**金融数据跨业共享标准自适应示范平台详细设计**

# 功能模块设计

## 1.1 数据接入模块

****1、类图****

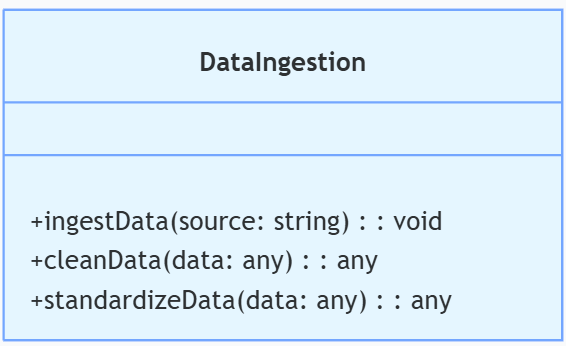


图1：数据接入类图

代码：

classDiagram

class DataIngestion {

+ingestData(source: string): void

+cleanData(data: any): any

+standardizeData(data: any): any

}

类图说明：

DataIngestion 类负责数据接入、清洗和标准化。

ingestData 方法用于从各种数据源获取数据。

cleanData 方法对获取的数据进行清洗操作。

standardizeData 方法将清洗后的数据进行标准化处理，使其格式统一且语义一致。

****2、时序图****

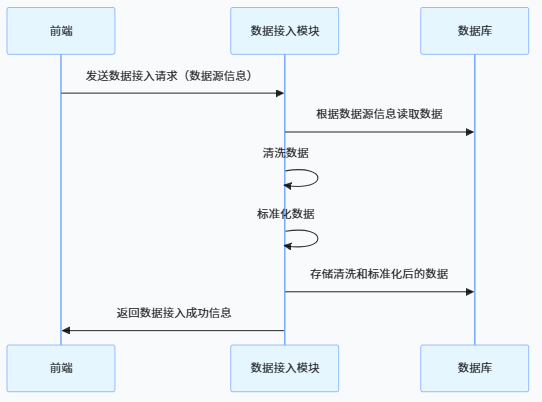


图2：数据接入时序图

代码：

sequenceDiagram

participant Frontend as 前端

participant DataIngestion as 数据接入模块

participant Database as 数据库

Frontend->>DataIngestion: 发送数据接入请求（数据源信息）

DataIngestion->>Database: 根据数据源信息读取数据

DataIngestion->>DataIngestion: 清洗数据

DataIngestion->>DataIngestion: 标准化数据

DataIngestion->>Database: 存储清洗和标准化后的数据

DataIngestion->>Frontend: 返回数据接入成功信息

时序图说明：

前端发起数据接入请求后，数据接入模块从数据库读取数据，进行清洗和标准化处理，再将处理后的数据存储回数据库，并向前端返回成功信息。

****3.算法伪代码****

python

class DataIngestion:

def \_\_init\_\_(self):

self.data = None

def ingestData(self, source):

# 根据数据源类型读取数据，假设数据源为文件路径

if source.endswith('.csv'):

import pandas as pd

self.data = pd.read\_csv(source)

elif source.endswith('.json'):

import json

with open(source, 'r') as f:

self.data = json.load(f)

# 其他数据源类型可继续扩展

def cleanData(self):

if self.data is not None:

# 处理缺失值，例如填充缺失值为0

self.data.fillna(0, inplace=True)

# 处理异常值，假设通过3倍标准差法识别异常值

from scipy import stats

self.data = self.data[(np.abs(stats.zscore(self.data)) < 3).all(axis=1)]

def standardizeData(self):

if self.data is not None:

from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()

numerical\_cols = self.data.select\_dtypes(include=[np.number]).columns

self.data[numerical\_cols] = scaler.fit\_transform(self.data[numerical\_cols])

## 1.2 数据处理与分析模块

****1、类图****

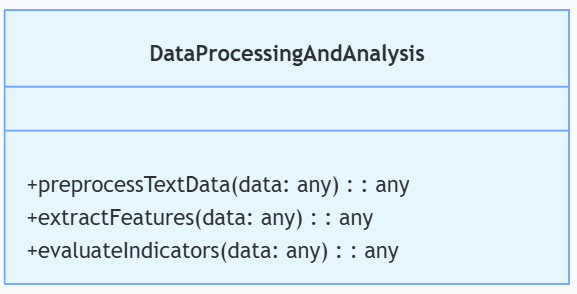


图3：数据处理与分析类图

代码：

classDiagram

class DataProcessingAndAnalysis {

+preprocessTextData(data: any): any

+extractFeatures(data: any): any

+evaluateIndicators(data: any): any

}

DataProcessingAndAnalysis类负责文本数据预处理、特征提取和指标评估。preprocessTextData方法对非结构化文本数据进行预处理，extractFeatures方法提取数据特征，evaluateIndicators方法基于机器学习算法评估指标影响度。

****2、时序图****

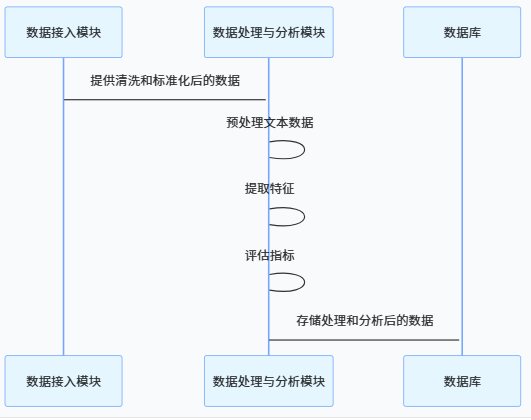


图4：数据处理与分析时序图

代码：

sequenceDiagram

participant DataIngestion as 数据接入模块

participant DataProcessingAndAnalysis as 数据处理与分析模块

participant Database as 数据库

DataIngestion->>DataProcessingAndAnalysis: 提供清洗和标准化后的数据

DataProcessingAndAnalysis->>DataProcessingAndAnalysis: 预处理文本数据

DataProcessingAndAnalysis->>DataProcessingAndAnalysis: 提取特征

DataProcessingAndAnalysis->>DataProcessingAndAnalysis: 评估指标

DataProcessingAndAnalysis->>Database: 存储处理和分析后的数据

时序图展示了数据接入模块将清洗和标准化后的数据传递给数据处理与分析模块，该模块依次进行文本数据预处理、特征提取和指标评估，最后将处理和分析后的数据存储到数据库的过程。

数据接入模块将清洗和标准化后的数据传递给数据处理与分析模块，该模块依次进行文本数据预处理、特征提取和指标评估，最后将处理和分析后的数据存储到数据库。

****3、算法伪代码****

python

import nltkfrom nltk.tokenize import word\_tokenizefrom nltk.corpus import stopwordsfrom sklearn.feature\_extraction.text import TfidfVectorizerfrom sklearn.model\_selection import train\_test\_splitfrom sklearn.ensemble import RandomForestClassifierfrom sklearn.metrics import accuracy\_score

class DataProcessingAndAnalysis:

def \_\_init\_\_(self):

nltk.download('punkt')

nltk.download('stopwords')

def preprocessTextData(self, data):

if isinstance(data, pd.Series):

data = data.apply(lambda x: word\_tokenize(str(x).lower()))

stop\_words = set(stopwords.words('english'))

data = data.apply(lambda x: [word for word in x if word not in stop\_words])

data = data.apply(lambda x: " ".join(x))

return data

def extractFeatures(self, data):

vectorizer = TfidfVectorizer()

features = vectorizer.fit\_transform(data)

return features

def evaluateIndicators(self, features):

X\_train, X\_test, y\_train, y\_test = train\_test\_split(features, y, test\_size=0.2, random\_state=42)

model = RandomForestClassifier()

model.fit(X\_train, y\_train)

y\_pred = model.predict(X\_test)

accuracy = accuracy\_score(y\_test, y\_pred)

return accuracy

## 1.3 风险分析模块

****1、类图****



图5：风险分析类图

代码

classDiagram

class RiskAnalysis {

+adjustWeights(data: any): any

+generateRiskIndicators(data: any): any

}

类图说明：

RiskAnalysis 类负责调整风险指标权重和生成风险指标。

adjustWeights 方法基于 SHAP 值分析调整指标权重。

generateRiskIndicators 方法根据调整后的权重生成风险指标

****2、时序图****

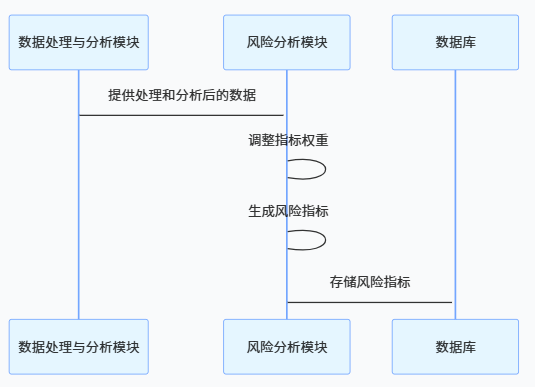


图6：风险分析时序图

代码：

sequenceDiagram

participant DataProcessingAndAnalysis as 数据处理与分析模块

participant RiskAnalysis as 风险分析模块

participant Database as 数据库

DataProcessingAndAnalysis->>RiskAnalysis: 提供处理和分析后的数据

RiskAnalysis->>RiskAnalysis: 调整指标权重

RiskAnalysis->>RiskAnalysis: 生成风险指标

RiskAnalysis->>Database: 存储风险指标

时序图说明：

数据处理与分析模块将处理和分析后的数据传递给风险分析模块，该模块进行指标权重调整和风险指标生成，最后将风险指标存储到数据库。

****3、算法伪代码****

python

import shapimport pandas as pdfrom sklearn.ensemble import GradientBoostingRegressor

class RiskAnalysis:

def adjustWeights(self, data):

X = data.drop('risk\_label', axis=1)

y = data['risk\_label']

model = GradientBoostingRegressor()

model.fit(X, y)

explainer = shap.Explainer(model)

shap\_values = explainer(X)

shap\_df = pd.DataFrame(shap\_values.values, columns=X.columns)

weights = shap\_df.abs().mean()

return weights

def generateRiskIndicators(self, weights):

risk\_indicators = []

for indicator, weight in weights.items():

risk\_indicator = {

'indicator': indicator,

'weight': weight

}

risk\_indicators.append(risk\_indicator)

return risk\_indicators

## 1.4 服务输出模块

****1、类图****

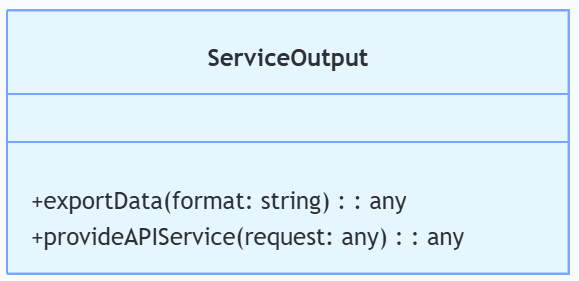


图7：服务输出类图

代码：

classDiagram

class ServiceOutput {

+exportData(format: string): any

+provideAPIService(request: any): any

}

类图说明：

ServiceOutput 类负责数据导出和提供 API 服务。

exportData 方法根据指定格式导出数据。

provideAPIService 方法处理 API 请求并返回响应。

****2、时序图****

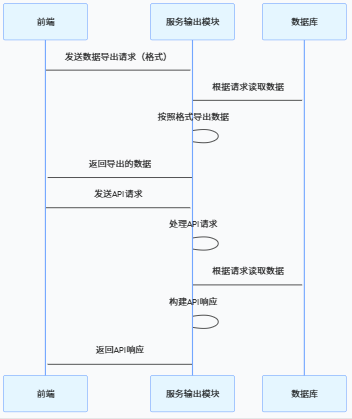


图8：服务输出时序图

代码：

sequenceDiagram

participant Frontend as 前端

participant ServiceOutput as 服务输出模块

participant Database as 数据库

Frontend->>ServiceOutput: 发送数据导出请求（格式）

ServiceOutput->>Database: 根据请求读取数据

ServiceOutput->>ServiceOutput: 按照格式导出数据

ServiceOutput->>Frontend: 返回导出的数据

Frontend->>ServiceOutput: 发送API请求

ServiceOutput->>ServiceOutput: 处理API请求

ServiceOutput->>Database: 根据请求读取数据

ServiceOutput->>ServiceOutput: 构建API响应

ServiceOutput->>Frontend: 返回API响应

时序图说明：

前端分别发起数据导出请求和 API 请求后，服务输出模块从数据库读取数据，进行相应处理并返回结果。

****3、算法伪代码****

python

import pandas as pd

class ServiceOutput:

def exportData(self, format):

data = pd.read\_sql('SELECT \* FROM risk\_indicators', self.db\_connection)

if format == 'csv':

return data.to\_csv(index=False)

elif format == 'json':

return data.to\_json(orient='records')

def provideAPIService(self, request):

# 假设请求包含查询条件

query = request.get('query')

data = pd.read\_sql(f"SELECT \* FROM risk\_indicators WHERE {query}", self.db\_connection)

response = {

'status':'success',

'data': data.to\_dict(orient='records')

}

return response

## 1.5 风险事件输出模块

1、类图

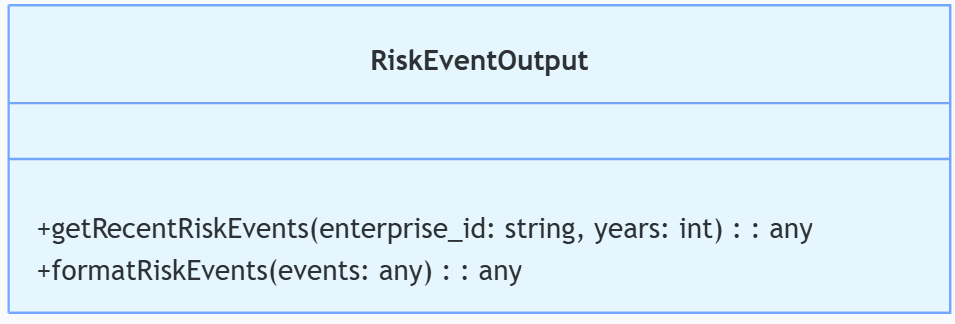


图9：风险事件输出类图

代码：

mermaid

classDiagram

class RiskEventOutput {

+getRecentRiskEvents(enterprise\_id: string, years: int): any

+formatRiskEvents(events: any): any

}

类图说明：

RiskEventOutput 类负责获取和格式化企业最近3年的重大风险事件。

getRecentRiskEvents 方法根据企业 ID 和年份范围从数据库中获取风险事件。

formatRiskEvents 方法将获取的风险事件格式化为前端所需的 JSON 格式。

1. 时序图

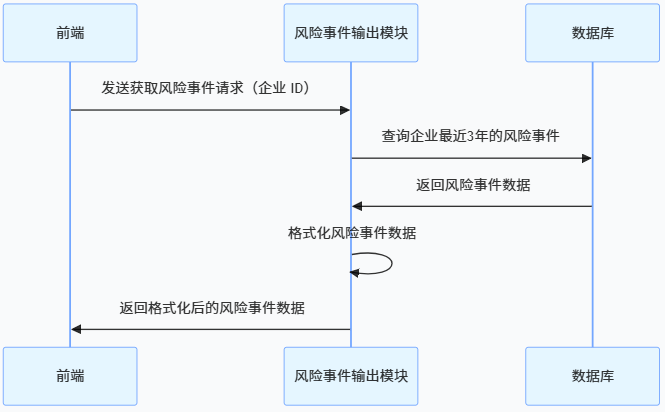


图10：风险事件输出时序图

代码：

sequenceDiagram

participant Frontend as 前端

participant RiskEventOutput as 风险事件输出模块

participant Database as 数据库

Frontend->>RiskEventOutput: 发送获取风险事件请求（企业 ID）

RiskEventOutput->>Database: 查询企业最近3年的风险事件

Database->>RiskEventOutput: 返回风险事件数据

RiskEventOutput->>RiskEventOutput: 格式化风险事件数据

RiskEventOutput->>Frontend: 返回格式化后的风险事件数据

时序图说明：

前端发起获取风险事件的请求后，风险事件输出模块从数据库中查询企业最近3年的风险事件，格式化后返回给前端。

1. 算法伪代码

import pandas as pd

class RiskEventOutput:

def getRecentRiskEvents(self, enterprise\_id, years=3):

# 查询企业最近3年的风险事件

query = f"""

SELECT \* FROM risk\_events

WHERE enterprise\_id = '{enterprise\_id}'

AND event\_date >= DATE\_SUB(CURDATE(), INTERVAL {years} YEAR)

"""

events = pd.read\_sql(query, self.db\_connection)

return events

def formatRiskEvents(self, events):

formatted\_events = []

for \_, row in events.iterrows():

formatted\_event = {

'event\_name': row['event\_name'],

'event\_description': row['event\_description'],

'event\_date': row['event\_date'],

'involved\_financial\_institutions': row['involved\_financial\_institutions'],

'involved\_funds': row['involved\_funds'],

'upstream\_supply\_chain': row['upstream\_supply\_chain'],

'downstream\_supply\_chain': row['downstream\_supply\_chain']

}

formatted\_events.append(formatted\_event)

return formatted\_events

## 2.3 获取企业风险事件接口

1、请求参数

|  |  |  |  |
| --- | --- | --- | --- |
| 参数名 | 类型 | 描述 | 是否必填 |
| enterprise\_id | string | 企业 ID | 是 |
| years | int | 查询最近多少年的风险事件，默认为3 | 否 |

2、响应格式（json）

{

"status": "success",

"data": [

{

"event\_name": "重大财务欺诈事件",

"event\_description": "企业涉及财务欺诈，导致股价大幅下跌。",

"event\_date": "2021-05-15",

"involved\_financial\_institutions": ["银行A", "银行B"],

"involved\_funds": "5000万",

"upstream\_supply\_chain": ["供应商X", "供应商Y"],

"downstream\_supply\_chain": ["客户M", "客户N"]

},

{

"event\_name": "供应链中断事件",

"event\_description": "由于自然灾害，供应链中断，导致生产停滞。",

"event\_date": "2022-08-20",

"involved\_financial\_institutions": ["银行C"],

"involved\_funds": "3000万",

"upstream\_supply\_chain": ["供应商Z"],

"downstream\_supply\_chain": ["客户P"]

}

]

}

3、错误码

|  |  |
| --- | --- |
| 错误码 | 描述 |
| 400 | 请求参数缺失或错误 |
| 500 | 服务器内部错误 |

## 3.3 风险事件表（risk\_events）

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 表名 | 字段名 | 字段类型 | 主键 | 外键 | 描述 |
| risk\_events | event\_id | int | 是 | 无 | 风险事件 ID |
| risk\_events | enterprise\_id | varchar(50) | 是 | enterprise\_info.enterprise\_id | 企业 ID |
| risk\_events | event\_name | varchar(200) | 否 | 无 | 事件名称 |
| risk\_events | event\_description | text | 否 | 无 | 事件简述 |
| risk\_events | event\_date | date | 否 | 无 | 事件发生日期 |
| risk\_events | involved\_financial\_institutions | text | 否 | 无 | 事件涉及的金融机构 |
| risk\_events | involved\_funds | varchar(100) | 否 | 无 | 事件涉及资金 |
| risk\_events | upstream\_supply\_chain | text | 否 | 无 | 事件涉及的供应链上游 |
| risk\_events | downstream\_supply\_chain | text | 否 | 无 | 事件涉及的供应链下游 |

## 4.3 风险事件查询算法

1、具体实现步骤

查询数据：根据企业 ID 和年份范围从数据库中查询风险事件。

格式化数据：将查询到的风险事件数据格式化为前端所需的 JSON 格式。

返回数据：将格式化后的风险事件数据返回给前端。

2、参数设置

查询范围：默认查询最近3年的风险事件，支持自定义年份范围。

数据格式化：确保返回的数据包含事件名称、简述、发生日期、涉及的金融机构、资金、供应链上下游等信息。

# 重点算法实现

## 2.1 SHAP 值分析

****1、具体实现步骤****

准备数据：从数据库中获取包含风险指标和风险标签的数据集，将其分为特征矩阵 X 和标签向量 y。

选择模型：使用 GradientBoostingRegressor 等模型进行训练，该模型可以较好地捕捉数据中的复杂关系。

初始化解释器：使用shap.Explainer初始化解释器，传入训练好的模型。

计算 SHAP 值：使用解释器对特征矩阵 X 进行计算，得到每个特征的 SHAP 值。

分析结果：根据 SHAP 值的绝对值大小确定每个特征（风险指标）对模型输出（风险评估结果）的影响程度，进而调整风险指标的权重。

****2、参数设置****

GradientBoostingRegressor模型参数：n\_estimators=100，表示模型中基学习器的数量；learning\_rate=0.1，控制每次迭代时步长；max\_depth=3，限制决策树的最大深度，防止过拟合。

shap.Explainer参数：默认参数即可，其会根据传入的模型自动选择合适的计算方法。

### **2.2 LIME 算法**

****1、具体实现步骤****

数据准备：同 SHAP 值分析，准备好特征矩阵 X 和标签向量 y。

训练模型：选择需要解释的模型，如逻辑回归模型。

初始化 LIME 解释器：使用lime.lime\_tabular.LimeTabularExplainer初始化解释器，传入特征矩阵 X 等参数。

选择实例进行解释：从数据集中选择一个或多个需要解释的实例。

计算解释：使用解释器对选择的实例进行解释，得到每个特征的重要性得分。

结果分析：根据重要性得分，评估每个风险指标对模型预测结果的影响，可用于优化风险指标体系。

****2、参数设置****

lime.lime\_tabular.LimeTabularExplainer参数：feature\_names=X.columns，指定特征名称；class\_names=['negative', 'positive']，指定类别名称；kernel\_width=3，控制核函数的带宽，影响解释的平滑度。

# 接口定义

### **3.1 获取风险指标接口**

**1、请求参数**

|  |  |  |  |
| --- | --- | --- | --- |
| 参数名 | 类型 | 描述 | 是否必填 |
| enterprise\_id | string | 企业 ID | 是 |
| risk\_type | string | 风险类型 | 是 |

2、相应格式：

json

{

"status": "success",

"data": [

{

"indicator": "资产负债率",

"weight": 0.35,

"description": "反映企业长期偿债能力的指标"

},

{

"indicator": "流动比率",

"weight": 0.28,

"description": "衡量企业流动资产在短期债务到期以前，可以变为现金用于偿还负债的能力"

}

]}

1. ****错误码****

|  |  |
| --- | --- |
| 错误码 | 描述 |
| 400 | 请求参数缺失或错误 |
| 500 | 服务器内部错误 |

## 3.2导入知识库接口

****1、请求参数****

|  |  |  |  |
| --- | --- | --- | --- |
| 参数名 | 类型 | 描述 | 是否必填 |
| file | file | 需要导入的文件 | 是 |
| vector\_db | boolean | 是否导入向量数据库，默认为 false | 否 |

****2、响应格式****

json

{

"status": "success",

"message": "知识库导入成功"}

****3、错误码****

|  |  |
| --- | --- |
| 错误码 | 描述 |
| 400 | 请求参数缺失或错误，如未上传文件 |
| 500 | 服务器内部错误，如文件导入失败 |

# 数据库设计

## 4.1 企业信息表（enterprise\_info）

| **表名** | **字段名** | **字段类型** | **主键** | **外键** | **描述** |
| --- | --- | --- | --- | --- | --- |
| enterprise\_info | enterprise\_id | varchar(50) | 是 | 无 | 企业 ID |
| enterprise\_info | enterprise\_name | varchar(200) | 否 | 无 | 企业名称 |
| enterprise\_info | industry | varchar(100) | 否 | 无 | 企业所属行业 |
| enterprise\_info | registration\_date | date | 否 | 无 | 企业注册日期 |

## 4.2 风险指标表（risk\_indicators）

| **表名** | **字段名** | **字段类型** | **主键** | **外键** | **描述** |
| --- | --- | --- | --- | --- | --- |
| risk\_indicators | indicator\_id | int | 是 | 无 | 风险指标 ID |
| risk\_indicators | enterprise\_id | varchar(50) | 是 | enterprise\_info.enterprise\_id | 企业 ID |
| risk\_indicators | indicator\_name | varchar(100) | 否 | 无 | 风险指标名称 |
| risk\_indicators | weight | decimal(5,2) | 否 | 无 | 风险指标权重 |
| risk\_indicators | description | text | 否 | 无 | 风险指标描述 |

### **（三）知识库表（knowledge\_base）**

| **表名** | **字段名** | **字段类型** | **主键** | **外键** | **描述** |
| --- | --- | --- | --- | --- | --- |
| knowledge\_base | kb\_id | int | 是 | 无 | 知识库 ID |
| knowledge\_base | file\_path | varchar(200) | 否 | 无 | 知识库文件路径 |
| knowledge\_base | vector\_db\_flag | boolean | 否 | 无 | 是否导入向量数据库标识 |
| knowledge\_base | import\_date | datetime | 否 | 无 | 导入日期 |

# 5. 前端模块设计

## 5.1 企业选择界面

### 5.1.1 交互逻辑

输入联想：用户输入企业名称时，实时调用大模型接口进行模糊匹配（最小输入长度 3 字符）。

自动补全：根据大模型返回的建议列表，支持鼠标点击或键盘方向键选择企业。

### 5.1.2 React 组件示例

/ EnterpriseSelector.jsximport { useState, useEffect } from 'react';

function EnterpriseSelector() {

const [inputValue, setInputValue] = useState('');

const [suggestions, setSuggestions] = useState([]);

useEffect(() => {

if (inputValue.length >= 3) {

fetch(`/api/enterprise-suggest?query=${encodeURIComponent(inputValue)}`)

.then(res => res.json())

.then(data => setSuggestions(data.suggestions));

}

}, [inputValue]);

const handleSelect = (enterpriseId) => {

// 调用后端接口确认企业并跳转至风险分析页面

window.location.href = `/risk-analysis?enterprise\_id=${enterpriseId}`;

};

return (

<div className="enterprise-selector">

<input

type="text"

value={inputValue}

onChange={(e) => setInputValue(e.target.value)}

placeholder="输入企业名称..."

/>

<ul className="suggestion-list">

{suggestions.map(suggestion => (

<li

key={suggestion.enterprise\_id}

onClick={() => handleSelect(suggestion.enterprise\_id)}

>

{suggestion.enterprise\_name} ({suggestion.industry})

</li>

))}

</ul>

</div>

);}

## 5.2 资源配置界面

### 5.2.1 文件上传组件

支持格式：PDF、Word、CSV（通过input[type="file"]实现）。

校验规则：文件大小≤50MB，类型通过accept属性限制。

### 5.2.2 URL 输入校验

/ 正则表达式验证URL格式const validateUrl = (url) => {

const pattern = /^(https?:\/\/)?([\da-z.-]+)\.([a-z.]{2,6})([/\w .-]\*)\*\/?$/;

return pattern.test(url);};

# 6. 大模型交互设计

## 6.1 Prompt 窗口设计

### 6.1.1 标准 Prompt 模板

prompt\_template = f"""

分析企业{enterprise\_name}的金融风险，重点关注以下维度：

财务指标：{financial\_metrics}

市场环境：{market\_conditions}

政策合规：{regulatory\_compliance}

输出JSON格式的风险指标权重，包含指标名称、权重和描述。

用户上传文件 → 调用大模型解析内容 → 提取风险关键词 → 存入向量数据库。

支持增量更新：检测文件哈希值，避免重复导入。

# 7. 安全与权限设计

## 7.1 用户认证系统

### 7.1.1 JWT 令牌生成

# Flask后端示例import jwtfrom datetime import datetime, timedelta

def generate\_token(user\_id):

payload = {

'user\_id': user\_id,

'exp': datetime.utcnow() + timedelta(hours=24)

}

return jwt.encode(payload, app.config['SECRET\_KEY'], algorithm='HS256')

### 7.1.2 权限控制

# 使用Flask-Login实现RBACfrom flask\_login import current\_user

@login\_required@roles\_required('admin')def admin\_dashboard():

return render\_template('admin.html')

## 7.2 数据加密

传输层：强制使用 TLS 1.3 加密 HTTP 通信。

存储层：对企业 ID、身份证号等敏感字段使用 AES-256 加密。

# 8. 性能优化设计

## 8.1 缓存策略

# Redis缓存示例from redis import Redis

cache = Redis(host='localhost', port=6379)

def get\_risk\_indicators(enterprise\_id):

key = f"risk\_indicators:{enterprise\_id}"

cached\_data = cache.get(key)

if cached\_data:

return json.loads(cached\_data)

# 否则查询数据库并缓存

data = db.query(RiskIndicators).filter\_by(enterprise\_id=enterprise\_id).all()

cache.setex(key, 3600, json.dumps(data))

return data

## 8.2 数据库优化

索引策略：为enterprise\_info.enterprise\_name添加全文索引以加速模糊查询。

读写分离：使用主从数据库架构，主库写、从库读。