Regarding technical debt, students who are doing software development should be familiar with the following scenarios:

In order to dare to progress the project, detailed design, unit testing and other processes will not be written, and will be supplemented later.

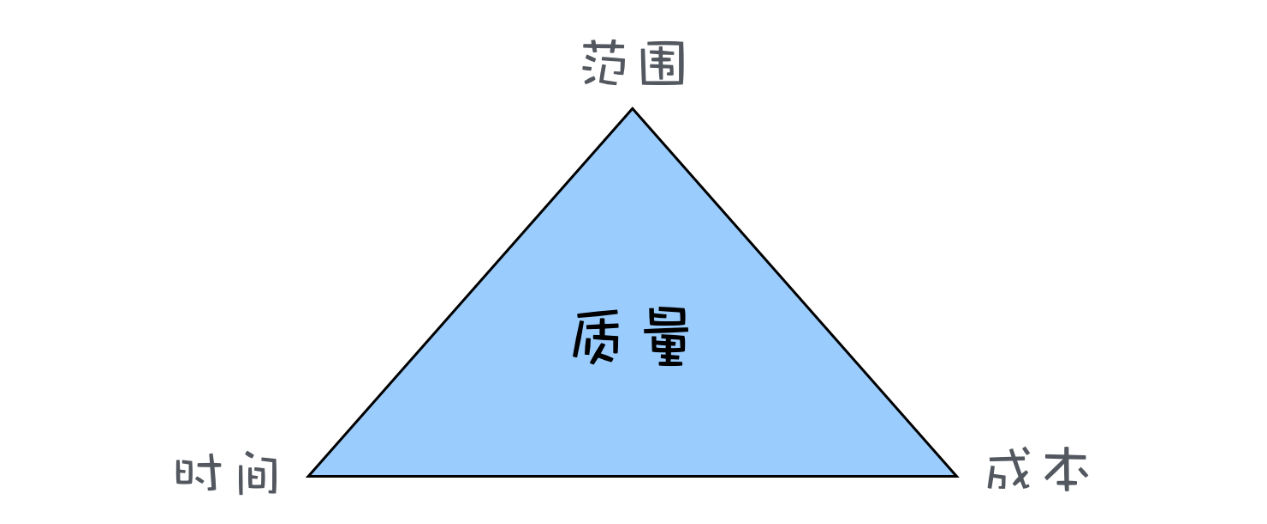
The demand has changed a lot. The original architecture design can't meet the new requirements, but I don't want to move the architecture, so I bypass the architecture design and add new code.

Old system, no documentation, no comments, difficult maintenance

As above, if we don't correct it in time, we will have more and more debts, which will lead to bloated code and inefficient system. We can use the term technical debt to describe the quality of the system.

 So what is technical debt?

  We know the triangle diagram, time, cost and scope of software quality when we are studying project management, as shown below:



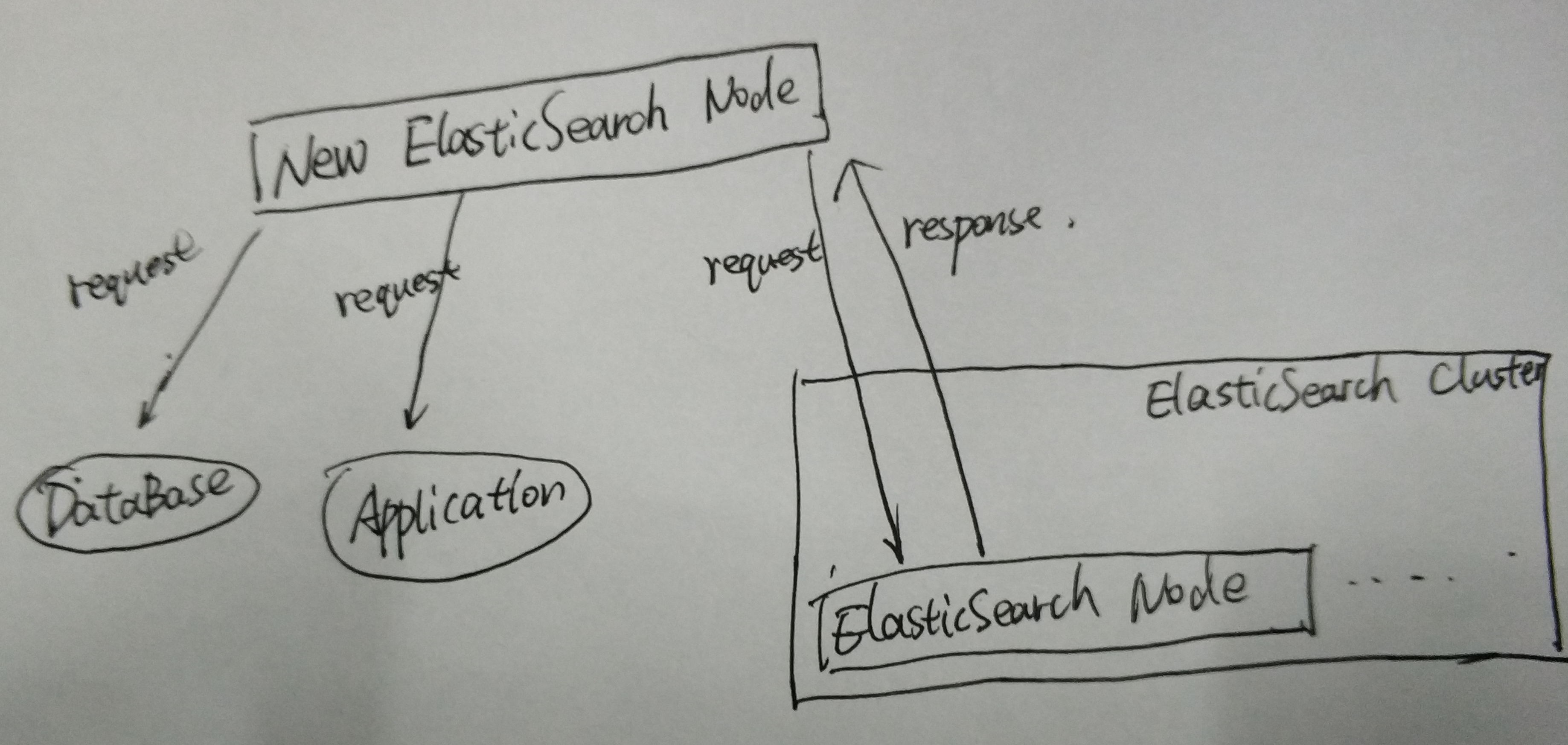
 Why is the quality placed in the middle of the triangle? Because quality is the result of the balance of the other three factors. For example, the scope is not reduced, the cost is not increased, and you want to save time and take shortcuts. Obviously, it will affect the quality. This quality is not only the product quality, but also the quality of the structure and the quality of the code. The overdraft of our quality is a kind of debt, and technical debt is the overdraft of the quality of the architecture and the quality of the code in the software project. Therefore, the technical debt is interest-bearing, and the interest on the debt is the additional time cost when the software is newly modified. Of course, technical debt is not necessarily bad. In software projects, technical debts are often deliberately owed, the development speed of short-term development is increased, software is quickly launched, and the market is seized. There are also models like rapid development, which owe technical debt. Quick verification of the method, even if the verification is not feasible, the technical debt does not need to be repaid.

Then we transfer the problem to elasticsearch. First, let's take a look at the basic content of elasticsearch.

 Elasticsearch is an open source, highly scalable distributed full-text search engine that can store and retrieve data in near real-time. It is very scalable and can be extended to hundreds of servers to process PB-level data. Elasticsearch also uses Java to develop and use Lucene as its core to implement all indexing and search functions, but its purpose is to hide the complexity of Lucene through a simple RESTful API, making full-text search simple.

ES mainly solves three problems: 1) Retrieving relevant data; 2) Returning statistical results; 3) Retrieving faster.

When ElasticSearch's node is started, it uses multicast (or unicast if the user changes the configuration) to find and connect to other nodes in the cluster. This process is shown below:



ES has five core concepts, they are:

 1) Cluster: ES can be used as a standalone single search server. However, to handle large data sets and achieve fault tolerance and high availability, ES can run on many servers that work together. The collection of these servers is called a cluster.

 2) Node: Each server forming a cluster is called a node.

 3) Shard: When there are a large number of documents, a node may not be enough due to memory limitations, insufficient disk processing capabilities, and insufficient response to client requests. In this case, the data can be divided into smaller pieces. Each shard is placed on a different server. When the index you query is distributed across multiple shards, the ES sends the query to each relevant shard and combines the results, and the application does not know the existence of the shard.

4) Replia: To improve query throughput or achieve high availability, you can use a fragmented copy. A copy is an exact copy of a slice, and each slice can have zero or more copies. There can be many identical shards in the ES, one of which is selected to change the indexing operation. This special shard is called the main shard.

 5) Full-text search. Full-text search is to index an article, which can be searched by keyword, similar to the like statement in mysql. Full-text indexing is to segment the content according to the meaning of the word, and then create an index separately.

In the development of Elasticsearch, many technical debts were inevitably generated. Technical debt needs to be managed and maintained at a reasonable level to ensure that developers can continue to contribute to the project. Solving, avoiding technical debt can maintain alternative maintenance costs, improve prevention and prevent accidental accidents. In this section, the technical debt that exists in the Arduino project will be analyzed.

SonarQube is a platform for continuous software quality monitoring. This platform is able to analyse a large software project like Gradle in a matter of minutes. It provides the user with crucial insights about bugs, security and technical debt within the project. We put the project on SonarQube for analysis and got the following results: SonarQube reported a large number of errors (162), vulnerabilities (263) and code smells (8400), the number of duplicate codes was 1.6% of the code, and the repeating block 493. SonarQube identified 280 days of technical debt, meaning that fixing all of the mentioned violations would take 280 working days, approximately 9 months. But after inquiry, we found that some of the debts monitored by SonarQube are actually the code style choices that the Elasticsearch team made during its development cycle, and therefore do not constitute actual technical debt. or example, the biggest contributor to violations in Elasticsearch is the Boolean comparison, which results in 1380 code smells. The reason for the detection is that the developer compares all Boolean comparisons to the format a==true instead of if(a). The latter is a format we usually use, because it is more concise. But using the former makes the code more readable, which is a reasonable choice for the Elasticsearch development team.

After browsing the code we found that there is a lot of extra code in Elasticsearch. SonarQube found 117 unused method parameters and 237 unused import statements. This extra code makes the code less readable. But cleaning up such code doesn't take much time. The Elasticsearch developers have a tendency to name String literals with an all-caps name, such as String FOO. These literals mostly represent the name of a certain field in a class, e.g. String FOO = "foo". These literals are used in so-called xContent methods, which deal with different types (X types) of content that needs to be written to an output stream. Instead of hardcoding the actual strings in such an xContent method, the all-caps String definitions are used. This causes a number of classes to contain both the normal field foo, which is named in lower-case, as well as the xContent field FOO, named in upper-case. Although this is a deliberate choice made by the developers, SonarQube flags this as a blocker type issue, meaning an immediate fix is preferred.

+