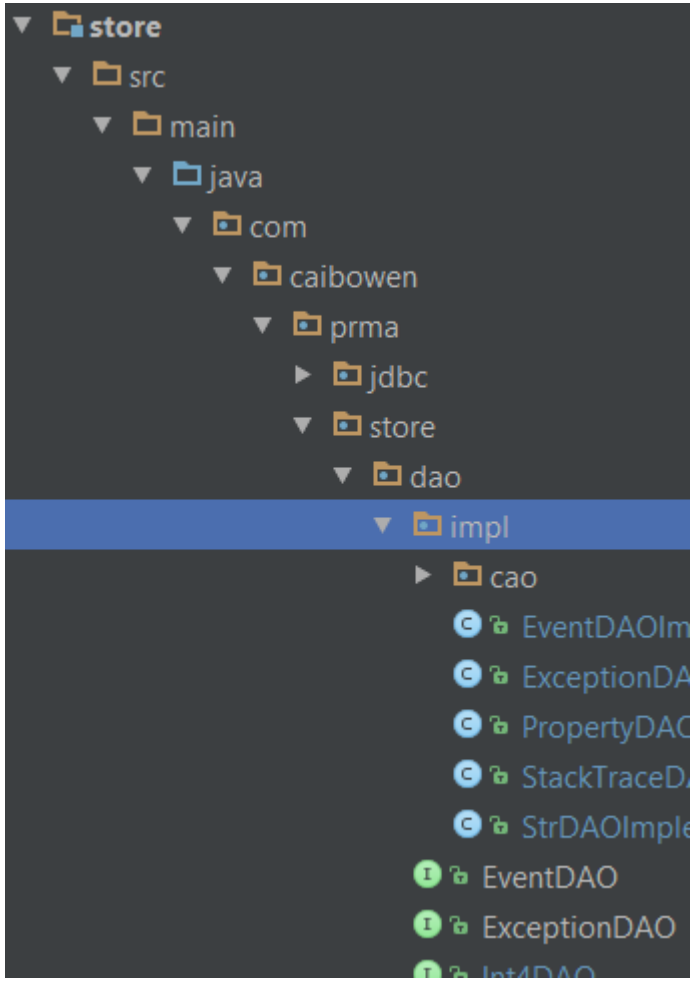


Project information

Module 2 api: domain model and api for external system

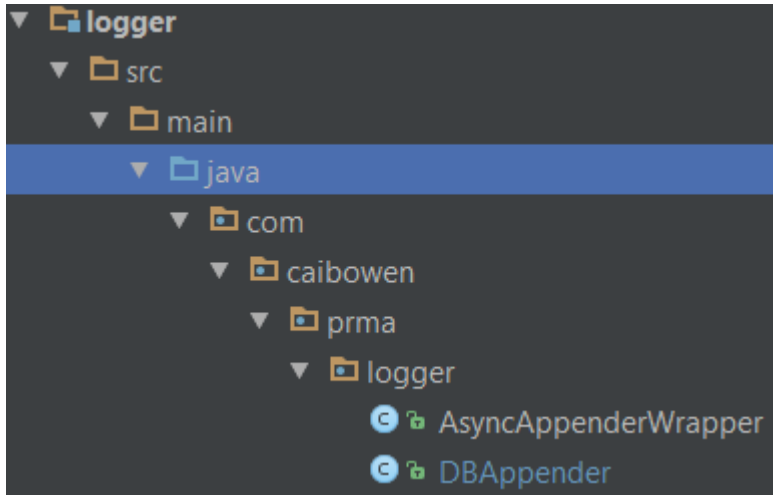


Module 3 store: low level data storage and query api, for internal use.



Project information

Module 4 logger: logger api
adopted to logback



Module 5 monitor : log event filtering , monitor config and callback

Module 6 analyzing: high level aggregation function and built-in analyzing-reporting tools

Project information

Project URL:
https://github.com/xkommando/PRMA

Note that the active branch is p2:
https://github.com/xkommando/PRMA/tree/p2

4. Next

Transaction management, Unit Testing, basic benchmarking.

Achieving High Reliability:

- Exception/failure handing system.
- Higher test coverage, verify and validate program

Achieving High Speed:

- Efficient data structure and algorithms
- Cache system (current for single machine embedding mode)
- custom aggregation build on database API, rather than complex queries