Advances in Computer, Signals and Systems (2023) **DOI: 10.23977/acss.2023.070815**
Clausius Scientific Press, Canada **ISSN 2371-8838 Vol. 7 Num. 8**

Research on the Application of Multi-Sensor Data Acquisition Technology in the Internet of Things (IoT) Field

Liu Fangshu

Zibo Vocational Institute, Zibo, Shandong, China

Keywords: Multi-sensor, data acquisition, real-time monitoring, historical trajectory, GPS positioning

Abstract: This article begins by researching and developing middleware for multi-sensor data management in the Internet of Things (IoT). This middleware enables functions such as session management, data parsing, command feedback, and data caching, exhibiting adaptability and scalability. Subsequently, this middleware is applied to create a GPS device management system that includes features such as device management, real-time data monitoring, and historical data monitoring. The system addresses issues related to the stability of data collection and the accuracy of data parsing with a large number of sensor connections. Furthermore, the system can simultaneously support multiple devices for a single type of business or different devices for various businesses.

The Internet of Things (IoT) refers to a vast network created by connecting various information sensing devices such as sensors, radio-frequency identification (RFID) technology, global positioning systems, infrared sensors, laser scanners, gas sensors, and other devices and technologies. These devices are used to collect real-time information from any object or process that requires monitoring, connectivity, and interaction. They collect various types of information, including sound, light, heat, electricity, mechanics, chemistry, biology, location, and more, which are needed for seamless integration with the Internet. The goal is to enable the connection of things to things, things to people, and all objects to the network, making it convenient for identification, management, and control[1].

From a technical architecture perspective, the Internet of Things can be divided into three layers: the perception layer, the network layer, and the application layer. Traditional development for data collection from multiple sensors would require the development of a device management system for each type of sensor, involving repetitive work at the lower levels. This paper explores an architecture for multi-sensor data management that operates at the application layer and uses the TCP/IP protocol of the network layer to obtain information from the perception layer. It is compatible with different communication methods of sensors in the perception layer and provides interfaces for business expansion at the application layer. With simple business extensions, it can efficiently manage devices and data for various sensors, effectively addressing core technical challenges in data collection and device identification and management in the Internet of Things.

1. Analysis and Design of Multi-Sensor Data Management Middleware

Multi-sensor data management middleware must address communication between the system and various sensors. Users can extend the functionality by extending business processing classes to retrieve and process data. It should shield the issues that the TCP server needs to handle, such as memory pools, connection states, and connection context management. It encapsulates various events related to device connections, making network layer communication transparent to users, allowing them to focus on business development[2].

1.1 Functions of Multi-Sensor Data Management Middleware

Multi-sensor data management middleware implements common functions for multi-sensor data collection. These functions include session management, data analysis, data caching, command feedback, and periodic cycling. These functions are generic in the server and provide extensible interfaces.

The process of multi-sensor data collection primarily involves sensor devices connecting to the server. The system assigns a context class to each device to record its information and allocate memory. It then waits for the device to send data. When data is received, the system traverses parsers to find the appropriate parser for data interpretation. Upon finding a suitable parser, the context obtains its reference for future data calls. Unverified devices generate authentication information, mainly to obtain their serial numbers, which triggers an authentication event. The upper-level processor retrieves this authentication information and returns the authentication result. Successfully authenticated devices generate data packets and trigger data reception events, which are handed over to upper-level processors for business processing.

1.2 Devices that fail authentication are disconnected

(1)There are also three exceptional processes: 1) Devices timing out without authentication result get disconnected by the server. 2) Devices timing out without sending data get disconnected. 3) Devices connecting and finding an existing connection with the same identifier lead to the server disconnecting the original connection and assigning the original context to the new connection.

The flow of data collection and data return interaction. External sensors send data to the system as byte streams. The data parser is obtained based on the user context when data first arrives, and it confirms which parser to use through validation[3].

(2) Command Feedback To manage devices, the client must also be able to receive user messages forwarded by the server. Therefore, the system reserves an interface for data feedback. Upper-level processors can send commands to the collector, and commands are routed to the appropriate socket by device ID and sent to the client asynchronously [4].

(3) Data Caching Mechanism After device data is processed at the business level, it is typically written to a database for future trajectory queries, data statistics, or even data mining.

(4) Assuming sensor devices send data to the server once per second (a test scenario; normal devices generally do not transmit this frequently), it would result in 86,400 data entries per day. Large-scale data writing to the database becomes the system's main performance bottleneck. Database read/write capability is typically a key indicator for evaluating system performance. Optimizing database read/write capabilities can be approached from various angles. Regarding reading, indexing is the primary focus, and it is designed based on the business needs [5]. On the writing side, this system uses SQL Server's SqlBulkCopy data import technology. Microsoft SQL Server provides a popular command-line utility called bcp for moving data from one table to another (tables can be on the same server or different servers). SqlBulkCopy allows developers to

implement a managed code solution with similar functionality. Other methods for loading data into SQL Server tables exist (e.g., INSERT statements), but SqlBulkCopy offers significant performance advantages when dealing with large datasets. Using the SqlBulkCopy class can only write data to SQL Server tables[6]. However, data sources are not limited to SQL Server; any data source can be used as long as data can be loaded into a DataTable instance or read using an IDataReader instance. Experiments have shown that using SqlBulkCopy for data insertion is 36 times faster than conventional insertion methods. Of course, this efficiency advantage is evident only in the case of large data volumes, which is precisely the situation in a sensor management system. Multiple devices correspond to one table, which, although advantageous in many ways, also presents a problem. The system operates using asynchronous mechanisms, and when multiple threads from different devices simultaneously write data to the same database table, it can lead to a pseudo-lock state. This directly results in timeouts, data loss, and even program crashes. To address this issue, the data cache module establishes a cache queue and uses a unique data writing thread to write data to the database. This resolves conflicts effectively.

(5) Periodic Cycling In practical applications, some devices are in a passive state and must be actively queried by the server for data. In such cases, the server must set a timer to periodically request data from the devices. The periodic cycling mechanism encapsulates the Timer class of the system, implementing timed cycles based on a thread pool. Additionally, it has references to device processors, allowing them to access processor operations. Please note that this translation is a direct conversion of the provided text, and some sentences may require further refinement for clarity and readability in English.

2. GPS Location System Analysis and Design

2.1 GPS Location System Function Analysis and Design

The GPS location system primarily implements two functions: device management and device data management. To achieve these functions, this paper uses a multi-sensor data management middleware to address communication with devices, device data processing, compatibility with various device protocols, and then modifies protocol parsers and business processors to create an efficient and stable GPS location system.

Device management includes functions such as adding, deleting, modifying, and querying devices. The processor for newly added devices is inserted into a processor container for management. Data management functions include real-time monitoring and historical trajectory playback [7].

(1) Data Collection Function: The data collection function is mainly to connect to locators produced by different manufacturers. The multi-sensor data management module's collection module parses the data into a unified format for business processing and database data writing.

After device data is processed by MSD, it becomes a data packet handed over to MSD's business module for processing. The business module keeps real-time data for devices in memory and writes historical data to the database. The GPS location interface interacts with MSD and the database to retrieve and modify data.

(2) Command Sending Function: Managers sometimes need to send commands to locator devices, such as automatic monitoring setup commands, clear alarm commands, mileage query commands, etc. Since locator devices are already connected to the data collection system, commands can only be sent to locator devices through the TCP channel between locator devices and the data collection system. The data collection system receives a command from the client, finds the corresponding locator device's TCP channel based on the device's serial number in the command, and then sends the command to that locator device [8].

(3) Real-Time Monitoring: Devices upload data to the server approximately every 30 seconds. Besides writing the data to the database, the latest data is also stored in memory for quick queries of the device's current status without the need to search the database. When users want to query the current location of a device, they can select the device's location function to obtain the device's current position.

(4) Historical Trajectory Playback: Users select a time period, download historical data, and the system reads historical data from the database for that time period, calculates stop times and mileage, and returns the data to the client. After receiving the data, the client can play back the historical trajectory.

2.2 GPS Location System Overall Architecture

The device data management module manages devices and interacts with them. MSD writes data to the database using ADO.NET.

The Service interface implemented using WCF technology provides interfaces for operations on devices, MSD servers, and device data. When users need to retrieve real-time data from devices, they can obtain the device's processor from MSD and retrieve its real-time data. When users need to retrieve historical data, they can directly query the database through the data layer module constructed using ADO.NET.

The website implemented using ASP.NET is intended to provide a browser-based user interface. It acts as a presentation layer and retrieves data from WCFService through service proxies. Data is presented to the client using AJAX for both data and maps.

This system calls the multi-sensor data management middleware to develop the GPS location system and must complete the following two development steps.

(1) Development of Parsers: The system can dynamically insert parsers, with each parser corresponding to a data encoding format. The system defines an interface for parsers, and parser classes need only inherit this interface to implement their data parsing functions to complete data parsing.

(2) Development of Business Processing Classes: The business processing base class includes event functions related to device connection, device authentication, device data reception, and device disconnection. Users can achieve various business extensions by overriding these functions.

3. GPS Location System Implementation

Implementation of Device Management Function shows the main interface for device management, with an operation menu on the left divided into device information and device data.

Implementation of Device Data Management Device data includes a map bar, a historical data operation bar, and a historical data list bar.

Implementation of Historical Trajectory Playback Function Open the historical data operation bar, select a time period, download historical data through AJAX, and play back the historical trajectory. Historical data can be viewed in the historical data list bar, and clicking play will draw the historical trajectory on the map [9].

4. Testing and Performance Comparison

Due to the large number of devices, software-simulated clients are used, which read device data from the database, generate data for different devices, and send it to the server.

With 4,000 sensor connections at a rate of approximately 3,666 messages per second, all connections and authentication processes are normal, and all data is received intact. The program runs stably, and system resource usage is minimal. Similar systems can typically handle only 2,000 sensor connections under stable conditions. This advantage is achieved in this system through the use of SocketAsyncEvent [10].

5. Args and thread pool technology.

CPU fluctuations mainly result from database writing. Since this is simulated data, there are peaks and valleys in the data.

Innovation and Significance This system overcomes the two major challenges in data processing in TCP servers.

(1) Handling Data Concurrency Issues for a Large Number of Connections In traditional server models, one thread is used to handle one connection. This quickly depletes system resources, and a large number of system resources remain idle. Therefore, SocketAsyncEventArgs and thread pool technology are used here to achieve maximum data processing with fewer threads. The server's performance is also excellent compared to similar servers [11].

(2) Solving Slow Data Writing to Databases After the connection is established, the system's primary performance consumption occurs during database writing. Here, the system cleverly uses SQL database table copying functionality by creating a table in memory to cache data. It is then batch-copied to the database, significantly reducing the performance cost of database writing and allowing the server to achieve the fastest data writing speed.

References

[1] Rusinkiewicz M, Sheth A. Specification and execution of transactional workflows [M]. Addison Wesley Publishing Company, 1995.
[2] Jiang Jinnan. WCF technical analysis [M]. Beijing: Electronic Industry Press, 2009.
[3] Nicolai M, Josuttis. SOA in practice [M]. O'Reilly, 2007.
[4] Alonso G, Agrawal D, Abbadi Ea, et al. Functionality and limitations of current workflow management systems [M]. 1997.
[5] Peng Renkui. EXT JS source code analysis and development instance[M]. Beijing: Electronic Industry Press, 2009.
[6] Zhang Wei. The design and research of software requirement? Management system for Guangdong Post[C]// Proceedings of 2010 International Conference on Management Science and Engineering, 2009.
[7] Zhang Wei, Feng Hua. The design and research of development architecture for medium or small information system[C]// 2009 International Conference on Management Science and Engineering, 2009.
[8] Liu Qiang. Key technology of internet of things and application [J]. Computer Science, 2010, 37 (6): 1-10.
[9] Wang Baoyun. Summary of things research [J]. Electronic Measurement and Instrument, 2009, 23 (12): 1-6.
[10] Shen Subin. On things architecture and related technologies [J]. Nanjing University of Posts and Telecommunications: Natural Science, 2009, 29(6): 1-2.
[11] Xu Diwei. Internet of things and its application analysis [J]. Computer Engineering and Applications, 2011, 47(15): 1-4.