# 研究背景

因为大量数据存储在集群中，我们能通过Hive等Hadoop工具能够很容易的获取存储在HDFS中的数据，当用户有需求，需要使用存储在HDFS中的数据，我们需要提供便于用户访问集群数据的途径。

传统的分析统计工具，比如SAS，SPSS，MATLAB，原生不能很好的支持直接访问HDFS或者直接从Hive获取数据，但是这些统计分析软件普遍能够很好的支持传统的关系型数据库的调用访问，因此我们通过研究设计一套以MySQL为数据交换媒介的系统为用户提供更加便利的数据访问。

# 研究方向

Sqoop是一款开源的工具，主要用于在Hadoop(Hive)与传统的数据库(MySQL、Postgresql等)间进行数据的传递，可以将一个关系型数据库（例如 ： MySQL ,Oracle ,Postgresql等）中的数据导进到Hadoop的HDFS中，也可以将HDFS的数据导进到关系型数据库中。

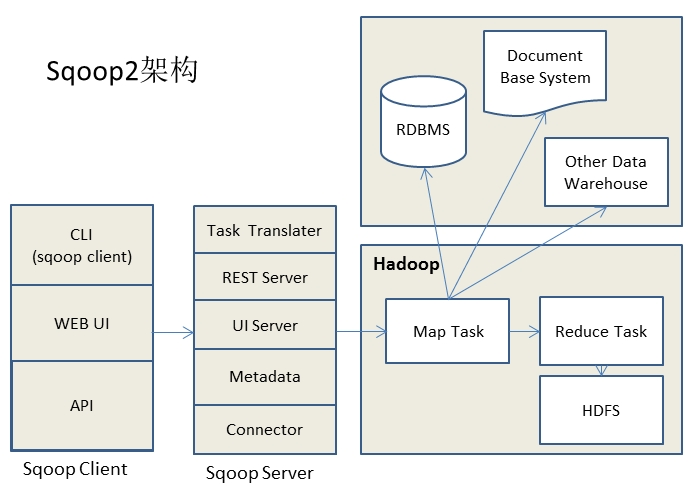
Sqoop中一大亮点就是可以通过Hadoop的MapReduce把数据从关系型数据库中导入数据到HDFS。Sqoop架构非常简单，其整合了Hive、Hbase和Oozie，通过map-reduce任务来传输数据，从而提供并发特性和容错。Sqoop主要通过JDBC和关系数据库进行交互。理论上支持JDBC的数据库都可以使用Sqoop和HDFS进行数据交互。

# 研究内容

## Sqoop1和Sqoop2的区别

架构上，Sqoop1使用MapOnly作业进行Hadoop（HDFS/HBase/Hive）同关系数据库进行数据的导入导出，用户使用命令行方式与之交互，数据传输和数据格式紧密耦合；易用性欠佳，Connector数据格式支持有限，安全性不好，对Connector的限制过死。Sqoop2则建立了集中化的服务，负责管理完整的MapReduce作业，提供多种用户交互方式（CLI/WebUI/RESTAPI），具有规范化的Connector。

## Sqoop2基本架构图



## Sqoop2 Client Java API 接口调用

接口实现的主要功能分为以下方面：

### 根据Connector建立Connection

示例代码如下：

**public** **static** String *url* = "http://10.1.65.10:12000/sqoop/";

**public** **static** SqoopClient *client* = **new** SqoopClient(*url*);

**public** **static void** connection(){

//create the new connection

MConnection newCon = *client*.newConnection(1);

//Get connection and framework forms. Set name for connection

MConnectionForms conForms = newCon.getConnectorPart();

MConnectionForms frameworkForms = newCon.getFrameworkPart();

newCon.setName("JAVA\_API\_TEST");

//Set connection forms values

conForms.getStringInput("connection.connectionString").setValue("jdbc:mysql://10.1.30.3:3306/Sqoop\_test");

conForms.getStringInput("connection.jdbcDriver").setValue("com.mysql.jdbc.Driver");

conForms.getStringInput("connection.username").setValue("root");

conForms.getStringInput("connection.password").setValue("root");

frameworkForms.getIntegerInput("security.maxConnections").setValue(0);

Status status = *client*.createConnection(newCon);

**if**(status.canProceed()) {

System.***out***.println("Created. New Connection ID : " +newCon.getPersistenceId());

} **else** {

System.***out***.println("Check for status and forms error ");

}

}

### 获取存在的Connection，并且能够做删除、更新操作

示例代码如下：

**public** **static** String *url* = "http://10.1.65.10:12000/sqoop/";

**public** **static** SqoopClient *client* = **new** SqoopClient(*url*);

**public** **static void** get\_connection( Long cid ){

//get the exist connection which is match with cid

MConnection newCon = *client*.getConnection(cid);

//Get connection and framework forms. Set name for connection

MConnectionForms conForms = newCon.getConnectorPart();

MConnectionForms frameworkForms = newCon.getFrameworkPart();

newCon.setName("JAVA\_API\_TEST");

//Set connection forms values

conForms.getStringInput("connection.connectionString").setValue("jdbc:mysql://10.1.30.3:3306/Sqoop\_test");

conForms.getStringInput("connection.jdbcDriver").setValue("com.mysql.jdbc.Driver");

conForms.getStringInput("connection.username").setValue("root");

conForms.getStringInput("connection.password").setValue("root");

frameworkForms.getIntegerInput("security.maxConnections").setValue(0);

Status status = *client*.updateConnection(newCon);

**if**(status.canProceed()) {

System.***out***.println("Created. New Connection ID : " +newCon.getPersistenceId());

} **else** {

System.***out***.println("Check for status and forms error ");

}

}

### 根据Connection的ID建立对应的JOB

示例代码如下：

**public** **static** String *url* = "http://10.1.65.10:12000/sqoop/";

**public** **static** SqoopClient *client* = **new** SqoopClient(*url*);

**public** **static** Long job( Long cid ){

MJobnewjob=*client*.newJob(cid,org.apache.sqoop.model.MJob.Type.***EXPORT***);

MJobForms connectorForm = newjob.getConnectorPart();

MJobForms frameworkForm = newjob.getFrameworkPart();

newjob.setName("export\_test");

//Database configuration

//connectorForm.getStringInput("table.schemaName").setValue("");

connectorForm.getStringInput("table.tableName").setValue("export\_data\_chinese");

//connectorForm.getStringInput("table.sql").setValue("select id,name from table where ${CONDITIONS}");

connectorForm.getStringInput("table.columns").setValue("number,chinese,double\_number");

//Input configurations

frameworkForm.getStringInput("input.inputDirectory").setValue("/user/zpl\_test\_export/test\_timestamp.dat");

//Job resources

//frameworkForm.getIntegerInput("throttling.extractors").setValue(1);

//frameworkForm.getIntegerInput("throttling.loaders").setValue(1);

Status status = *client*.createJob(newjob);

**if**(status.canProceed()) {

System.***out***.println("New Job ID: "+ newjob.getPersistenceId());

} **else** {

System.***out***.println("Check for status and forms error ");

}

//Print errors or warnings

*printMessage*(newjob.getConnectorPart().getForms());

*printMessage*(newjob.getFrameworkPart().getForms());

**return** newjob.getPersistenceId();

}

### 获取已经存在的JOB，并且能够做删除、更新操作

示例代码如下：

**public** **static** String *url* = "http://10.1.65.10:12000/sqoop/";

**public** **static** SqoopClient *client* = **new** SqoopClient(*url*);

j

**public** **static** Long job( Long jid ){

MJob newjob = *client*.getJob(jid);

MJobForms connectorForm = newjob.getConnectorPart();

MJobForms frameworkForm = newjob.getFrameworkPart();

newjob.setName("export\_test");

//Database configuration

//connectorForm.getStringInput("table.schemaName").setValue("");

//Input either table name or sql

connectorForm.getStringInput("table.tableName").setValue("export\_data\_chinese");

//connectorForm.getStringInput("table.sql").setValue("select id,name from table where ${CONDITIONS}");

connectorForm.getStringInput("table.columns").setValue("number,chinese,double\_number");

//Input configurations

frameworkForm.getStringInput("input.inputDirectory").setValue("/user/zpl\_test\_export/test\_timestamp.dat");

//Job resources

//frameworkForm.getIntegerInput("throttling.extractors").setValue(1);

//frameworkForm.getIntegerInput("throttling.loaders").setValue(1);

//Status status\_2 = client.createJob(newjob);

Status status\_2 = *client*.updateJob(newjob);

**if**(status\_2.canProceed()) {

System.***out***.println("New Job ID: "+ newjob.getPersistenceId());

} **else** {

System.***out***.println("Check for status and forms error ");

}

//Print errors or warnings

*printMessage*(newjob.getConnectorPart().getForms());

*printMessage*(newjob.getFrameworkPart().getForms());

**return** newjob.getPersistenceId();

}

### 提交运行JOB

示例代码如下：

**public** **static** String *url* = "http://10.1.65.10:12000/sqoop/";

**public** **static** SqoopClient *client* = **new** SqoopClient(*url*);

**public** **static** **void** submission(Long jid){

MSubmission submission = *client*.startSubmission(jid);

System.***out***.println("Status : " + submission.getStatus());

**if**(submission.getStatus().isRunning() && submission.getProgress() != -1) {

System.***out***.println("Progress : " + String.*format*("%.2f %%", submission.getProgress() \* 100));

}

System.***out***.println("Hadoop job id :" + sub mission.getExternalId());

System.***out***.println("Job link : " + submission.getExternalLink());

Counters counters = submission.getCounters();

**if**(counters != **null**) {

System.***out***.println("Counters:");

**for**(CounterGroup group : counters) {

System.***out***.print("\t");

System.***out***.println(group.getName());

**for**(Counter counter : group) {

System.***out***.print("\t\t");

System.***out***.print(counter.getName());

System.***out***.print(": ");

System.***out***.println(counter.getValue());

}

}

}

**if**(submission.getExceptionInfo() != **null**) {

System.***out***.println("Exception info : " +submission.getExceptionInfo());

}

}

### 监控JOB的状态

示例代码如下：

**public** **static** String *url* = "http://10.1.65.10:12000/sqoop/";

**public** **static** SqoopClient *client* = **new** SqoopClient(*url*);

**public** **static** **void** checkstatus(Long jid){

MSubmission submission = *client*.getSubmissionStatus(jid);

**if**(submission.getStatus().isRunning() && submission.getProgress() != -1) {

System.***out***.println("Progress : " + String.*format*("%.2f %%", submission.getProgress() \* 100));

}

}

### 终止正在运行的JOB

示例代码如下：

**public** **static** String *url* = "http://10.1.65.10:12000/sqoop/";

**public** **static** SqoopClient *client* = **new** SqoopClient(*url*);

**public** **static** **void** stopmission( Long jid ){

*client*.stopSubmission(jid);

}

### 获取已有的Connection 的信息

示例代码如下：

**public** **static** String *url* = "http://10.1.65.10:12000/sqoop/";

**public** **static** SqoopClient *client* = **new** SqoopClient(*url*);

**public** **static** **void** Connection\_describe(Long cid){

//While printing connection forms, pass connector id to getResourceBundle(cid).

*describe*(*client*.getConnection(cid).getConnectorPart().getForms(), *client*.getResourceBundle(1));

*describe*(*client*.getConnection(cid).getFrameworkPart().getForms(), *client*.getFrameworkResourceBundle());

}

**private** **static** **void** describe(List<MForm> forms, ResourceBundle resource) {

**for** (MForm mf : forms) {

System.***out***.println(resource.getString(mf.getLabelKey())+":");

List<MInput<?>> mis = mf.getInputs();

**for** (MInput mi : mis) {

System.***out***.println(resource.getString(mi.getLabelKey()) + " : " + mi.getValue());

}

System.***out***.println();

}

}

### 获取已有的Job的信息

**public** **static** String *url* = "http://10.1.65.10:12000/sqoop/";

**public** **static** SqoopClient *client* = **new** SqoopClient(*url*);

**public** **static** **void** Job\_describe(Long jid){

//While printing connection forms, pass connector id to getResourceBundle(cid).

*describe*(*client*.getJob(jid).getConnectorPart().getForms(), *client*.getResourceBundle(1));

*describe*(*client*.getJob(jid).getFrameworkPart().getForms(), *client*.getFrameworkResourceBundle());

}

**private** **static** **void** describe(List<MForm> forms, ResourceBundle resource) {

**for** (MForm mf : forms) {

System.***out***.println(resource.getString(mf.getLabelKey())+":");

List<MInput<?>> mis = mf.getInputs();

**for** (MInput mi : mis) {

System.***out***.println(resource.getString(mi.getLabelKey()) + " : " + mi.getValue());

}

System.***out***.println();

}

}

## Sqoop2测试

### 测试遇到的问题

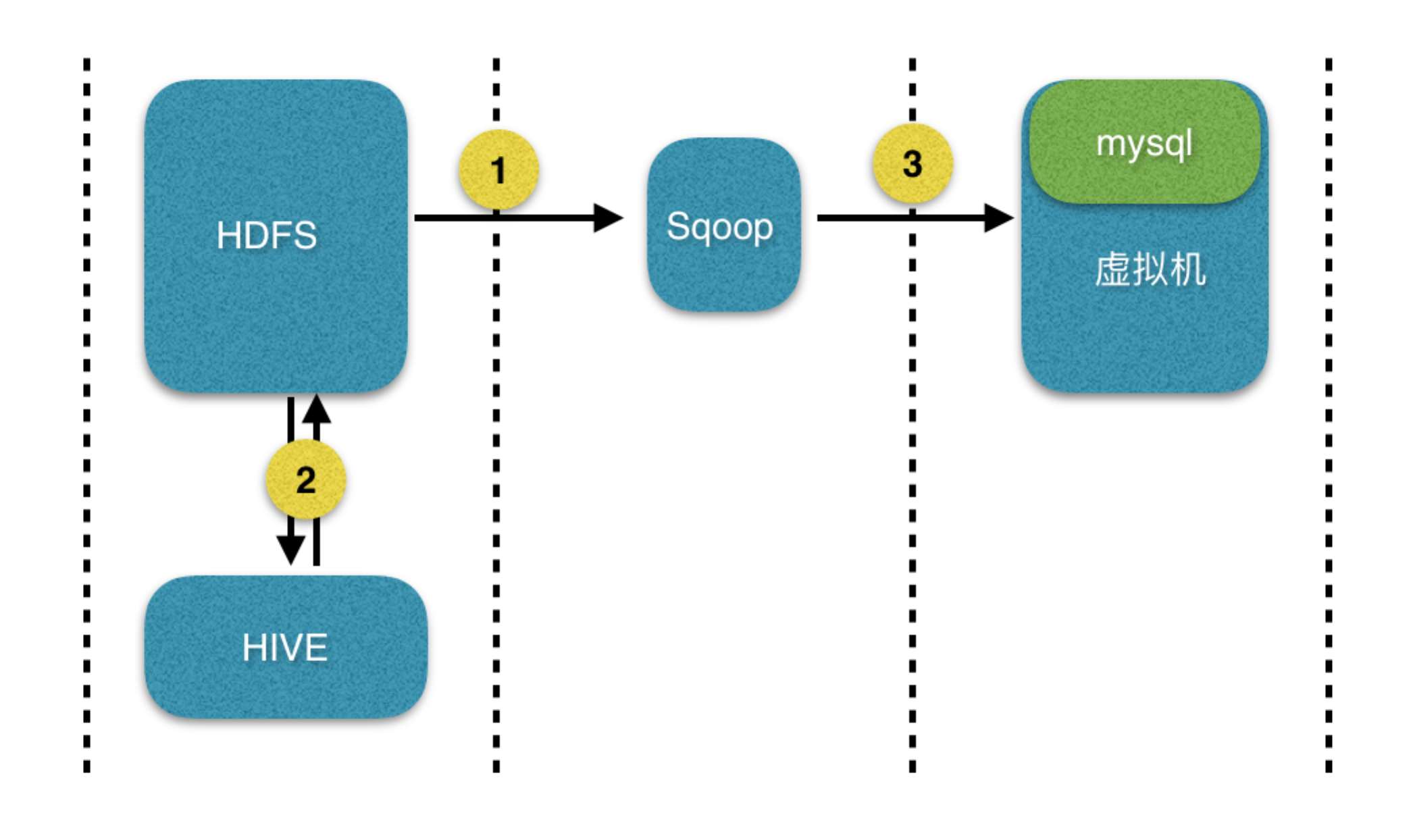
* + - 1. 从HDFS中导出的数据在导入MYSQL中时float类型的数据发生了精度损失，由此我们需要注意，需要定义规整的数据精度，编码规范，否则在数据导出的过程中会遇到很多精度，编码的问题。
      2. 数据导入导出时，浮点数据的损耗，测试寻找数据损耗的原因，主要原因是SQL server 和MySQL之间对于默认的float类型的精度不一样导致精度损失。
      3. 当非数值型数据导入时必须要用引号括起来，否则Sqoop会在数据预处理的时候进行转型，会导致报错。
      4. 中文的时候测试遇到了数据库中文导入乱码的问题，最后通过调整数据库的存储编码解决。
      5. 测试时间类型数据导出时会遇到的问题，Datetime类型数据导出的时候Sqoop在数据预处理的时候会破坏Datetime类型的格式，删掉中间的空格，导致导入MySQL引发错误，解决方案可能要修改Sqoop源码。所有的数据除了数字类型都要用引号包起来，否则Sqoop会把数据类型强制装换为long类型，从而引发异常。
      6. 测试大量数据发现集群导出数据量和HDFS中的数据量有出入，但是多次测试后问题又没有出现同样问题。

### 测试性能数据

|  |  |  |
| --- | --- | --- |
| 数据量 | 平均花费时间 | 备注 |
| 10000000 | 55mins |  |
| 1000000 | 6mins |  |

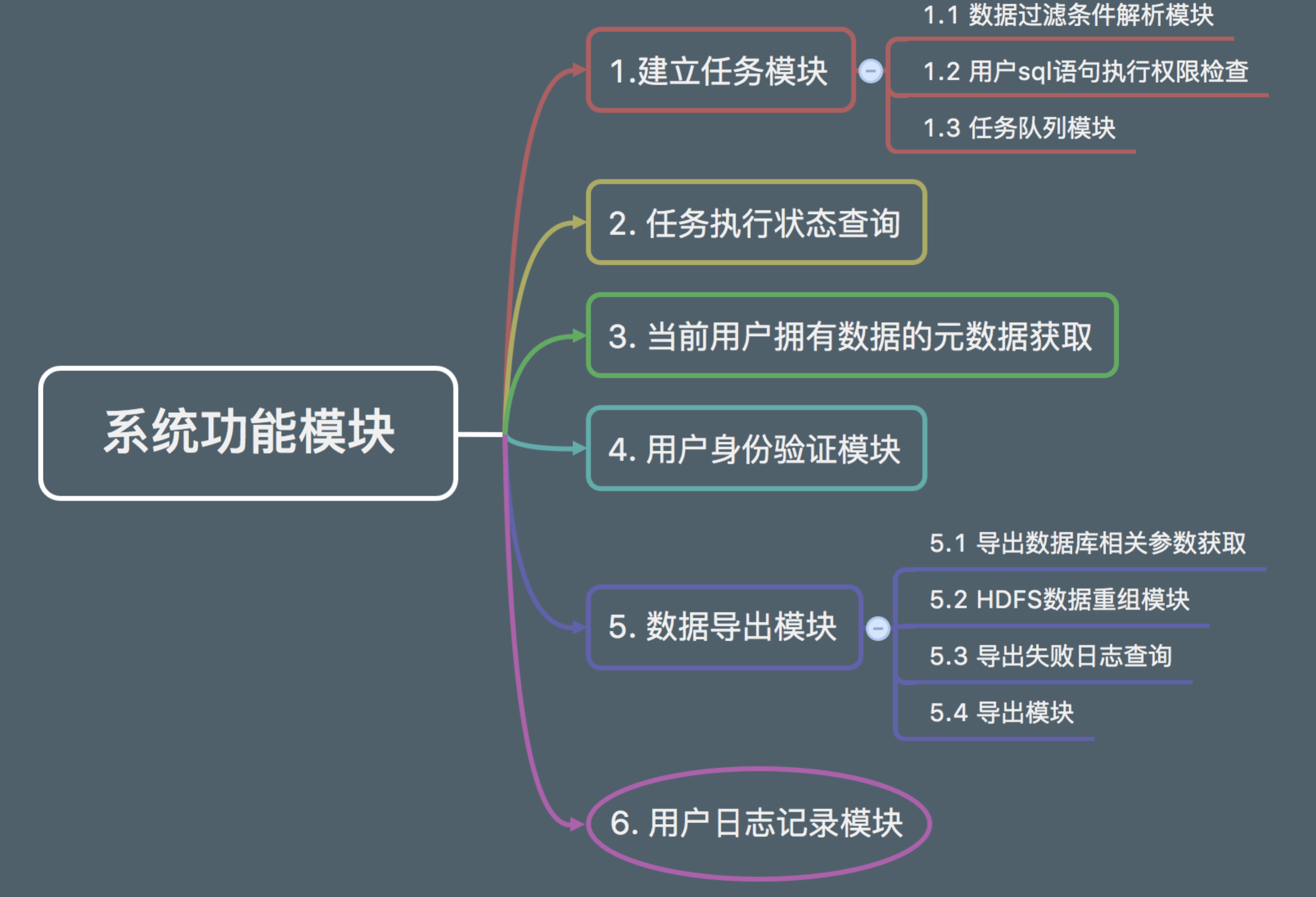
注意：这部分测试结果可能不准确，下面会调整一下测试的环境以及策略做进一步的性能测试

## 系统设计思路



1. 系统通过Sqoop从HDFS中直接导入数据至MySQL。
2. 当用户在建立导出任务时使用了条件查询，系统通过用户录入查询过滤条件，通过Hive在HDFS中生成用户需求临时数据，然后在通过第一步导入用户MySQL。
3. Sqoop将用户建立的临时生成的数据插入对应的MySQL中。

# 系统功能模块设计



### 建立任务模块

#### 数据过滤条件解析模块

* 系统要求用户熟悉基本的SQL语句，用户通过模块3提供的用户拥有的数据元数据填写提交给hive的SQL语句，系统通过把用户提供的SQL语句提交给hive获取生成的数据，从而在HDFS生成要导出的临时数据。
* 关于实现方面，我们通过在后台调用Hive API接口执行SQL语句实现生成用户需求中间数据。

#### 用户SQL语句执行权限检查

* 每次用户提交给系统的SQL语句首先检查用户是否有权限执行当前SQL语句，要检查用户是否有权限做当前操作，用户是否有权限操作当前表。
* 实现方面，我们通过在后台集中Hive SQL解析器，首先能够检查SQL语句的正确性，从而获得用户发生操作的表名，以及用户即将要进行的行为，然后通过我们后台定义的规则以及获取用户的元数据检查用户是否有权限执行当前操作和是否对当前表有操作权限。

#### 用户任务队列模块

* 用户可能会在同一时间想要获取多份数据，为了避免单用户为系统带来过大的性能损耗，影响整个系统的用户体验，系统应该为用户提供一个可以接受申请的上限，如果超过上限，为用户维护一个任务队列，只有当用户的在执行任务结束，在继续获取等待队列中的任务执行。
* 实现细节方面，在数据库为用户维护一张表，每次调用任务完成后修改相应的导出任务的标志位。当有新的任务进来之后，把任务相应的参数插入数据库中。

### 任务执行状态查询

用户建立导出任务，可能有以下几种状态：

* 等待调度中，即用户一次提交了多个数据申请，但是系统设计了用户能够同时提交的任务的上限，所以此时，任务在用户的任务队列中等待被系统调度
* 执行中，任务已经被任务系统调用，等待系统完成数据的导出
* 失败，任务被调用，但是在执行中发生错误，任务被退出，此处提供失败原因。
* 完成，已经完成的任务列表，显示完成的时间以及导出的数据的基本参数
* 用户空间不足，用户的导出数据大小大于数据库的剩余空间，提供给用户操作选择，1.增加过滤条件，对数据再次进行过滤2.整理自己的数据空间，重新尝试导入3.放弃当前数据

实现方面，我们为每个任务维护一个状态变量用来指定当前任务的实际状态，当状态发生改变时，修改数据库中状态变量的值，从而修改任务状态到它实际的状态。

### 当前用户拥有数据的元数据获取

* 系统为用户提供能够获取的数据资源的元数据，包括数据的数量，种类，数据组织结构，对应字段的含义等。
* 实现方式上，通过获取hive中属于当前用户的数据表的元数据信息，从而获取样例数据以及对应的字段名。

### 用户身份验证模块

* 用户在建立任务前要先登录，登录的过程中包含用户身份的验证，用户拥有的数据的元数据获取，登录日志的录入等模块。
* 具体实现方面，首先我们任务管理平台的用户和数据平台的用户系统是一致的，所以，我们只需要构建登录模块，验证hive的用户中是否存在当前用户，验证密码即可。

### 数据导出模块

#### 导出数据库相关参数获取

* 为了保证数据的安全性，系统指定数据库地址以及用户名，密码，要导用户提供要导出的库名及表名，并且保证已经在数据库建立了与导出数据格式一致的表，系统会在对应的页面检查该表是否已经被建立。
* 实现方式上，后台通过JDBC连接用户数据库，检查用户对应的表是否存在。

#### HDFS数据重组模块

* 用户提交获取数据的SQL语句，系统验证SQL语句的权限之后，需要提交SQL语句到Hive，从而生成中间临时数据，以便Sqoop能直接导出数据到MySQL。这个模块要提供任务执行的状态给导出模块，任务执行进度以及任务执行失败原因。

#### 导出失败原因查询

提供详细的任务执行失败原因，任务执行失败的原因提取出来反馈至任务管理平台。

#### 导出模块

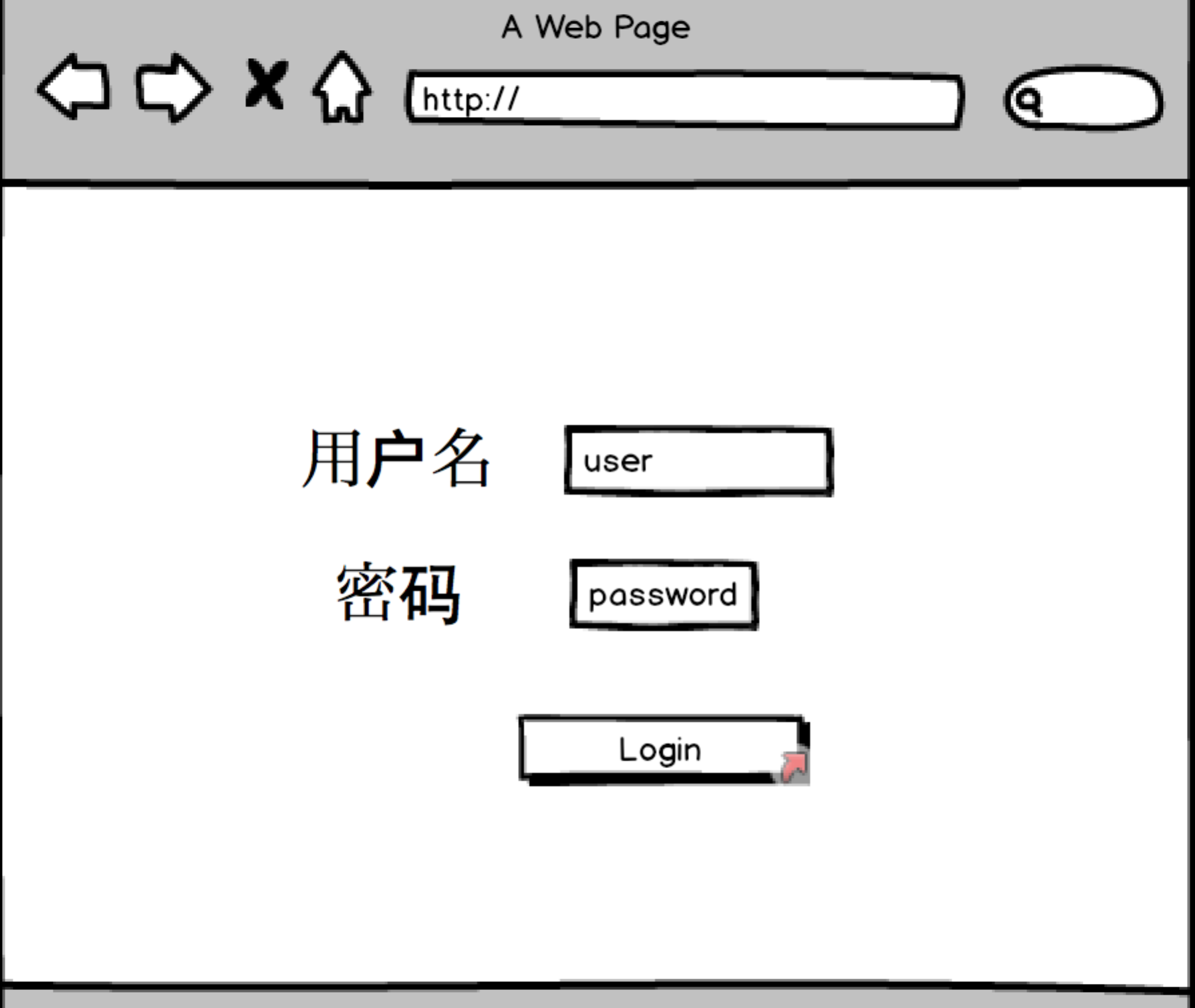
当Hive生成了用户所需的临时数据，导出模块调用Sqoop API从HDFS把数据导入与用户对应的MySQL数据库，并且记录相关日志。

### 用户日志记录模块

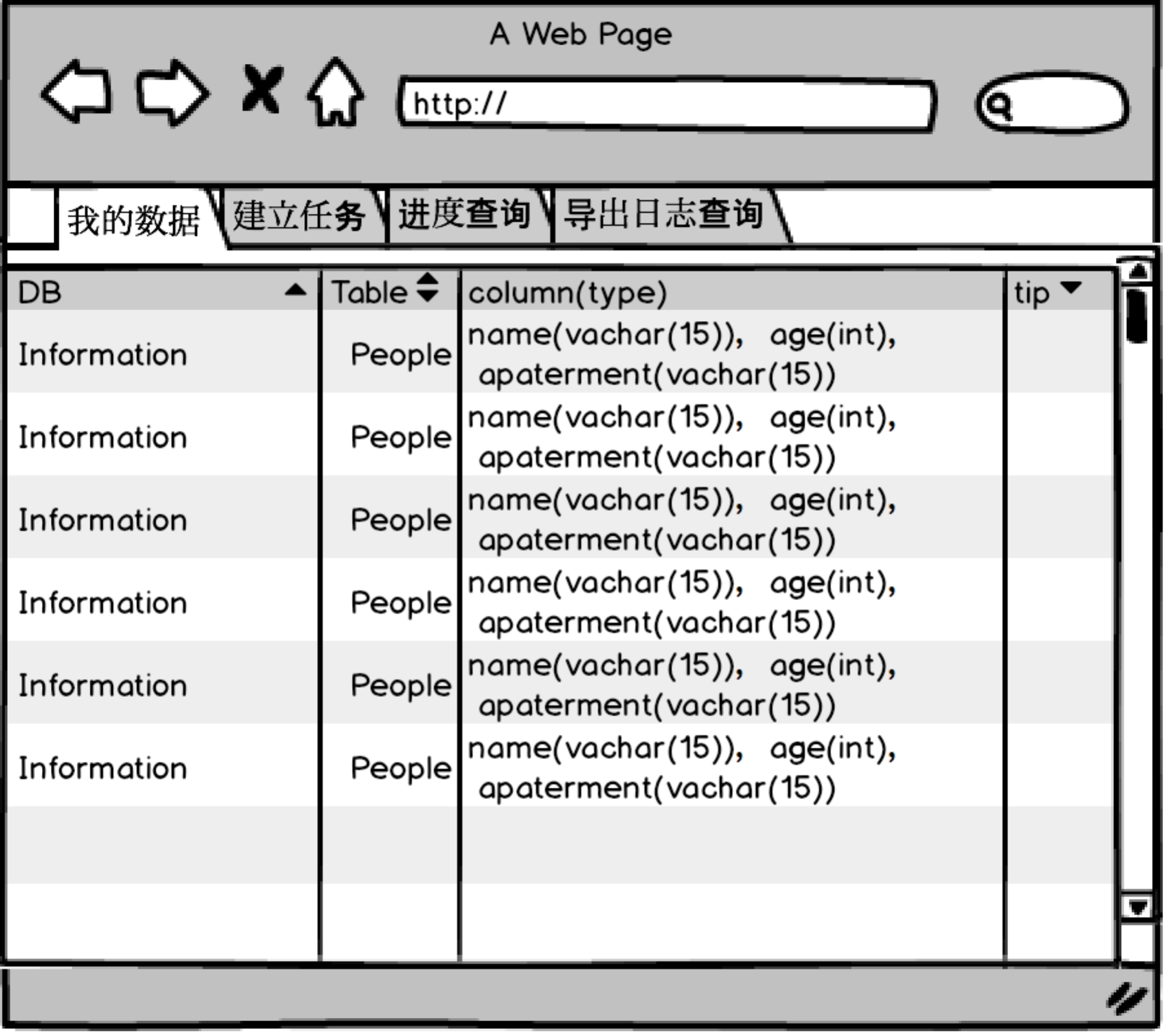
用户日志记录模块，记录用户的登录时间，用户的操作，提交任务的时间，提交的任务的详情，任务结束的时间。

# 原型设计

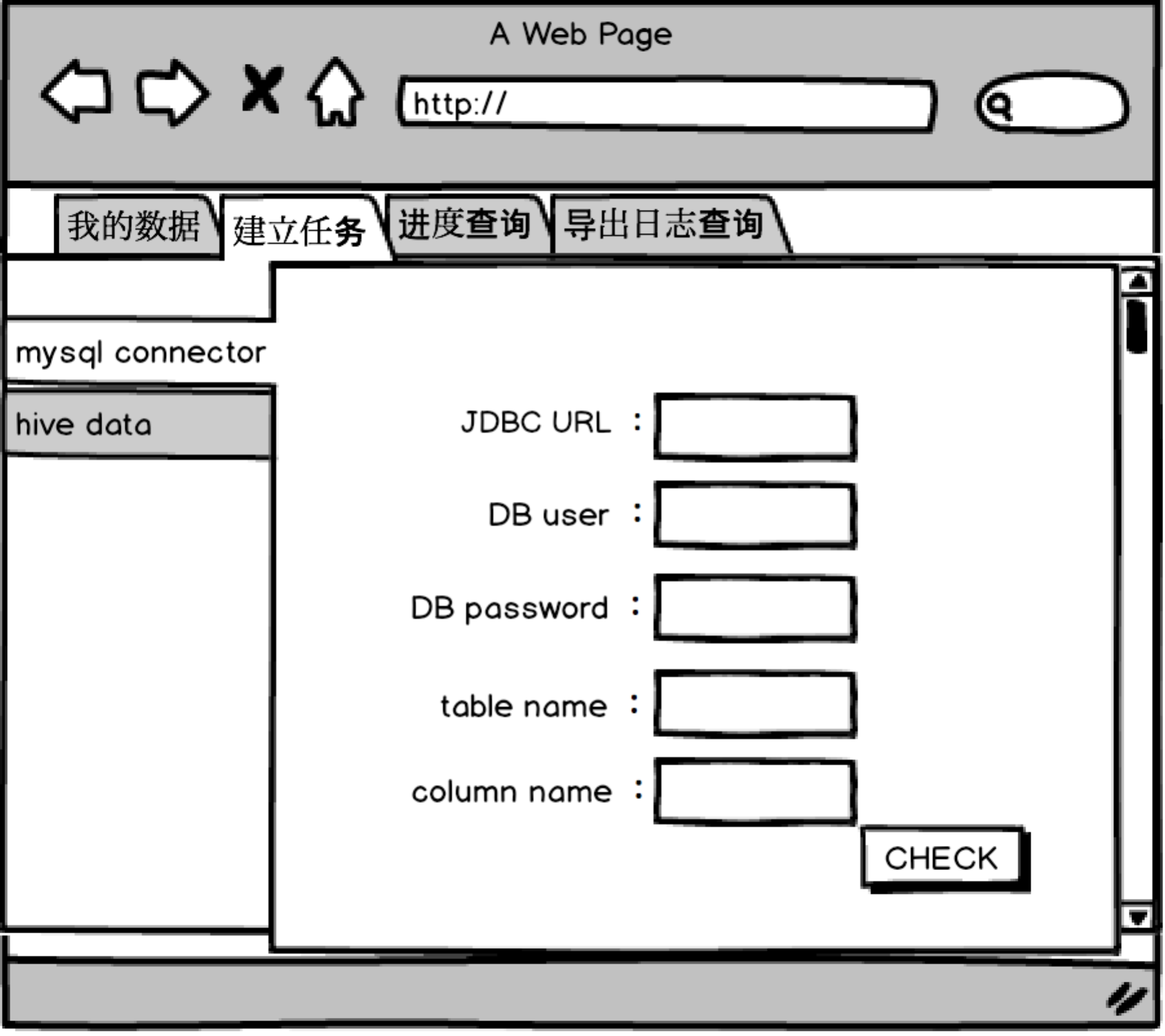
* 登录



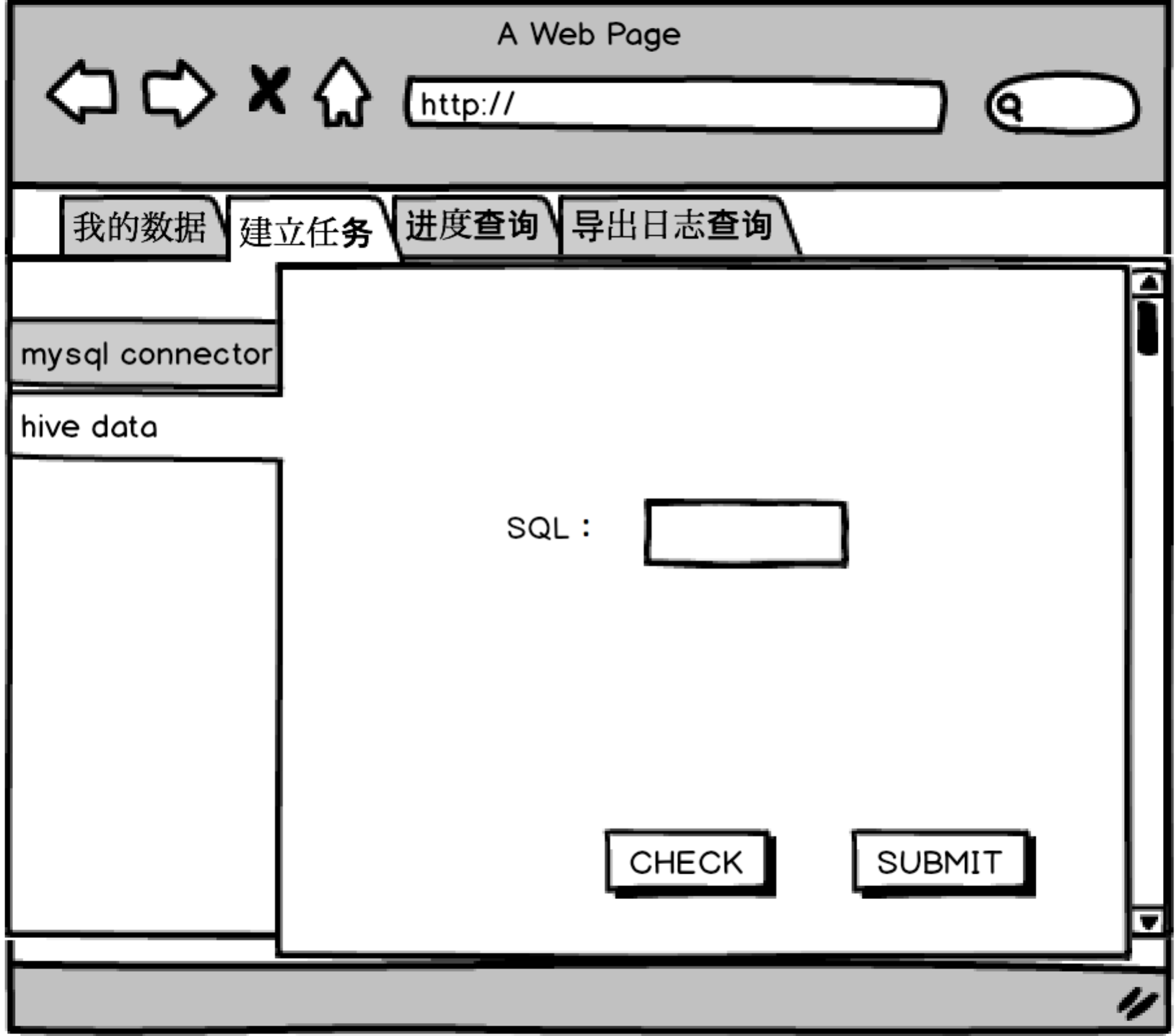
* HOME



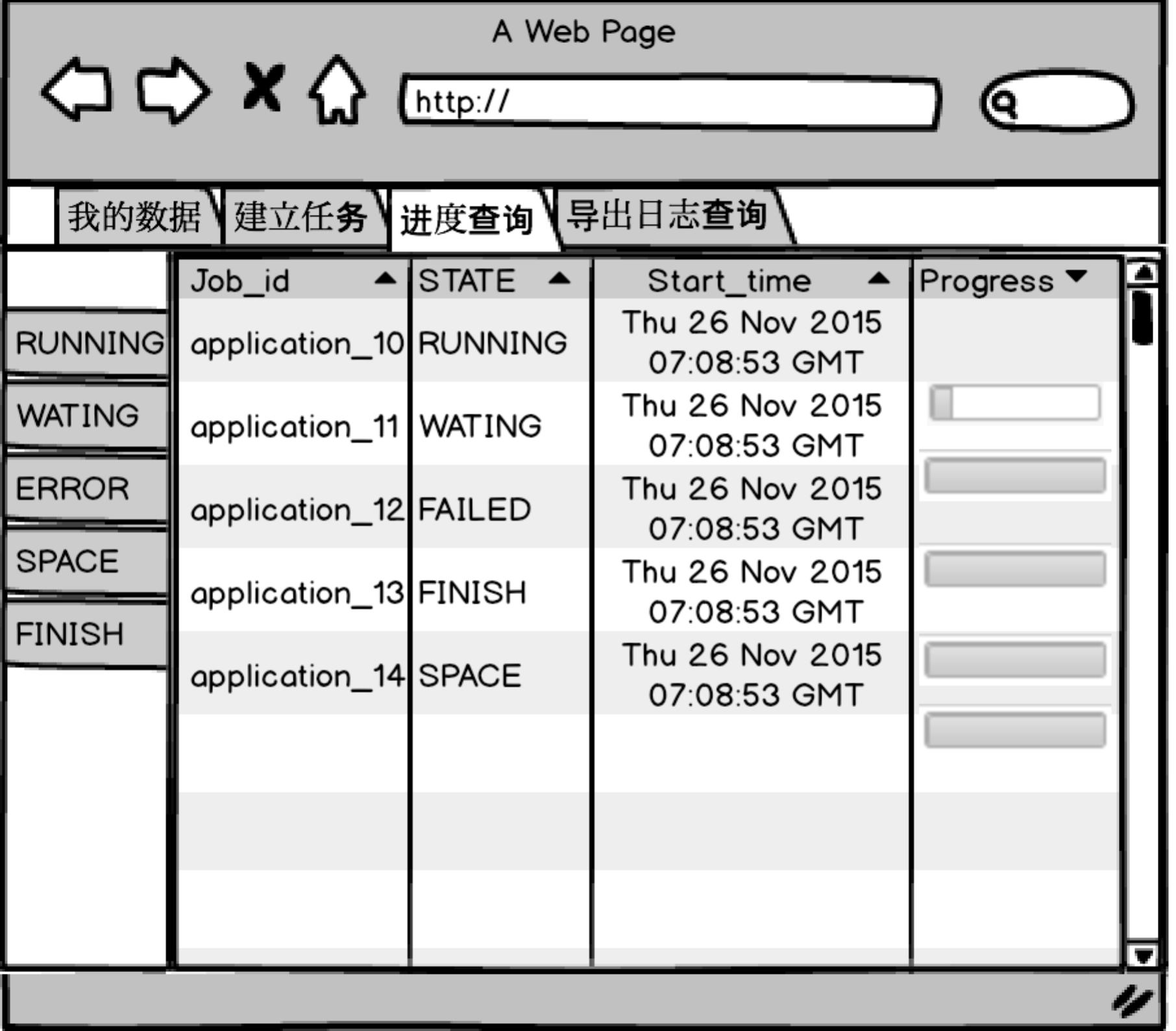
* MySQL Connector



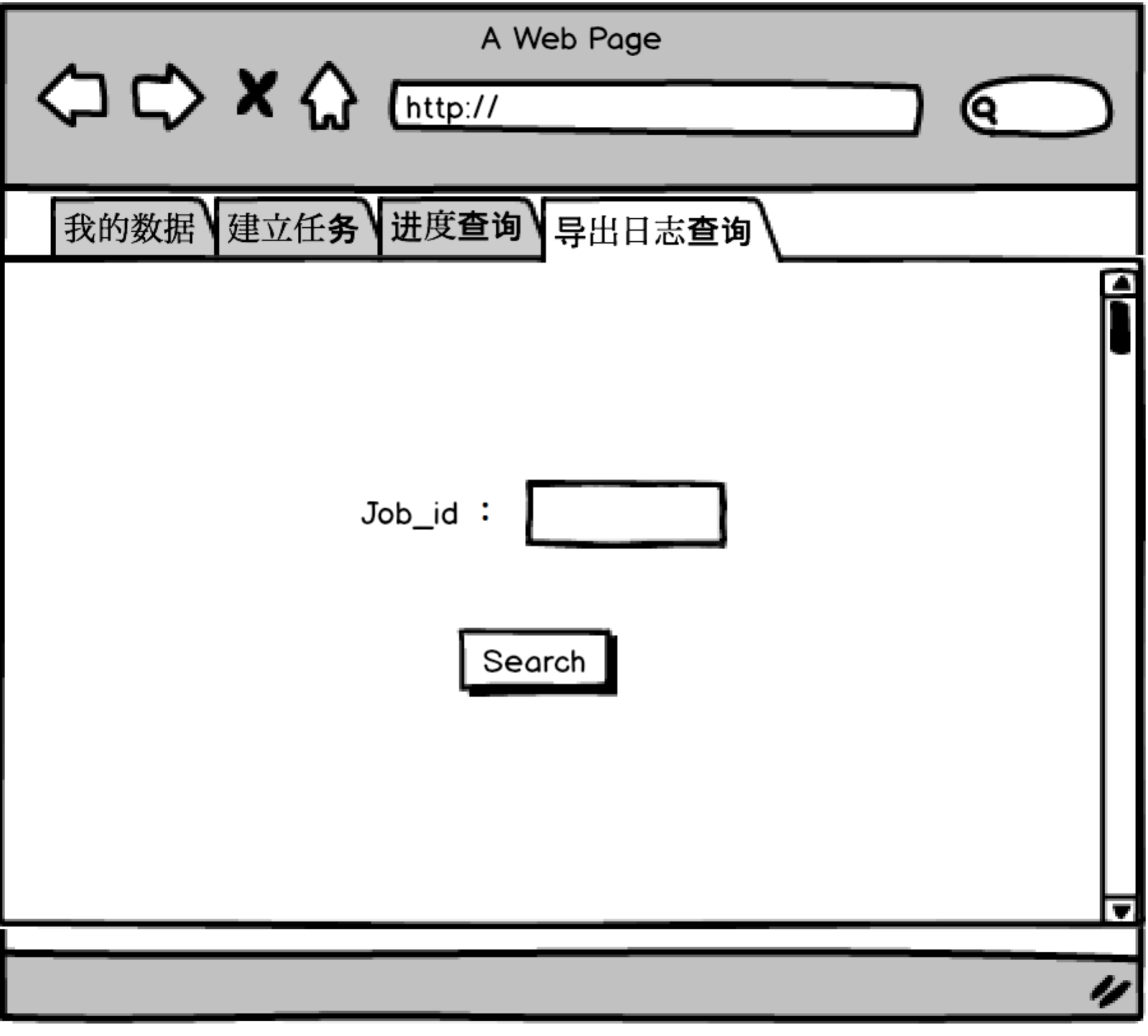
* Hive Data



* Job



* 日志查询



# 问题汇总

1. **用户登录自己的虚拟机之后，启动MySQL数据库，任务平台才可以完成数据导入，当用户下线之后，怎么保证数据库在线，且物理机不会因为常驻的虚机太多而过载？**

当用户维护自己的虚拟机内部数据库时，发生任务提交行为之后要一直保持虚机数据库在线以便数据任务平台能将用户建立的数据导入对应的数据库，但是这样的话系统的资源浪费又很严重，所以，考虑使用容器解决数据库常驻问题，在Docker上部署与用户绑定的数据库，这样就不需要庞大的虚拟机保持在线，只在需要使用对应用户的数据库时启动对应的Docker服务即可。

1. **数据库对用户开辟的空间大小决定这用户可以导出的数据大小，也引申出当Sqoop导出的数据大小大于用户数据库大小时，如何处理？**

第一，每次提交任务请求时获取用户数据库的剩余空间大小；

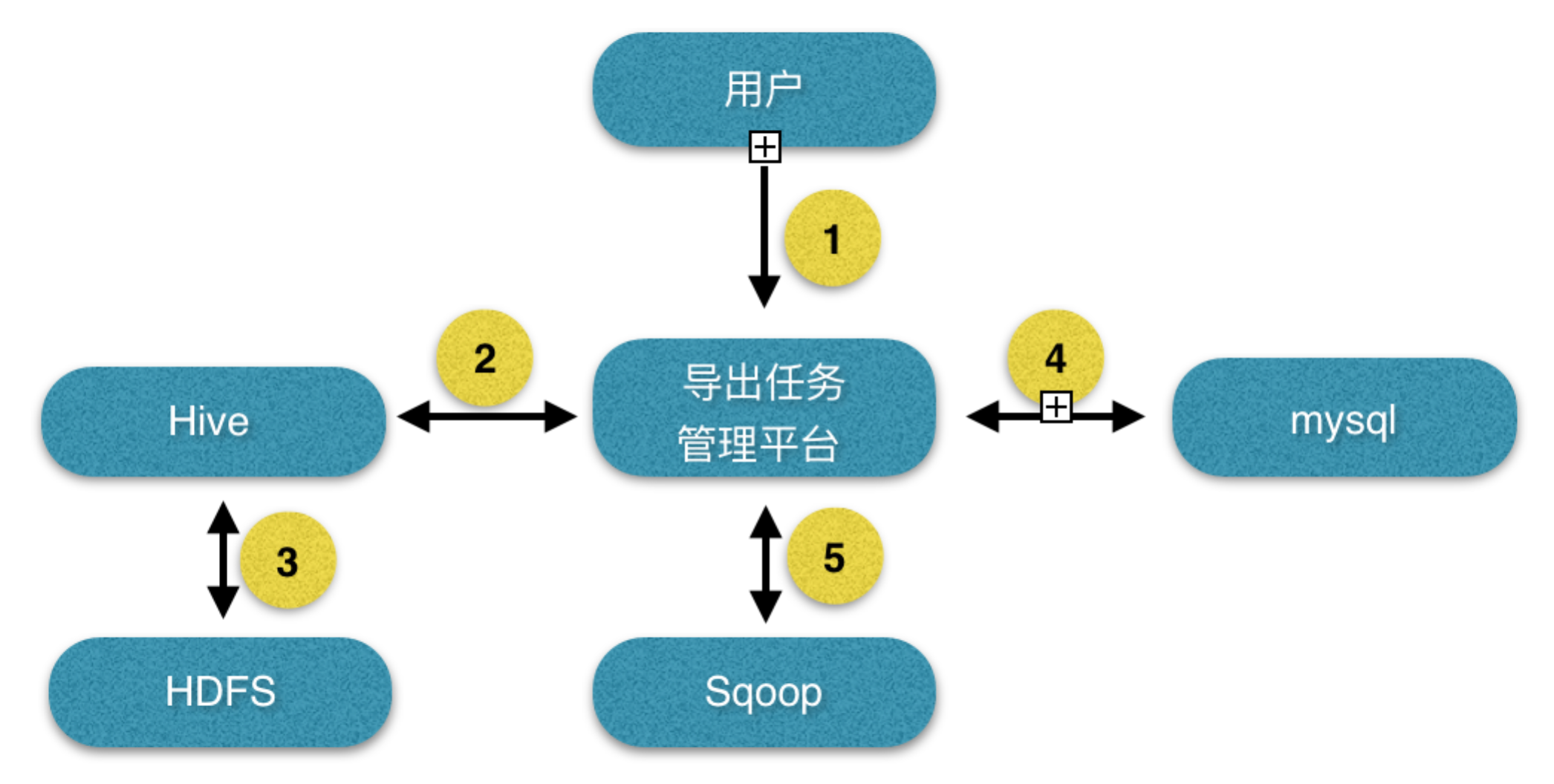
第二，Hive获取中间数据之后，比较中间数据的大小和用户剩余空间的大小，如果数据库空间不足，则返回状态给用户提示空间不足。保留生成的中间数据，等待选择操作（见四.2）

1. **Hive获取的导入数据的大小不能代表实际占MySQL空间的大小，怎么解决？**

对于文本数据，导入数据库做大量测试，得到一个比较稳定的比例，即文本大小和其所占数据库空间大小的比例，每次生成导入数据后估算其占数据库空间大小，然后和实际剩余空间大小作比较。

1. **怎么保证系统的容错性，几个模块之间的粒度非常大，中间任何一个步骤出现错误，怎么样保证用户能够获取到反馈，并且做出判断？**

我们通过下图来了解如何让整个系统保持高度的容错性：



* 1. 用户通过登录任务管理平台来确保用户的身份被验证，在这里我们会记录用户的操作行为日志。
  2. 管理平台通过调用Hive来准备用户需要的数据，在这个过程中，如果任务执行失败，返回执行状态，给出失败原因，反馈至任务管理平台。
  3. Hive准备需求数据，生成的数据保存在HDFS的临时数据文件中。
  4. 当第二步，第三步执行没有问题的时候先执行第四步，即获取MySQL剩余空间的大小，然后和生成对的中间数据大小比较，如果空间不允许，则反馈给任务管理平台，修改任务状态，并提供操作选项（见四.2）
  5. 当前面的操作都没有问题的时候，执行第五步，通过Sqoop导入数据到MySQL，如果执行失败，反馈原因至数据管理平台。

通过上面的结构设计，任务的状态转移每一步都通过任务管理平台决定，那个模块出现问题都会反馈给管理平台在做处理，能够使用户很快定位任务执行失败的原因。